

Alexandre Colombo

Abordagem Paralela para Mineração de Regras de Associação Negativas

Bauru, São Paulo, Brasil

Março 2021

Alexandre Colombo

Abordagem Paralela para Mineração de Regras de Associação Negativas

Monografia elaborada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”.

Universidade Estadual Paulista “Júlio de Mesquita Filho” – UNESP

Instituto de Biociências, Letras e Ciências Exatas

Programa de Pós-Graduação em Ciência da Computação

Orientador: Profa. Associada Roberta Spolon

Bauru, São Paulo, Brasil

Março 2021

C718a

Colombo, Alexandre

Abordagem Paralela para Mineração de Regras de Associação
Negativas / Alexandre Colombo. -- Bauru, 2021

89 p. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp),
Faculdade de Ciências, Bauru

Orientadora: Roberta Spolon

1. Ciência da computação. 2. Mineração de dados (Computação). 3.
Algoritmos paralelos. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de
Ciências, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

ATA DA DEFESA PÚBLICA DA DISSERTAÇÃO DE MESTRADO DE ALEXANDRE COLOMBO, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO, DA FACULDADE DE CIÊNCIAS - CÂMPUS DE BAURU.

Aos 15 dias do mês de março do ano de 2021, às 14:00 horas, por meio de Videoconferência, realizou-se a defesa de DISSERTAÇÃO DE MESTRADO de ALEXANDRE COLOMBO, intitulada **Abordagem Paralela para Mineração de Regras de Associação Negativas**. A Comissão Examinadora foi constituída pelos seguintes membros: Professora Associada ROBERTA SPOLON (Orientador(a) - Participação Virtual) do(a) Departamento de Computação da Faculdade de Ciências da Unesp - Câmpus de Bauru / Universidade Estadual Paulista, Profa. Dra SARITA MAZZINI BRUSCHI (Participação Virtual) do(a) Departamento de Sistemas de Computação. / Universidade de São Paulo- Instituto de Ciências Matemáticas e de Computação, Prof. Dr. RODRIGO CAPOBIANCO GUIDO (Participação Virtual) do(a) Departamento de Ciências de Computação e Estatística / Instituto de Biociências, Letras e Ciências Exatas. Após a exposição pelo mestrando e arguição pelos membros da Comissão Examinadora que participaram do ato, de forma presencial/ou virtual, o discente recebeu o conceito final **APROVADO**. Nada mais havendo, foi lavrada a presente ata, que após lida e aprovada, foi assinada pelo(a) Presidente(a) da Comissão Examinadora.

Professora Associada ROBERTA SPOLON



Agradecimentos

Agradeço primeiramente a Deus por me ajudar nesta longa caminhada, que está completando mais um ciclo.

Agradeço também à Profa. Dra. Roberta Spolon pela orientação, conselhos, oportunidades e experiências que pude viver neste período. À meu pai, José Roberto, minha mãe, Sandra, e aos meus irmãos Júnior e Ana Cláudia pela companhia e por todo o apoio. Aos meus colegas de trabalho, com os quais pude compartilhar vasto conhecimento que levarei para toda a vida.

Deixo um agradecimento ao Núcleo de Computação Científica (NCC/GridUNESP) da Universidade Estadual Paulista (UNESP), que por meio da disponibilização dos recursos computacionais tornou possível esta pesquisa.

Dedico também um agradecimento especial à família Nishimura e à empresa Máquinas Agrícolas Jacto, pelo suporte e incentivo em estar aprofundando meus estudos.

E por fim agradeço à Géssica, minha esposa, que durante vários anos sempre me apoiou e deu forças para superar as dificuldades.

“Ninguém cresce sozinho.”
(Shunji Nishimura)

Resumo

Mineração de padrões frequentes e regras de associação são um dos principais campos de pesquisa em Mineração de Dados, que apresenta o objetivo de determinar relações consistentes entre elementos. Algoritmos existentes neste campo de estudo se baseiam principalmente na informação de ocorrência dos elementos.

Entretanto, considerar a ausência de elementos para a geração de regras pode resultar em associações de grande interesse para algumas aplicações, que poderá fornecer conhecimento até então desconhecido para o cientista de dados ou analista. Este tipo de associação é denominada regra de associação negativa, e a sua principal característica é a explosão da quantidade de regras geradas, que demanda uma capacidade computacional adequada para seu processamento.

Neste projeto de mestrado foram exploradas diversas abordagens, e proposto um método que apresenta como principais objetivos acelerar o processo de geração de regras, e permitir que conjuntos de dados maiores possam ser minerados.

Considerando a etapa de identificação de conjuntos frequentes, este método dispõe de quatro abordagens que exploram plataformas paralelas de computação. Estas apresentam destaque em situações específicas, de forma que a depender do conjuntos de dados a ser analisado, será recomendada o uso de uma destas. Na etapa de geração de regras do método proposto também são exploradas plataformas paralelas.

Através dos resultados obtidos foi possível verificar que o método alcança os objetivos propostos. Além disso, o método desenvolvido permite minerar conjuntos de dados grandes que são considerados restritivos para implementações existentes. Por fim, foi constatado que o método desenvolvido é escalável, permitindo melhorar seu desempenho com o incremento de recursos computacionais.

Palavras-chaves: mineração de dados. padrões frequentes. regras de associação negativas. algoritmos paralelos.

Abstract

Frequent pattern mining and association rules are one of the main fields of research in Data Mining, which aims to identify consistent relationships among elements. Existing algorithms in this field of study are based mainly on the occurrence information of the elements.

However, considering the absence of elements for rules generation may result in interesting associations for some applications, which may provide previously unknown knowledge to data scientists or analysts. This type of association is called negative association rule, and its main characteristic is the explosion of the number of generated rules which demands adequate computational capacity for its processing.

In this work, several approaches were explored, and a method was proposed. Such method presents as main objectives to accelerate the rule generation process and to allow the mining of larger datasets.

Considering the stage of identifying frequent sets, the proposed method implements four approaches that explore parallel computing platforms. Such approaches presents better performance in specific situations, so that depending on the datasets to be analyzed, the use of one of these will be recommended. The rule generation stage of the proposed method also explores parallel platforms.

Through the obtained results it was possible to verify that the method reaches the proposed objectives. In addition, the developed method enables mining large datasets that are considered restrictive for existing implementations. Finally, the proposed method is scalable, which allows to improve its performance through increasing computational resources.

Keywords: data mining. frequent patterns. negative association rules. parallel algorithm.

Lista de ilustrações

Figura 1	– Exemplo da operação do método Apriori. Nós mais escurecidos indicam candidatos identificados como não frequentes, e nós e vértices pontilhados indicam elementos eliminados pelo mecanismo de poda. Fonte: Adaptado de (ZAKI; JR, 2014).	22
Figura 2	– Exemplo de estrutura FP-tree para o banco de dados exemplo. Fonte: Adaptado de (ZAKI; JR, 2014).	23
Figura 3	– Exemplo de estrutura bitmap de representação de dados. Fonte: Adaptado de (FANG et al., 2009).	23
Figura 4	– Exemplo de cálculo de linhas bitmap relativas à candidatos com cardinalidade superior por meio de operações booleanas. Fonte: Adaptado de (FANG et al., 2009).	23
Figura 5	– Exemplo de geração de regras de associação. Fonte: Adaptado de (FANG et al., 2009).	26
Figura 6	– Exemplo de árvore de prefixos. Fonte: Criado pelo autor.	30
Figura 7	– Geração de candidatos em árvore de prefixos (números nos nós indicam a iteração em que o candidato é gerado). Método Apriori realiza a geração em largura (a), demandando um uso maior de memória porém estando mais apto à paralelização. Método Eclat gera candidatos em profundidade (b), estando menos apto à paralelização porém apresentando a menor demanda de uso de memória. Método proposto (c) irá atuar de uma forma balanceada, avançando de forma conjunta em largura e profundidade. Fonte: Criado pelo autor.	31
Figura 8	– Estratégia de cálculo da soma de bits 1s em que cada <i>thread</i> faz a análise de um trecho da linha bitmap. Esta estratégia é mais adequada para o caso em que as linhas da representação bitmap sejam muito extensas. Fonte: Criado pelo autor.	32
Figura 9	– Estratégia de cálculo da soma de bits 1s em que cada <i>thread</i> faz a análise completa de uma linha bitmap de um conjunto candidato. Esta estratégia é mais adequada para o caso em que existam muitos candidatos. Fonte: Criado pelo autor.	32
Figura 10	– Esquema lógico da implementação Apriori-Bitmap-Cuda. A identificação de conjuntos frequentes é feita executando o cálculo do valor de suporte em GPU de maneira dinâmica. Fonte: Criado pelo autor.	34
Figura 11	– Algoritmo Apriori-Bitmap-Cuda.	35
Figura 12	– Algoritmo Apriori-Simd-Parallel.	36

Figura 13 – Algoritmo Apriori-Roaring-Paralelo.	37
Figura 14 – Esquema lógico da implementação Apriori-Cuda-Single-Memcpy, em que a identificação de conjuntos frequentes é feita executando o cálculo do valor de suporte em GPU realizando apenas uma única transferência de estrutura bitmap. Fonte: Criado pelo autor.	38
Figura 15 – Algoritmo Apriori-Cuda-Single-Memcpy.	39
Figura 16 – Partição do processo de mineração de regras de associação em quatro partes. Fonte: Adaptado de (CORNELIS et al., 2006).	41
Figura 17 – Algoritmo para obtenção dos conjuntos frequentes da terceira parte do método PNAR. Fonte: Adaptado de (CORNELIS et al., 2006).	42
Figura 18 – Algoritmo para obtenção dos conjuntos frequentes da quarta parte do método PNAR. Fonte: Adaptado de (CORNELIS et al., 2006).	44
Figura 19 – Algoritmo para geração de regras de associação do método PNAR. Fonte: Adaptado de (CORNELIS et al., 2006).	45
Figura 20 – Análise de pico de uso de memória das implementações do método de identificação de conjuntos frequentes para conjuntos de dados esparsos.	52
Figura 21 – Análise de pico de uso de memória das implementações do método de identificação de conjuntos frequentes com o comparativo das implementações de Borgelt para conjuntos de dados esparsos.	53
Figura 22 – Análise de pico de uso de memória das implementações do método de identificação de conjuntos frequentes para conjuntos de dados densos.	54
Figura 23 – Análise de pico de uso de memória das implementações do método de identificação de conjuntos frequentes com o comparativo das implementações de Borgelt para conjuntos de dados densos.	55
Figura 24 – Análise de tempo de execução das implementações do método de identificação de conjuntos frequentes para conjuntos de dados esparsos.	57
Figura 25 – Análise de tempo de execução das implementações do método de identificação de conjuntos frequentes com o comparativo das implementações de Borgelt para conjuntos de dados esparsos.	58
Figura 26 – Análise de tempo de execução das implementações do método de identificação de conjuntos frequentes para conjuntos de dados densos.	59
Figura 27 – Análise de tempo de execução das implementações do método de identificação de conjuntos frequentes com o comparativo das implementações de Borgelt para conjuntos de dados densos.	60
Figura 28 – Análise de escalabilidade da implementação Apriori-Roaring-Parallel.	61
Figura 29 – Análise de escalabilidade da implementação Apriori-Simd-Parallel.	62
Figura 30 – Análise de pico de uso de memória do método desenvolvido de identificação de regras de associação negativas utilizando conjuntos de dados do repositório FIMI.	64

Figura 31 – Análise de pico de uso de memória do método desenvolvido de identificação de regras de associação negativas utilizando conjuntos de dados sintéticos.	65
Figura 32 – Análise de participação das etapas três e quatro do método desenvolvido na composição do tempo de execução, considerando o conjunto de dados accidents e valores de suporte mínimo e confiança mínima variando de 20% à 40%.	65
Figura 33 – Análise de tempo de execução do método desenvolvido de identificação de regras de associação negativas utilizando conjuntos de dados sintéticos.	66
Figura 34 – Análise de tempo de execução do método desenvolvido de identificação de regras de associação negativas utilizando conjuntos de dados do repositório FIMI.	67
Figura 35 – Análise de quantidade de regras geradas pelo método desenvolvido utilizando conjuntos de dados do repositório FIMI, representando valores totais.	69
Figura 36 – Análise de quantidade de regras geradas pelo método desenvolvido utilizando conjuntos de dados do repositório FIMI, representando contribuição de partes do método.	70
Figura 37 – Análise de quantidade de regras geradas pelo método desenvolvido utilizando conjuntos de dados sintéticos, representando valores totais.	71
Figura 38 – Análise de quantidade de regras geradas pelo método desenvolvido utilizando conjuntos de dados sintéticos, representando contribuição de partes do método.	72
Figura 39 – Análise de escalabilidade do método desenvolvido utilizando conjuntos de dados do repositório FIMI.	74
Figura 40 – Análise de escalabilidade do método desenvolvido utilizando conjuntos de dados sintéticos, com representação discriminada.	75
Figura 41 – Análise de escalabilidade do método desenvolvido utilizando conjuntos de dados sintéticos, com representação agrupada.	76

Lista de tabelas

Tabela 1 – Estrutura de banco de dados de transação. Fonte: (ZAKI; JR, 2014).	21
Tabela 2 – Representação de banco de dados considerando itens negativos. Fonte: Adaptado de (ZAKI; JR, 2014).	27
Tabela 3 – Características dos conjuntos de dados do repositório FIMI utilizados. Fonte: Criado pelo autor.	46
Tabela 4 – Características dos conjuntos de dados sintéticos utilizados. Fonte: Cri- ado pelo autor.	47
Tabela 5 – Comparativo de tamanhos de conjuntos de dados.	49
Tabela 6 – Cenários de indicação de uso das implementações de identificação de conjuntos frequentes de acordo com uso de memória.	56
Tabela 7 – Cenários de indicação de uso das implementações de identificação de conjuntos frequentes de acordo com tempo de execução.	61

Lista de abreviaturas e siglas

GRD	Generalized Rule Discovery
NICGAR	Niching Genetic Algorithm
FP-tree	Frequent Pattern Tree
RLE	Run-length encoding
API	Application Programming Interface
CUDA	Compute Unified Device Architecture
OpenMP	Open Multi Processing
GPU	Graphics Processing Unit
AVX	Advanced Vector Extension
CPU	Central Process Unit
FIMI	Frequent Itemset Mining Dataset
UCI	University of California - Irvine
PUMS	Public Use Microdata Sample
HTML	HyperText Markup Language
GCC	GNU Compiler Collection
NCC	Núcleo de Computação Científica

Sumário

1	Introdução	13
1.1	Motivação e justificativa	14
1.2	Objetivos	15
2	Trabalhos relacionados	16
3	Fundamentação teórica	20
3.1	Padrões frequentes	20
3.2	Estruturas de representação bitmap	22
3.3	Mineração de regras de associação	25
3.4	Regras de associação negativas	27
4	Métodos desenvolvidos	29
4.1	Etapa 1: Identificação de conjuntos frequentes	29
4.1.1	Apriori-Bitmap-Cuda	33
4.1.2	Apriori-Simd-Parallel	34
4.1.3	Apriori-Roaring-Parallel	36
4.1.4	Apriori-Cuda-Single-Memcpy	37
4.2	Etapa 2: Mineração de regras negativas	39
4.3	Conjuntos de dados	45
4.3.1	Repositório FIMI de conjuntos de dados	45
4.3.2	Ferramenta IBM Quest para geração de conjuntos de dados	46
5	Experimentos e resultados	48
5.1	Representações de dados	48
5.2	Identificação de conjuntos frequentes	49
5.2.1	Pico de uso de memória	50
5.2.2	Tempo de execução	56
5.2.3	Escalabilidade	61
5.3	Mineração de regras negativas	62
5.3.1	Pico de uso de memória	63
5.3.2	Tempo de execução	66
5.3.3	Regras geradas	68
5.3.4	Escalabilidade	73
6	Conclusões e trabalhos futuros	77
6.1	Contribuições	78

6.2	Trabalhos futuros	78
Referências	80

1 Introdução

Mineração de dados é a denominação dada para processos de descoberta de conhecimento e modelos em conjuntos de dados, e é segmentado em tarefas como análise exploratória, mineração de padrões frequentes, agrupamento e classificação (ZAKI; JR, 2014).

As tarefas de mineração de padrões frequentes englobam as atividades de detectar padrões e associações em conjuntos de dados, sendo esta última denominada mineração de regras de associação. Seu uso é exemplificado pelo caso da análise da cesta de mercado, em que os dados de itens adquiridos são utilizados para determinar padrões de compras de clientes.

Regras de associação expressam relações entre elementos, de forma que a presença de um dado conjunto de elementos X (antecedente) irá implicar na presença de outro conjunto de elementos Y (consequente). Em abordagens tradicionais é somente considerada a informação de ocorrência de elementos, e são portanto denominadas regras de associação positivas.

Entretanto, considerar a ausência de elementos para a geração de regras pode resultar na identificação de associações de grande interesse para certas aplicações (SILVERSTEIN; BRIN; MOTWANI, 1998). Associações deste tipo expressam relações em que a ausência de um dado conjunto de elementos \bar{X} irá implicar na presença de outro conjunto de elementos Y , e demais combinações. Logo, associações deste tipo são denominadas regras de associação negativas.

Para o cálculo deste tipo de regras de associação, diversas métricas podem ser utilizadas, sejam elas exatas (CORNELIS et al., 2006), baseadas em alguma medida estatística (AGGARWAL; YU, 2001; ANTONIE; ZAIANE, 2004; HäMäLÄINEN, 2011) ou utilizando abordagens de inteligência artificial (MI et al., 2018).

Este tipo de mineração, entretanto, demanda uma capacidade computacional consideravelmente alta para seu processamento, visto que o espaço de dados será superior ao caso positivo (MCNICHOLAS; MURPHY; O'REGAN, 2008), e portanto ocorrerá uma explosão na quantidade de regras geradas.

Concomitante com o desenvolvimento de métodos para mineração deste tipo de abordagem, observou-se a evolução das plataformas de computação, que apresentam diversas configurações para processamento paralelo e/ou distribuído. Vários estudos envolvendo estas plataformas aplicados à problemas de mineração de dados podem ser encontrados na literatura (GAN et al., 2017). Dessa forma, entende-se que o uso destas plataformas

pode ser empregado para a mineração do tipo de regras de associação mencionado.

1.1 Motivação e justificativa

A mineração de regras de associação negativas é um processo que, dadas as suas características, apresenta-se como um problema que envolve maior quantidade de dados e etapas que o caso positivo. Esta particularidade é também destacada em diversos estudos (BIAN *et al.*, 2018; BEMARISIKA; TOTOHASINA, 2018; AGRAWAL *et al.*, 2014; MI *et al.*, 2018), que citam que além do problema de demanda de recursos computacionais relacionado à explosão da quantidade de regras geradas, existe um problema de identificar regras que sejam interessantes.

O primeiro estudo que mensura a dimensão deste problema remete-se à (MCNICHOLES; MURPHY; O'REGAN, 2008), onde é apresentado que, para m itens conhecidos, a quantidade máxima de regras negativas geradas é de $5^m - 2 \times 3^m + 1$ regras, enquanto que o caso positivo apresenta um número máximo reduzido de $3^m - 2^{m+1} + 1$ regras.

Em contrapartida, considerar regras negativas no processo de mineração permite que a fronteira de conhecimento obtido do conjunto de dados seja amplamente expandida, possibilitando que um número maior de regras interessantes sejam identificadas. (BIAN *et al.*, 2018) indicam em seu estudo que de 17 regras interessantes geradas considerando mineração de regras negativas, apenas três seriam obtidas analisando somente o caso positivo.

O uso prático de mineração de regras negativas é constatado em diversos estudos de aplicações, como em identificação de padrões de compra de clientes (VERMA; SINGH, 2017), análise de dados criminais (ENGLIN, 2015), cuidados médicos (MAHMOOD; SHAHBAZ; GUERGACHI, 2014) e identificação de erros em código fonte (BIAN *et al.*, 2018). Em alguns destes estudos, os autores afirmam que somente com a mineração de regras negativas foi possível obter o êxito desejado.

Em relação à questão de demanda de recursos computacionais, alguns estudos apresentam implementações que fazem o uso de plataformas heterogêneas (FANG *et al.*, 2009; AROUR; BELKAHLA, 2014; HUANG *et al.*, 2013), memória compartilhada (AGRAWAL; SHAFER, 1996; LIU *et al.*, 2007; SCHLEGEL *et al.*, 2013) e computação distribuída (BAGUI; DHAR, 2018; DJENOURI *et al.*, 2018; VERMA; SINGH, 2017), o que constata que o processo é passível de ser paralelizado.

Entretanto, através de um processo de revisão sistemática da literatura foi possível observar que não existe uma quantidade grande de estudos com foco em paralelização do processo específico de identificação de regras negativas, fato este somado aos demais

citados que motivaram a proposição deste projeto de mestrado.

1.2 Objetivos

O projeto de mestrado descrito na presente dissertação possui como objetivo aplicar abordagens e plataformas paralelas para o desenvolvimento de um método adequado ao caso específico de mineração de regras de associação negativas. Este método desenvolvido tem como objetivos acelerar o processo de geração de regras, e permitir que conjuntos de dados maiores possam ser minerados.

A presente dissertação está estruturada da seguinte maneira. A Seção 2 irá apresentar alguns trabalhos relacionados. A Seção 3 apresenta uma contextualização de mineração de padrões frequentes, representações de dados e regras de associação, com destaque para regras de associação negativas. Na Seção 4 é apresentada a metodologia de pesquisa utilizada. A Seção 5 irá discutir os experimentos e resultados obtidos. A Seção 6 finaliza a dissertação apresentando as conclusões.

2 Trabalhos relacionados

Nesta seção serão descritos alguns trabalhos relacionados com o trabalho desenvolvido. Como o procedimento de mineração de regras de associação apresenta duas principais etapas, são abordados inicialmente estudos que focaram em identificação de padrões frequentes e posteriormente em regras de associação negativas.

Em relação à etapa de identificação de conjuntos frequentes, a maioria dos métodos apresentam um enfoque em tempo de execução. Existem, porém, alguns trabalhos desenvolvidos com o intuito de demandar menos memória durante o processo de mineração. Neste trabalho buscou-se desenvolver o método focando em um baixo consumo de memória, de forma a permitir que conjuntos de dados maiores possam ser minerados.

Um dos primeiros trabalhos com foco em uso de memória foi apresentado por (GOETHALS, 2004), em que realizou-se uma análise entre os métodos Eclat e FP-Growth neste quesito, e foi proposto um novo método baseado no Eclat denominado Medic. Este método apresenta uma melhora na demanda de uso de memória em relação ao seu algoritmo base, entretanto possui como restrição o uso em conjuntos esparsos.

Em (Jian Pei et al., 2001), os autores propõem uma nova estrutura para armazenamento intermediário das transações denominada *H-struct*, baseada na abordagem FP-Growth e que apresenta um mecanismo de reorganização dinâmica durante o processo de mineração. O método trabalha com esta estrutura em memória usando todo o conjunto de dados para análise ou subconjuntos apenas. Devido à essa abordagem, o método demanda pouca memória durante o processo de mineração. Outros métodos baseados no FP-Growth com melhorias no uso de memória incluem (ITKAR; KULKARNI, 2017; YIN et al., 2018; SCHLEGEL; GEMULLA; LEHNER, 2011).

Uma outra abordagem muito comum em métodos de identificação de conjuntos frequentes consiste em reduzir o tamanho da saída, e conseqüentemente reduzir a demanda de memória. Abordagens deste tipo são identificadas em mineração de conjuntos frequentes fechados (PASQUIER et al., 1999) e máximos (BAYARDO, 1998). Nesse caso mantém-se apenas o registro dos maiores conjuntos frequentes, e caso haja necessidade de consultar conjuntos menores, é realizado um processo de recuperação da informação.

Em (MOONESINGHE; FODEH; TAN, 2006) é apresentado um método para mineração de conjuntos frequentes fechados denominado *PGMiner*. Esse método possui como principais características a transformação do conjunto de dados de entrada em uma representação de vetores de bits de comprimentos variáveis, além de estratégias para reduzir o espaço de busca. Por meio destas melhorias e também da característica de minerar conjuntos frequentes fechados, os autores puderam constatar um uso bastante inferior de

memória em comparação aos outros métodos analisados.

Outros estudos propostos para mineração de conjuntos frequentes fechados e máximos são apresentados em (LIU et al., 2003; PAN et al., 2003; PEI; HAN; MAO, 2001; WANG; HAN; PEI, 2003; ZAKI; HSIAO, 2002).

Métodos que empregam representações bitmap podem ser observados nos estudos (ARYABARZAN; MINAEI-BIDGOLI; TESHNEHLAB, 2018) e (CHON; KIM, 2018). Em (ARYABARZAN; MINAEI-BIDGOLI; TESHNEHLAB, 2018), é proposto um método denominado *negFin* que emprega uma estrutura de árvore de prefixos com nós baseados em representações bitmap dos conjuntos. Os autores destacam o ganho no tempo de execução devido ao uso de operações booleanas para o cálculo do valor de suporte, além de outras melhorias e mecanismos de seleção.

No estudo apresentado em (CHON; KIM, 2018), os autores apresentam uma abordagem de emprego de estruturas bitmap, porém totalmente armazenadas em disco, o que reduz consideravelmente o uso de memória. Para evitar um aumento expressivo do tempo de execução, o método analisa blocos de conjuntos bitmap através de operações booleanas, e em discos de tecnologia de estado sólido, o que garante uma latência mais próxima à observada em componentes de memória.

Representações bitmap simples podem, entretanto, apresentar elevado uso de memória para armazenamento das informações, com um pior caso observado para conjuntos de dados esparsos. Para contornar esta deficiência são identificados métodos que utilizam representações bitmap comprimidas.

Compressões para estruturas bitmap são observadas em estudos como WAH (WU et al., 2001), EWAH (LEMIRE; KASER; AOUICHE, 2009), PLWAH (DELIEGE; PEDERSEN, 2010), CONCISE (COLANTONIO; PIETRO, 2010), VALWAH (GUZUN et al., 2014), SBH (KIM et al., 2016), Roaring (LEMIRE; KAI; KASER, 2016) e BitMagic (PIETERSE et al., 2010).

Em (FADISHEI; DOUSTIAN; SAADATI, 2019), os autores realizam uma análise de quatro tipos de compressão (EWAH, CONCISE, Roaring e BitMagic) empregadas em implementações customizadas do método Eclat, avaliando tempo de execução, uso de memória e consumo de energia.

A análise realizada por (WANG et al., 2017) também buscou comparar os diversos tipos de compressão bitmap, e também estruturas de listas invertidas em questões de armazenamento, recuperação de informações e operações de consultas. A compressão Roaring (LEMIRE; KAI; KASER, 2016) é a que possui melhor desempenho e portanto é recomendada para implementações de algoritmos que empregam representações comprimidas de bitmaps.

Em (SAEED et al., 2016), os autores apresentam um método para identificação de

conjuntos frequentes utilizando compressão Roaring e o paradigma MapReduce. Esse método minera conjuntos de dados com valores muito baixos de suporte mínimo, e portanto um foco em tempo de execução.

Em relação à etapa de geração de regras de associação negativas, a solução deste problema envolve um conjunto de regras muito superior ao caso apenas positivo. Devido à isto, a maioria dos métodos busca obter soluções que utilizem medidas estatísticas, mecanismos de poda ou abordagens de inteligência artificial para permitir que esta solução seja encontrada com os recursos computacionais disponíveis.

O primeiro estudo a mencionar regras de associação negativas é o descrito em (SILVERSTEIN; BRIN; MOTWANI, 1998). Neste, os autores determinam a relação entre dois conjuntos de itens com o uso da medida estatística do teste chi-quadrado, que avalia o grau de independência entre conjuntos. Uma limitação deste método é seu uso em conjuntos de dados grandes e densos.

No estudo (ANTONIE; ZAIĀNE, 2004), os autores geram regras de associação negativas através do emprego do coeficiente de correlação de Person. Além do emprego da medida estatística, este método gera apenas regras em que ou o conseqüente ou o antecedente possuem conjuntos negativos, e portanto não considera regras em que ambos são negativos, o que caracteriza um mecanismo de poda.

Uma outra abordagem utilizando medidas estatísticas é o método Kingfisher (HÄMÄLÄINEN, 2011), que identifica regras de dependência utilizando o teste exato de Fisher e as utiliza como base para geração de regras de associação. Dois mecanismos de poda são empregados neste estudo, sendo o primeiro o fato de que o método apenas identifica regras com conjuntos conseqüentes contendo apenas um item e o segundo que a autora introduz um mecanismo de verificação para evitar gerar regras redundantes.

Outros estudos para identificação de regras negativas que utilizam medidas estatísticas são apresentados em (AGGARWAL; YU, 1998) e (KOH; PEARS, 2007).

Em relação à métodos exatos que empregam mecanismos de poda podem ser citados os estudos apresentados em (THIRUVADY; WEBB, 2004), (CORNELIS et al., 2006), (BEMARISIKA; TOTOHASINA, 2018) e (BIAN et al., 2018). O método GRD (*Generalized Rule Discovery*) (THIRUVADY; WEBB, 2004) minera regras negativas utilizando as medidas exatas de suporte mínimo e alavancagem. Como forma de reduzir a demanda de recursos computacionais ele apenas minera um número reduzido de regras definido pelo usuário e apresenta a limitação de minerar regras contendo no máximo cinco itens, somados antecedente e conseqüente. Este método, apesar de ser exato apresenta condições de poda.

O algoritmo PNAR (CORNELIS et al., 2006) minera regras de maneira exata e empregando as medidas de suporte e confiança. Este método utiliza uma definição

de que regras negativas são geradas através de regras positivas. Esta definição permite que a busca de regras seja feita em quatro etapas e de maneira incremental, explorando uma propriedade de restrição de regras. Este método foi selecionado como base para a implementação deste trabalho de mestrado e será descrito com mais detalhes na seção de metodologia.

O método ERAPN (BEMARISIKA; TOTOHASINA, 2018) gera regras de associação negativas com o emprego das medidas suporte, confiança, e também de uma medida adicional denominada MGK para seleção de regras interessantes. O emprego desta medida adicional caracteriza um mecanismo de poda, reduzindo a quantidade de regras geradas. O método também trabalha com o conceito de geradores, que permite reduzir diversas etapas de cálculo das métricas de interesse, e portanto a quantidade de leituras do conjunto de dados de entrada.

No estudo (BIAN et al., 2018), os autores adaptaram um procedimento de geração de regras negativas baseado no método Apriori para um problema específico de identificar erros em código fonte. Nesta adaptação, não é utilizada a métrica confiança, mas sim uma medida de entropia definida pelos autores. No estudo é destacado o problema de explosão de regras geradas, sendo que várias destas apresentam pouco interesse prático. A medida adotada de entropia é descrita pelos autores como um mecanismo de poda que foi construído com base no tipo de problema sendo abordado, e que para outros problemas, medidas semelhantes devem ser elaboradas.

Abordagens utilizando conceitos de inteligência artificial são apresentadas em (MI et al., 2018) e (MARTÍN et al., 2016). Em (MI et al., 2018), os autores apresentam um método de mineração de regras de associação negativas na forma de um problema de otimização multi-objetivo, empregando a teoria de algoritmos evolutivos para execução. Para isto, é definida uma medida de certeza da regra, a qual é atualizada a cada iteração do algoritmo até que satisfaça um limiar informado. Como resultado dos experimentos, os autores expressam que esta abordagem possui tempo de execução e medida de interesse das informações mineradas melhores em relação às demais avaliadas.

O trabalho de (MARTÍN et al., 2016) apresenta um método denominado NICGAR (*Niching Genetic Algorithm*) que emprega os conceito de algoritmos genéticos para mineração de regras de associação negativas. No estudo, os autores realizam diversas comparações com outros algoritmos determinísticos e genéticos, e expressam que a grande vantagem do algoritmo proposto é que ele agrupa as regras de associação em nichos, evitando que sejam geradas muitas regras similares.

3 Fundamentação teórica

Nesta seção é apresentada uma contextualização dos fundamentos envolvidos no desenvolvimento do projeto de mestrado. Inicialmente serão apresentados o conceito de padrões frequentes e os modelos de estruturação de dados necessários para executar esta tarefa, que contemplam representações de bancos de dados por transações e bitmap, incluindo modelos de bitmap comprimidos.

Na sequência, é descrito o processo de mineração de regras de associação e suas métricas, as quais são utilizadas para determinar o quão forte são as regras identificadas.

Por fim, é apresentado o conceito de regras de associação negativas, que é o foco principal deste trabalho.

3.1 Padrões frequentes

O objeto de estudo de mineração de padrões frequentes será um banco de dados \mathbf{D} estruturado no formato de transações, como por exemplo produtos que compõem uma compra, palavras presentes em um parágrafo de um texto ou posições intermediárias durante a movimentação de um objeto.

Cada elemento das transações é um item, o qual deverá pertencer à um conjunto $\mathcal{I} = \{x_1, x_2, \dots, x_m\}$ que englobe todos os m itens conhecidos. Definimos também que um ou mais itens irão representar um conjunto $X_k \subseteq \mathcal{I}$, em que o índice k representa a cardinalidade ou tamanho do conjunto.

Cada transação em \mathbf{D} deve possuir um identificador, que irá pertencer à um conjunto $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ e será denominado *tid*. Um conjunto $T \subseteq \mathcal{T}$ será denominado um conjunto de tids.

A representação de uma transação contida em \mathbf{D} é então uma tupla do formato $\langle t, \mathbf{i}(t) \rangle$, onde $t \in \mathcal{T}$ é um identificador único de transação, e $\mathbf{i}(t)$ é um conjunto de itens relativo àquela transação. A Tabela 1 apresenta um exemplo desta estrutura de banco de dados.

Para determinar os conjuntos frequentes, é necessário definir a métrica *suporte*. Dado um banco de dados de transações \mathbf{D} , definimos $sup(X, \mathbf{D})$ como a métrica *suporte* do conjunto de itens X em relação à \mathbf{D} , que é o número de transações em \mathbf{D} que contém X :

$$sup(X, \mathbf{D}) = |\{t | \langle t, \mathbf{i}(t) \rangle \in \mathbf{D} \text{ e } X \subseteq \mathbf{i}(t)\}| = |\mathbf{t}(X)| \quad (3.1)$$

Logo, um conjunto de itens X é dito frequente em \mathbf{D} se $sup(X, \mathbf{D}) \geq minsup$,

Tabela 1 – Estrutura de banco de dados de transação. Fonte: (ZAKI; JR, 2014).

t	$\mathbf{i}(t)$
1	$ABDE$
2	BCE
3	$ABDE$
4	$ABCE$
5	$ABCDE$
6	BCD

onde $minsup$ é um limiar de suporte mínimo definido pelo usuário.

Como a definição de suporte neste caso está relacionada ao tamanho total do banco de dados, define-se a métrica de *suporte relativo*, que é a fração das transações que contém o conjunto de itens X :

$$rsup(X, \mathbf{D}) = \frac{sup(X, \mathbf{D})}{|\mathbf{D}|} \quad (3.2)$$

Desta forma, o usuário poderá definir o valor de suporte mínimo sem que seja necessário conhecer o tamanho total do banco. Conjuntos que sejam frequentes, serão parte de \mathcal{F} , que é a representação dos conjuntos frequentes encontrados.

Por exemplo, considerando o banco de dados da Tabela 1, e um valor de $minsup = 3$ ($rminsup = 0,5$), tem-se o seguinte conjunto de itens frequentes \mathcal{F} , apresentado com discriminação por cardinalidade k :

$$\mathcal{F}_1 = \{A, B, C, D, E\} \quad (3.3)$$

$$\mathcal{F}_2 = \{AB, AD, AE, BC, BD, BE, CE, DE\} \quad (3.4)$$

$$\mathcal{F}_3 = \{ABD, ABE, ADE, BCE, BDE\} \quad (3.5)$$

$$\mathcal{F}_4 = \{ABDE\} \quad (3.6)$$

Existem várias abordagens para mineração de padrões frequentes difundidas na literatura, dentre as quais destacam-se os algoritmos Apriori (AGRAWAL; SRIKANT, 1994), Eclat (ZAKI, 2000) e FPGrowth (HAN; PEI; YIN, 2000).

O algoritmo Apriori realiza uma exploração em largura no espaço de busca de conjuntos candidatos, e foi um dos primeiros a empregar uma estratégia de poda baseada em duas propriedades. Sejam $X, Y \subseteq \mathcal{I}$ dois conjuntos de itens quaisquer, se $X \subseteq Y$, então $sup(X) \geq sup(Y)$, o que leva às duas importantes propriedades citadas: (1) caso X seja frequente, então qualquer subconjunto $Y \subseteq X$ também será frequente, e (2) caso X não seja frequente, então qualquer superconjunto $X \supseteq Y$ também não será frequente.

Na Figura 1 é possível observar o efeito dessas duas propriedades de poda ao analisar o conjunto de dados exemplo. Nós escurecidos indicam conjuntos identificados como não frequente, e qualquer outro conjunto de cardinalidade superior e que seja supercon-

juntos destes devem ser removidos do procedimento de mineração (nós e vértices que estão pontilhados).

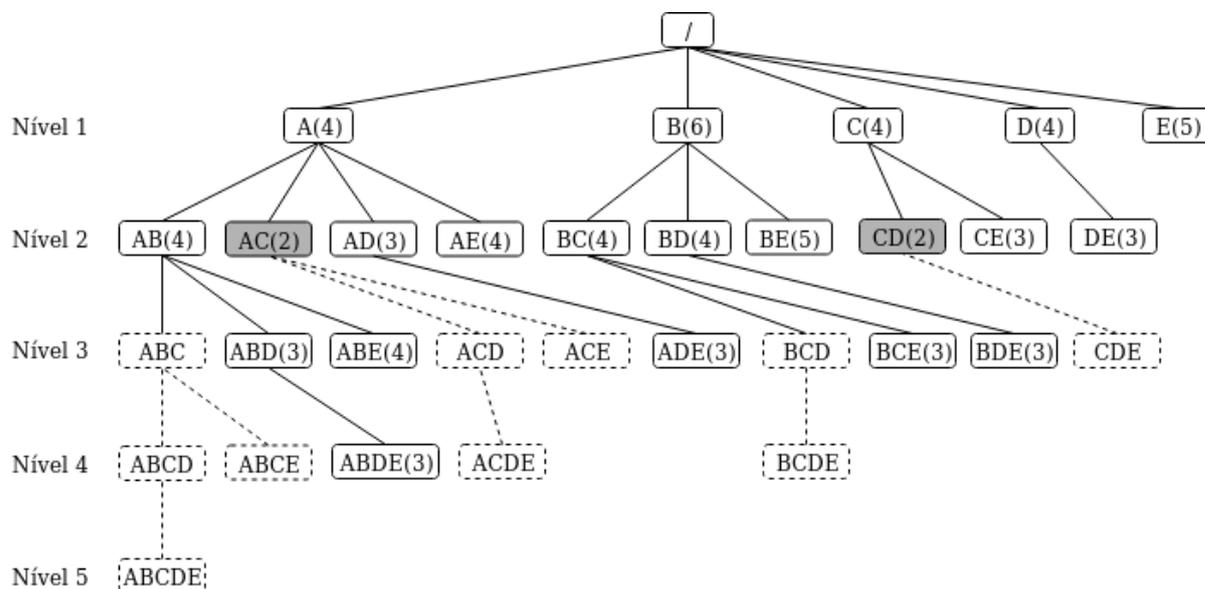


Figura 1 – Exemplo da operação do método Apriori. Nós mais escurecidos indicam candidatos identificados como não frequentes, e nós e vértices pontilhados indicam elementos eliminados pelo mecanismo de poda. Fonte: Adaptado de (ZAKI; JR, 2014).

O algoritmo Eclat apresenta uma abordagem que objetiva aprimorar o cálculo de suporte, alterando a representação do banco para utilizar a informação de índices das transações (conjunto de *tids*) durante este processamento. A ideia básica é que o suporte de um conjunto candidato pode ser calculado pela interseção de dois subconjuntos.

Já o algoritmo FPGrowth possui como estratégia principal representar **D** em uma estrutura compacta denominada FP-tree (*frequent pattern tree*), que será ordenada de maneira decrescente de acordo com o valor do suporte. A Figura 2 ilustra esta estrutura FP-tree para o banco de dados exemplo.

Estes três principais métodos são base para diversas outras abordagens, que apresentam alterações em etapas do processo de mineração.

3.2 Estruturas de representação bitmap

Bitmap é uma estrutura de representação de dados muito utilizada em sistemas gerenciadores de banco de dados, e pode ser visualizada como uma tabela, em que as colunas estão bem definidas, e os valores são do tipo verdadeiro ou falso.

A Figura 3 demonstra a correspondência entre um banco de dados de transações e este tipo de representação. Cada valor é representado por um bit, que será 1 caso o conjunto esteja presente em determinada transação, e 0 caso esteja ausente.

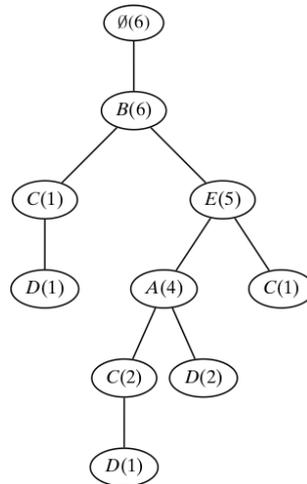


Figura 2 – Exemplo de estrutura FP-tree para o banco de dados exemplo. Fonte: Adaptado de (ZAKI; JR, 2014).

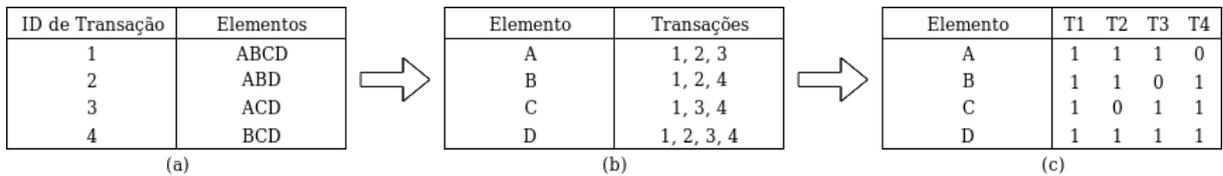


Figura 3 – Exemplo de estrutura bitmap de representação de dados. Fonte: Adaptado de (FANG et al., 2009).

Este tipo de representação é interessante no contexto mencionado, pois apresenta algumas vantagens substanciais. Uma primeira a ser citada é que requisições que se baseiam em operações sobre conjuntos, como união, interseção e diferença, são realizadas de maneira muito eficiente, dado que são executadas por meio de operadores booleanos (**AND**, **OR**) sobre as diversas linhas do bitmap. A Figura 4 apresenta um exemplo de cálculo de linhas bitmap de conjuntos candidatos utilizando operações booleanas.

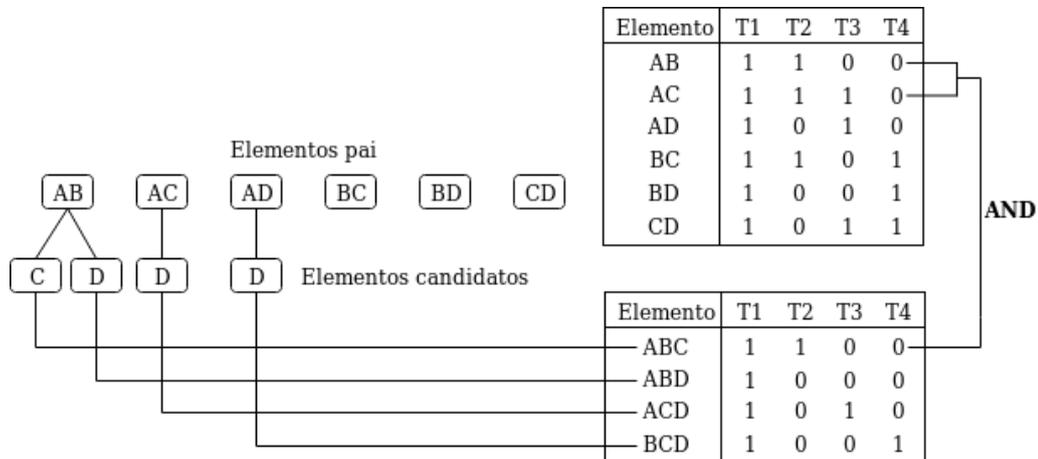


Figura 4 – Exemplo de cálculo de linhas bitmap relativas à candidatos com cardinalidade superior por meio de operações booleanas. Fonte: Adaptado de (FANG et al., 2009).

Uma segunda vantagem é que, devido à característica de a representação possuir as colunas bem definidas, esta estrutura é apta a ser executada por meio de instruções vetoriais (POHL et al., 2016), o que permite explorar diversas plataformas de computação paralela (FANG et al., 2009).

Além disso, para bancos de dados densos, existe uma redução da quantidade de *bytes* necessários para o armazenamento da informação, nos quais ao invés de representar a presença de um conjunto em uma transação por uma palavra do processador, é utilizado um bit.

Por outro lado, para os bancos de dados esparsos, ou seja, que apresentam uma grande diversidade de itens para a quantidade de transações presente, o tamanho da representação em bitmap poderá ser superior à original.

Neste tipo de representação de dados, o cálculo do valor da métrica suporte se reduz à uma simples soma de bits 1s, que em processadores atuais é realizada de forma eficiente através de instruções do tipo *popcount* (MULA; KURZ; LEMIRE, 2016).

Na prática, para fazer uso das operações eficientes da representação bitmap e evitar desperdício de espaço para o armazenamento dos dados, são utilizadas representações comprimidas, das quais destacam-se WAH (WU et al., 2001), EWAH (LEMIRE; KASER; AOUICHE, 2009), PLWAH (DELIEGE; PEDERSEN, 2010), CONCISE (COLANTONIO; PIETRO, 2010), VALWAH (GUZUN et al., 2014), SBH (KIM et al., 2016), Roaring (LEMIRE; KAI; KASER, 2016) e BitMagic (PIETERSE et al., 2010).

Com exceção da compressão Roaring, os demais tipos empregam o mecanismo *run-length encoding (RLE)*, que pode ser aplicado à *bytes* ou palavras.

Run-length encoding (RLE) é uma técnica de compressão que, de maneira geral, consiste em separar o bitmap não comprimido em vários grupos, que podem ter duas classificações. A primeira ocorre quando o grupo contém apenas bits com o mesmo valor (somente 1s ou 0s), e é denominado grupo de preenchimento. Já na segunda classificação, o grupo apresenta tanto bits 1s quanto bits 0s, e é denominado grupo literal. Finalizada a partição, a compressão é então aplicada aos grupos, de forma que estes sejam representados por uma sequência menor de bits, porém que apresentem significados para sua interpretação.

As demais compressões citadas, com exceção da Roaring que não utiliza o conceito de *Run-length encoding (RLE)*, são baseadas no método WAH, e apresentam alguma melhoria, seja em tempo de execução das operações ou em eficiência da compressão.

A compressão Roaring, por sua vez, é um método híbrido que particiona os dados em trechos de 2^{16} inteiros em que os elementos presentes em trechos idênticos compartilham os mesmos 16 bits mais significativos, e podem ser representados por um registro não comprimido de 65536 bits ou uma lista ordenada de inteiros de 16 bits.

A escolha da representação do trecho é em razão da quantidade dos k elementos presentes neste. Caso $k > 4096$, será utilizado o bitmap não comprimido, e caso contrário, a lista ordenada de inteiros. A escolha deste valor de limiar garante que cada inteiro não utilize mais que 16 bits para representação do trecho, dado que o Roaring utiliza 65536 bits para representar 4096 inteiros ou 16 bits no máximo por inteiro na lista ordenada (WANG et al., 2017).

Outra característica importante do Roaring, é que, diferentemente das demais compressões, permite que operações de interseção (**AND**) e união (**OR**) sejam realizadas diretamente sobre o bitmap comprimido, o que as torna mais eficientes. A busca de um determinado elemento no bitmap também é eficiente, pois a maneira de armazenamento permite que seja aplicada uma busca binária.

Em relação à operação de descompressão, segundo a análise de (WANG et al., 2017), dentre todos os métodos de compressão, o Roaring é o que apresenta o melhor desempenho neste quesito.

3.3 Mineração de regras de associação

Dados dois conjuntos de elementos X e Y , tal que $X, Y \subseteq \mathcal{I}$ e $X \cap Y = \emptyset$, uma regra de associação é uma expressão da forma $X \rightarrow Y$. O suporte da regra será a quantidade de transações que contém o conjunto $X \cup Y$ (ou XY):

$$\text{sup}(X \rightarrow Y) = |\mathbf{t}(XY)| = \text{sup}(XY) \quad (3.7)$$

Semelhante ao que foi apresentado anteriormente, o suporte relativo da regra será o valor do suporte dividido pela quantidade total de transações:

$$\text{rsup}(X \rightarrow Y) = \frac{\text{sup}(XY)}{|\mathbf{D}|} = P(X \wedge Y) \quad (3.8)$$

Define-se então a medida de *confiança*, que é a probabilidade condicional de que uma transação contenha Y , dado que ela contém X :

$$\text{conf}(X \rightarrow Y) = P(Y|X) = \frac{P(X \wedge Y)}{P(X)} = \frac{\text{sup}(XY)}{\text{sup}(X)} \quad (3.9)$$

A *confiança* é uma medida de acurácia da regra gerada, que define a proporção das transações para as quais a regra é satisfeita (BRAMER, 2013). Idealmente, espera-se que toda transação que contenha Y , também contenha X , o que definiria a regra como exata, com valor de confiança igual a 1. Porém, em casos práticos as regras não são exatas, e possuirão um valor de confiança menor que 1.

Similar ao que foi estipulado para *suporte*, define-se um valor limite de *confiança* (*minconf*), e toda regra que apresentar valor superior à este será classificada como regra forte.

Após esta definição, para cada conjunto frequente serão geradas regras candidatas através de um procedimento incremental, no qual serão realizadas combinações de subconjuntos no conjunto conseqüente. Caso a regra não seja forte, os candidatos que seriam derivados desta não precisam ser avaliados, o que caracteriza um mecanismo de poda. A Figura 5 ilustra esse mecanismo de geração de regras.

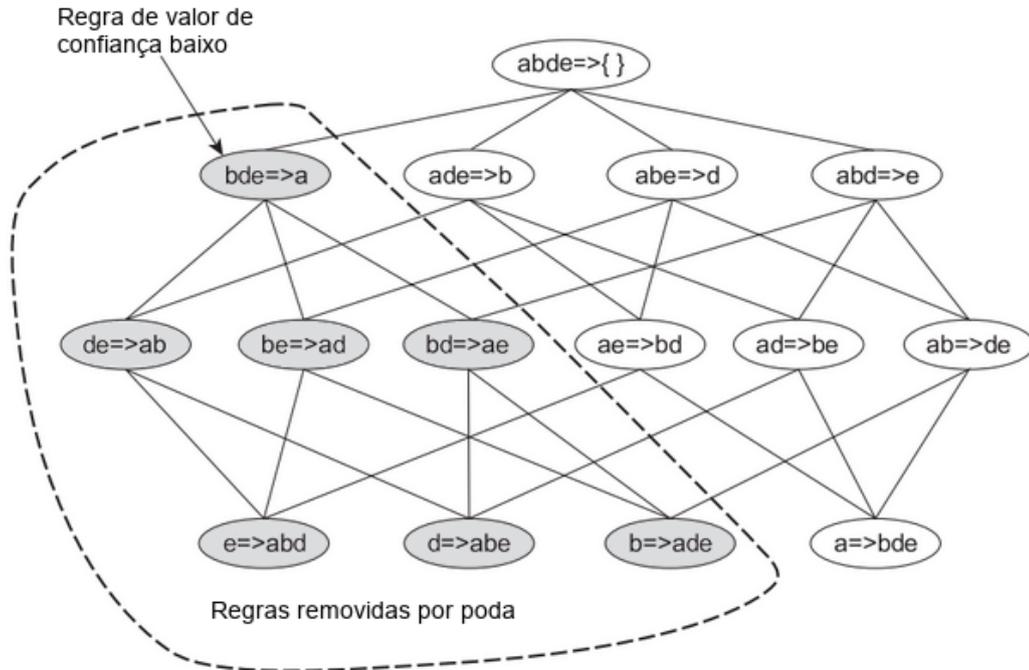


Figura 5 – Exemplo de geração de regras de associação. Fonte: Adaptado de (FANG et al., 2009).

Duas medidas adicionais de interesse de regras de associação podem ser utilizadas, que são *elevação* (*lift*) (BRIN et al., 1997) e *alavancagem* (*leverage*) (PIATETSKY-SHAPIRO, 1991).

A *elevação* (*lift*) da regra $X \rightarrow Y$ mede quantas vezes a mais os itens em X e Y ocorrem juntos em transações das quais seria esperado que estes conjuntos fossem estatisticamente independentes.

$$lift(X \rightarrow Y) = \frac{sup(X \cup Y)}{sup(X) \times sup(Y)} = \frac{conf(X \rightarrow Y)}{sup(Y)} \quad (3.10)$$

Valores de *elevação* superiores à 1 são considerados interessantes, pois indicam que transações contendo X tendem a conter Y com frequência superior à transações que contenham somente X .

Já *alavancagem* é uma medida da diferença entre os valores de suporte de $X \cup Y$ e do suporte que seria esperado caso X e Y fossem independentes.

$$leverage(X \rightarrow Y) = sup(X \cup Y) - sup(X) \times sup(Y) \quad (3.11)$$

3.4 Regras de associação negativas

Define-se um item negativo com a denominação \bar{x}_n , que representa que o item $x_n \in \mathcal{I}$ é ausente em uma determinada transação $\mathbf{i}(t)$. O valor de suporte de \bar{x}_n pode ser obtido pelo seu complemento, $sup(\bar{x}_n) = 1 - sup(x_n)$. Esta propriedade de suporte é decorrente da representação aumentada do banco de dados quando consideram-se os itens negativos, conforme é exemplificado na Tabela 2.

Tabela 2 – Representação de banco de dados considerando itens negativos. Fonte: Adaptado de (ZAKI; JR, 2014).

t	$\mathbf{i}(t)$ original	$\mathbf{i}(t)$ aumentado
1	$ABDE$	$ABCDE$
2	BCE	\overline{ABCDE}
3	$ABDE$	$AB\overline{CDE}$
4	$ABCE$	$ABC\overline{DE}$
5	$ABCDE$	$ABCDE$
6	BCD	\overline{ABCDE}

Uma regra de associação negativa será definida então como uma relação $X \rightarrow Y$, em que $X \subseteq \mathcal{I}, Y \subseteq \mathcal{I}$, e $X \cap Y = \emptyset$, e que X e/ou Y contenham pelo menos um item negativo.

A principal decorrência de se considerar itens negativos é a explosão de regras geradas, que segundo (MCNICHOLAS; MURPHY; O'REGAN, 2008), o número total seria de $5^m - 2 \times 3^m + 1$ regras. Considerando somente o caso positivo, este número se reduz à $3^m - 2^{m+1} + 1$, onde m é a quantidade de itens conhecidos.

Para exemplificar, considerando o banco de dados apresentado na Tabela 2, é possível encontrar até 180 regras positivas, e até 2640 regras quando consideram-se os itens negativos. Esta explosão no espaço de busca demanda uma capacidade de processamento e memória consideravelmente superior ao caso de regras positivas.

Além disso, em questão qualitativa, devido à geração de uma quantidade superior de regras, é necessário que as métricas sejam adaptadas para que ocorra a discriminação de regras que são interessantes, evitando gerar associações triviais e/ou de pouca utilidade. Em (BIAN et al., 2018) os autores reforçam este ponto, comentando que foi necessário adicionar uma restrição específica do problema em estudo para reduzir a quantidade de regras geradas.

O cálculo deste tipo de regras de associação pode ser realizado de forma exata, fazendo o uso de alguma medida estatística ou através de abordagens de inteligência artificial.

Em relação ao cálculo exato, em (CORNELIS et al., 2006) os autores apresentam duas possíveis maneiras de se interpretar regras negativas. A primeira abordagem define

que, dada uma regra positiva $X \rightarrow Y$, existem três possíveis casos de regras negativas a serem analisados: $X \rightarrow \bar{Y}$, $\bar{X} \rightarrow Y$ e $\bar{X} \rightarrow \bar{Y}$. Nesta definição os valores de suporte são calculados de maneira similar ao caso somente positivo, discriminando esta etapa e também o processo de geração de regras em quatro partes. No estudo os autores deduzem diversas identidades matemáticas do caso positivo para cada um dos três casos negativos, e indicam que o número máximo de regras negativas geradas nesta definição é dado pela equação $(4 \times (3^m - 2^{m+1} + 1))$.

Já uma segunda abordagem define que o espaço de itens conhecidos no conjunto de dados seja ampliado com cada um dos seus complementos $I' = I \cup \bar{i}_1, \bar{i}_2, \dots, \bar{i}_m$ (semelhante ao exibido na Tabela 2). Dessa maneira, os autores argumentam que o entendimento e a implementação serão mais simples, pois os itens negativos são tratados como positivos durante o cálculo de suporte. Além disso, mantém-se a propriedade de poda apresentada no método Apriori, na qual conjuntos com cardinalidade superior somente serão frequentes se seus subconjuntos forem frequentes. Um ponto restritivo se refere à quantidade máxima possível de regras geradas, que é muito superior à primeira definição e é dada pela equação $(5^m - 2 \times (3^m) + 1)$. É interessante ressaltar que os próprios autores comentam que esta segunda definição possui aplicações práticas apenas em conjuntos de dados menores.

Considerando a geração de regras negativas através de medidas estatísticas, várias abordagens são identificadas na literatura, como por exemplo utilizando medidas de correlação de regras (AGGARWAL; YU, 2001), coeficiente de correlação de Person (ANTONIE; ZAIANE, 2004), teste chi-quadrado (SILVERSTEIN; BRIN; MOTWANI, 1998) e teste exato de Fisher (HÄMÄLÄINEN, 2011). Métodos utilizando conceitos de inteligência artificial incluem problemas de otimização multi-objetivo (MI et al., 2018) e algoritmos genéticos (MARTÍN et al., 2016).

4 Métodos desenvolvidos

Esta seção descreve a metodologia utilizada no desenvolvimento do trabalho. Entende-se que a mineração de regras negativas consiste de duas principais etapas, que são identificação de conjuntos frequentes e geração de regras de associação.

Para a identificação de conjuntos frequentes são apresentadas as quatro abordagens desenvolvidas: Apriori-Bitmap-Cuda, Apriori-Simd-Parallel, Apriori-Roaring-Parallel e Apriori-Cuda-Single-Memcpy. Para a geração de regras é apresentado o método desenvolvido.

Serão descritos também os conjuntos de dados utilizados para avaliação das implementações realizadas.

4.1 Etapa 1: Identificação de conjuntos frequentes

Para o desenvolvimento da primeira etapa do trabalho, que é identificação de conjuntos frequentes, buscou-se elaborar um método que faça uso de plataformas paralelas de computação. Foi escolhido o modelo de arquitetura de computação heterogênea, que envolve o desenvolvimento de uma implementação que possa utilizar vetorização, paralelização em memória compartilhada e paralelização em aceleradores.

No desenvolvimento do método foram selecionados os algoritmos Apriori e Eclat como base, que embora apresentem desempenho sequencial inferior ao FP-Growth, devido à sua estrutura são mais adequados à modificações para paralelização.

Estes métodos, de maneira genérica, apresentam os seguintes passos de execução:

1. Leitura do conjunto de dados de entrada para identificação de itens únicos frequentes
2. Geração de conjuntos candidatos com cardinalidade imediatamente superior
3. Cálculo do valor de suporte dos candidatos e eliminação de candidatos não frequentes
4. Repetição do processo de geração de candidatos e cálculo de suporte até que não existam mais candidatos

Conjuntos de dados de transações apresentam uma estrutura irregular, com tamanhos variados de linhas, o que é inadequado para um processamento paralelo, acarretando em divergência entre os elementos de processamento. A estrutura de bitmap, por outro lado, apresenta tamanhos fixos de linhas.

Dessa forma, no método proposto, o primeiro passo comentado anteriormente será adicionado do processo de transformação do conjunto de dados de representação de transações para bitmap (ilustrado pela Figura 3). Este primeiro passo consiste de um processo altamente dependente de operações de leitura em disco, de forma que possíveis paralelizações serão realizadas no âmbito de memória compartilhada.

Para armazenamento de conjuntos frequentes e geração de candidatos, é utilizada a estrutura de dados de árvore de prefixos, conforme apresentada na Figura 6.

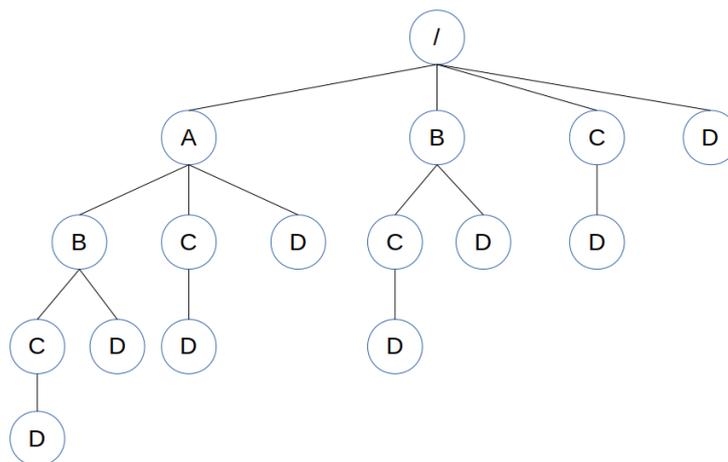


Figura 6 – Exemplo de árvore de prefixos. Fonte: Criado pelo autor.

É importante observar que a geração de conjuntos candidatos será sempre baseada em elementos vizinhos à direita do elemento atual. Dessa forma, é possível manipular o balanceamento da árvore através de algumas modificações no método, como por exemplo ordenar os elementos frequentes iniciais em valor de suporte crescente, o que irá impedir o lado esquerdo da árvore de se aprofundar. Caso a ordem seja decrescente na mesma métrica, a árvore será na maioria dos casos desbalanceada à esquerda. Candidatos que não forem frequentes serão removidos da árvore para reduzir o uso de memória.

Em questão da maneira como a árvore será percorrida, a abordagem Apriori realiza a geração de candidatos em largura, o que caracteriza a melhor condição para paralelização mas também a que mais demanda uso de memória (representada pela imagem (a) da Figura 7). O método Eclat, por sua vez, gera candidatos em profundidade, apresentando o menor uso de memória, porém o pior cenário para paralelização (representado pela imagem (b) da Figura 7).

O método proposto adotará uma abordagem balanceada, onde será avançado um nível em profundidade e gerados os candidatos daquele nó (imagem (c) da Figura 7). Desta forma, para problemas grandes, o método estará apto à paralelização com um uso menor de memória.

Este passo de geração de candidatos apresenta uma estrutura complexa de representação (árvore de prefixos), de maneira que será desenvolvido no âmbito de programação

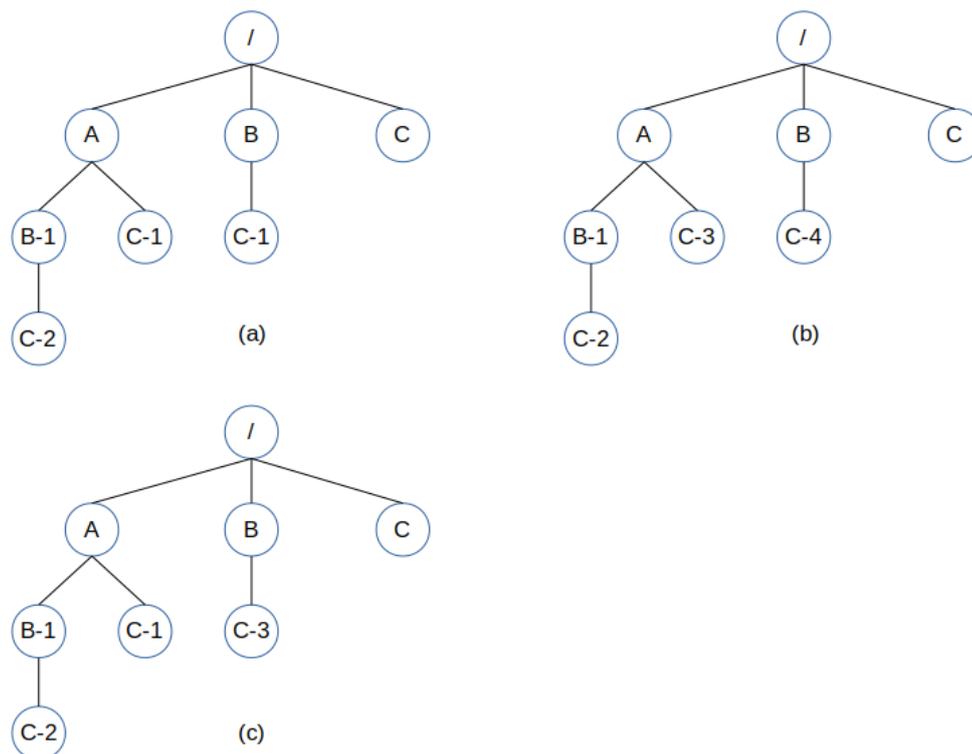


Figura 7 – Geração de candidatos em árvore de prefixos (números nos nós indicam a iteração em que o candidato é gerado). Método Apriori realiza a geração em largura (a), demandando um uso maior de memória porém estando mais apto à paralelização. Método Eclat gera candidatos em profundidade (b), estando menos apto à paralelização porém apresentando a menor demanda de uso de memória. Método proposto (c) irá atuar de uma forma balanceada, avançando de forma conjunta em largura e profundidade. Fonte: Criado pelo autor.

paralela em memória compartilhada.

O desenvolvimento do terceiro passo será o que irá apresentar o maior grau de paralelização devido à estrutura regular do bitmap. Neste serão calculadas as novas linhas da representação bitmap relativas aos conjuntos candidatos e os respectivos valores de suporte.

Conforme apresentado na Figura 4, o cálculo das linhas do bitmap relativas aos conjuntos candidatos será feito por meio de operações booleanas, que são computacionalmente executadas de maneira eficiente. Já o cálculo do valor de suporte envolve realizar a soma dos bits 1s presentes na linha do bitmap, o que também é executado de maneira eficiente por meio de instruções do tipo *popcount*. Ambas as atividades são passíveis de paralelização e vetorização.

Este cálculo de soma dos bits 1s pode adotar duas estratégias de uso de *threads*. Uma primeira consiste em atribuir à cada *thread* a tarefa de analisar, para um mesmo conjunto candidato, um trecho da respectiva linha bitmap, conforme representado na Figura 8. O tamanho deste trecho será definido pela quantidade de *threads* disponíveis e comprimento da linha bitmap, e entende-se que esta estratégia é adequada para análise de conjuntos bitmaps que apresentem linhas muito extensas (conjuntos com muitas

transações).

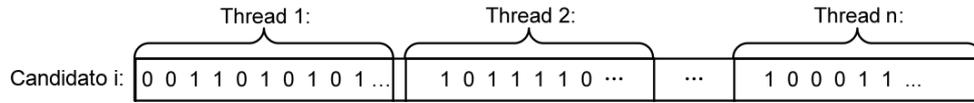


Figura 8 – Estratêgia de cálculo da soma de bits 1s em que cada *thread* faz a análise de um trecho da linha bitmap. Esta estratêgia é mais adequada para o caso em que as linhas da representação bitmap sejam muito extensas. Fonte: Criado pelo autor.

Uma segunda estratêgia consiste em atribuir à cada *thread* a tarefa de analisar a linha bitmap de um conjunto candidato, conforme representado na Figura 9. Entende-se que esta estratêgia é adequada para análise de conjuntos de dados densos, que permitem a geração de muitos candidatos durante o processo de mineração.

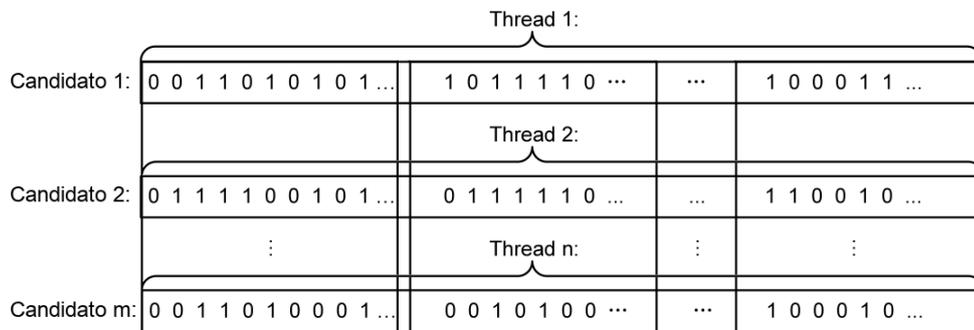


Figura 9 – Estratêgia de cálculo da soma de bits 1s em que cada *thread* faz a análise completa de uma linha bitmap de um conjunto candidato. Esta estratêgia é mais adequada para o caso em que existam muitos candidatos. Fonte: Criado pelo autor.

Para comparação de desempenho do método proposto, serão utilizadas as implementações de Christian Borgelt dos algoritmos Apriori (BORGELT; KRUSE, 2002), Eclat (BORGELT, 2003) e FP-Growth (BORGELT, 2005). Estas são implementações sequenciais dos algoritmos base de identificação de padrões frequentes e que apresentam código fonte disponível publicamente. Entende-se que estas são as melhores implementações disponíveis, contemplando diversas otimizações aplicadas por Borgelt, documentação detalhada e atualizações constantes.

Em relação à implementações paralelas, muito pouco conteúdo foi identificado, sendo que estes apresentam uma documentação deficitária, o que não as caracteriza como padrões para comparação.

Em relação às ferramentas utilizadas para desenvolvimento de código, foram empregadas linguagem de programação C/C++, API de computação heterogênea CUDA e API OpenMP de computação paralela em memória compartilhada.

Feitas estas considerações, foi desenvolvido um método de identificação de conjuntos frequentes que contempla quatro implementações que exploram diferentes características do *hardware*:

- Apriori-Bitmap-Cuda
- Apriori-Simd-Parallel
- Apriori-Roaring-Parallel
- Apriori-Cuda-Single-Memcpy

O desenvolvimento das quatro implementações teve como motivação a ideia de que cada uma destas possui um desempenho melhor em determinado cenário, os quais serão listados em uma tabela na apresentação dos resultados. Nas subseções são descritas estas implementações.

4.1.1 Apriori-Bitmap-Cuda

Para essa implementação, a principal característica é a execução do cálculo de suporte (terceiro passo) em GPU (Figura 10) de maneira dinâmica. No Apriori-Bitmap-Cuda, a navegação pela árvore de prefixos é feita de forma balanceada e dividida em subiterações, conforme representado pelos nós destacados na Figura 10. A quantidade máxima de candidatos para cada uma dessas subiterações depende dos recursos computacionais em uso, de forma que as estruturas bitmap possam estar residentes na memória da GPU.

Nessa implementação foram levantados os tamanhos dos bitmaps representando os conjuntos de dados original (conjuntos frequentes únicos) e intermediários (conjuntos candidatos com dois ou mais elementos), para que possa ser determinada a quantidade ϵ máxima de candidatos que podem ser calculados pela GPU em cada subiteração.

Considerando as estruturas bitmap representadas na Figura 10, com linhas de tamanho $W \times \text{sizeof}(\text{uint4})$, o valor de ϵ é ser obtido através do seguinte desenvolvimento:

$$2 \times \epsilon \times W \times \text{sizeof}(\text{uint4}) + W \times \text{sizeof}(\text{uint4}) + \epsilon \times \text{sizeof}(\text{uint3}) = \text{memGPU}$$

$$2 \times \epsilon \times W \times \text{sizeof}(\text{uint4}) + \epsilon \times \text{sizeof}(\text{uint3}) = \text{memGPU} - W \times \text{sizeof}(\text{uint4})$$

$$\epsilon \times (2 \times W \times \text{sizeof}(\text{uint4}) + \text{sizeof}(\text{uint3})) = \text{memGPU} - W \times \text{sizeof}(\text{uint4})$$

$$\epsilon = \frac{\text{memGPU} - W \times \text{sizeof}(\text{uint4})}{2 \times W \times \text{sizeof}(\text{uint4}) + \text{sizeof}(\text{uint3})}$$

Entende-se que uma vez que um ramo tenha sido calculado completamente (atingido o nó folha), este poderá ser removido da árvore de prefixos, reduzindo assim o uso de memória principal.

Como podem ocorrer iterações em que a quantidade de conjuntos candidatos seja extremamente baixa, esta implementação optou pela estratégia de alocação de *threads* por partes, conforme apresentado na Figura 8.

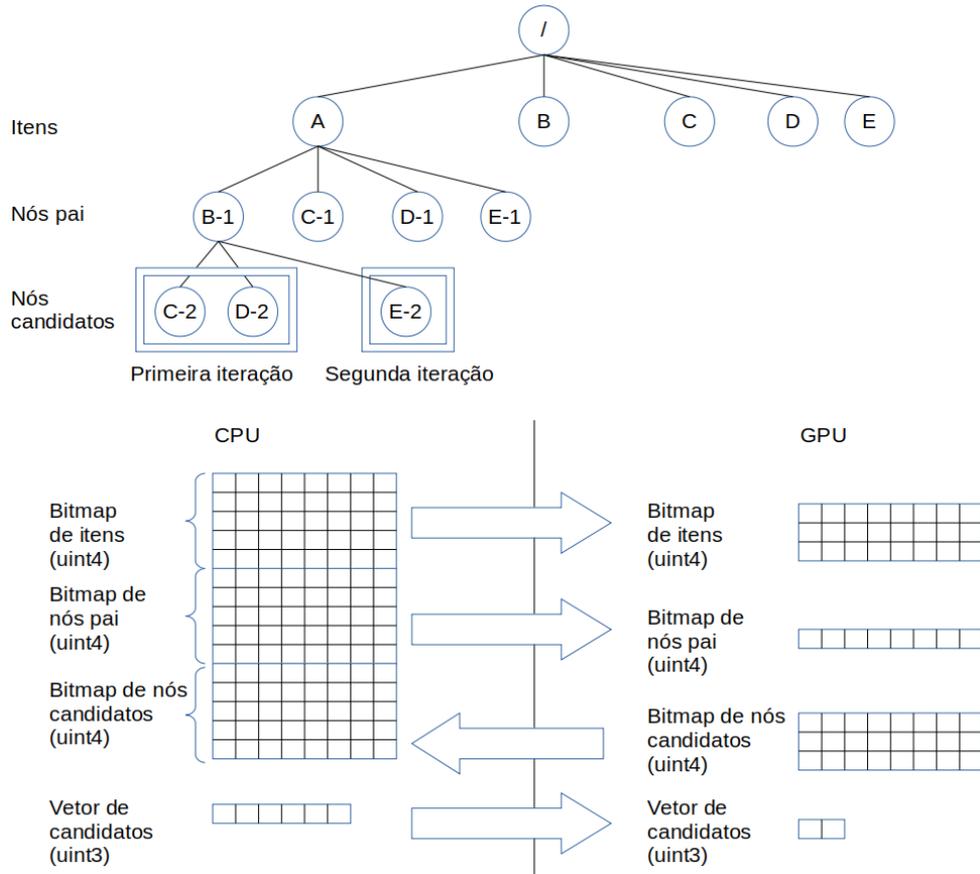


Figura 10 – Esquema lógico da implementação Apriori-Bitmap-Cuda. A identificação de conjuntos frequentes é feita executando o cálculo do valor de suporte em GPU de maneira dinâmica. Fonte: Criado pelo autor.

A Figura 11 apresenta o algoritmo desta implementação.

O propósito principal desta aplicação é acelerar o cálculo do valor de suporte pelo uso da GPU, porém respeitando os limites de memória do acelerador, o que é feito pela divisão de subiterações. A penalidade desta implementação, entretanto, é o custo de transferência de dados entre *host* e GPU.

4.1.2 Apriori-Simd-Parallel

Esta implementação desenvolvida foi denominada Apriori-Simd-Parallel e apresenta como principal característica a execução do cálculo de suporte (terceiro passo) empregando instruções vetoriais em CPU e paralelismo em memória compartilhada. Para o desenvolvimento desta implementação foram utilizadas instruções AVX2.

Da mesma forma que na implementação descrita na seção 4.1.1, a navegação pela árvore de prefixos é feita de forma balanceada porém sem a necessidade de subiterações. Além disso, uma vez que um ramo tenha sido calculado completamente (atingido o nó folha), este poderá ser removido da árvore de prefixos, reduzindo assim o uso de memória principal.

```

Entrada  : Conjunto de dados, valor de suporte mínimo
Saída   : Arquivo contendo conjuntos frequentes e respectivos valores de
           suporte
1 Realiza a leitura do arquivo de conjunto de dados;
2 Identifica os possíveis elementos distintos, e calcula os respectivos valores de
  suporte;
3 Identifica elementos únicos frequentes e descarta os demais;
4 Escreve os elementos frequentes e seus valores de suporte no arquivo de saída;
5 Transforma em paralelo o conjunto de dados de entrada em uma representação
  bitmap residente em memória principal considerando apenas os elementos
  frequentes;
6 A partir dos elementos frequentes, de maneira balanceada, gera possíveis
  candidatos com dois elementos;
7 Enquanto existir candidatos gerados faça
8   De acordo com o tamanho da memória da GPU, identifica a quantidade de
   subiterações;
9   Enquanto existir subiterações para cálculo faça
10  |   Envia as respectivas linhas bitmap para a GPU;
11  |   Calcula as linhas bitmap relativas aos candidatos da subiteração em GPU;
12  |   Obtém o valor de suporte através da soma de bits verdadeiros em GPU;
13  |   Envia as linhas bitmap calculadas e valores de suporte para o host;
14  fim
15  Identifica os candidatos frequentes e descarta os demais;
16  Escreve os candidatos frequentes e seus valores de suporte no arquivo de saída;
17  A partir dos candidatos frequentes, gera candidatos em paralelo com
   cardinalidade superior, seguindo a abordagem balanceada de navegação pela
   árvore de prefixos;
18 fim

```

Figura 11 – Algoritmo Apriori-Bitmap-Cuda.

Nesta implementação também optou-se pela estratégia de alocação de *threads* por partes, conforme apresentado na Figura 8.

A Figura 12 apresenta o algoritmo desta implementação.

O propósito principal desta aplicação é atender cenários em que o custo de transferência de dados entre *host* e GPU da implementação anterior se mostrou ineficiente. Esta implementação, embora seja feita somente em CPU, apresenta foco em acelerar o processo de mineração ao utilizar recursos de vetorização e explorar paralelismo em memória compartilhada.

Entrada	: Conjunto de dados, valor de suporte mínimo
Saída	: Arquivo contendo conjuntos frequentes e respectivos valores de suporte
1	Realiza a leitura do arquivo de conjunto de dados;
2	Identifica os possíveis elementos distintos, e calcula os respectivos valores de suporte;
3	Identifica elementos únicos frequentes e descarta os demais;
4	Escreve os elementos frequentes e seus valores de suporte no arquivo de saída;
5	Transforma em paralelo o conjunto de dados de entrada em uma representação bitmap residente em memória principal considerando apenas os elementos frequentes;
6	A partir dos elementos frequentes, de maneira balanceada, gera possíveis candidatos com dois elementos;
7	Enquanto existir candidatos gerados faça
8	Calcula as linhas bitmap relativas aos candidatos em paralelo e empregando instruções vetoriais ;
9	Obtém o valor de suporte através da soma de bits verdadeiros em paralelo e empregando instruções vetoriais ;
10	Identifica os candidatos frequentes e descarta os demais;
11	Escreve os candidatos frequentes e seus valores de suporte no arquivo de saída;
12	A partir dos candidatos frequentes, gera candidatos em paralelo com cardinalidade superior, seguindo a abordagem balanceada de navegação pela árvore de prefixos;
13	fim

Figura 12 – Algoritmo Apriori-Simd-Parallel.

4.1.3 Apriori-Roaring-Parallel

A implementação desenvolvida denominada Apriori-Roaring-Parallel apresenta como principal característica um uso muito baixo de memória devido ao emprego de estruturas bitmap comprimidas. Esta implementação é completamente executada em CPU.

Existem diversos tipos de compressão bitmap que foram descritos nas seções anteriores, e para esta implementação foi adotada a compressão Roaring, que segundo estudos comparativos apresentou destaque frente às demais.

Nesta implementação, devido à uma demanda menor de memória para a estrutura bitmap, a navegação pela árvore de prefixos é feita em largura, o que permite melhor reaproveitamento das linhas já calculadas da estrutura bitmap. Além disso, somente são mantidos em memória o primeiro nível e o nível atual em mineração da árvore, removendo níveis intermediários para os quais já foram calculados os valores de suporte dos candidatos.

Em relação à alocação de *threads*, optou-se pela estratégia de alocação de *threads*

por candidatos, conforme apresentado na Figura 9. Esta escolha se deve ao fato de que o paralelismo foi realizado em memória compartilhada, e a quantidade de *threads* disponíveis na maior parte do processo de mineração será inferior à quantidade de candidatos a serem calculados.

A Figura 13 apresenta o algoritmo desta implementação.

Entrada	: Conjunto de dados, valor de suporte mínimo
Saída	: Arquivo contendo conjuntos frequentes e respectivos valores de suporte

```

1 Realiza a leitura do arquivo de conjunto de dados;
2 Identifica os possíveis elementos distintos, e calcula os respectivos valores de
  suporte;
3 Identifica elementos únicos frequentes e descarta os demais;
4 Escreve os elementos frequentes e seus valores de suporte no arquivo de saída;
5 Cria um conjunto de dados em que exista uma linha para cada item frequente,
  contendo as transações em que está presente;
6 Transforma em paralelo este conjunto de dados em um bitmap Roaring
  residente em memória;
7 A partir dos elementos frequentes, gera possíveis candidatos com dois elementos;
8 Enquanto existir candidatos gerados faça
9   Para cada elemento pai faça em paralelo
10    |   Calcula a linha bitmap relativa ao elemento pai atual;
11    |   Para cada elemento filho do pai atual faça
12    |   |   Calcula a linha bitmap relativa ao elemento filho atual;
13    |   |   Obtém o valor de suporte através da soma de bits verdadeiros;
14    |   fim
15  fim
16 Identifica os candidatos frequentes e descarta os demais;
17 Escreve os candidatos frequentes e seus valores de suporte no arquivo de saída;
18 A partir dos candidatos frequentes, gera candidatos em paralelo com
  cardinalidade superior;
19 fim

```

Figura 13 – Algoritmo Apriori-Roaring-Paralelo.

O propósito principal desta implementação é atender cenários em que há uma grande demanda de uso de memória, o que é alcançado ao empregar estruturas de bitmap comprimido. Devido à esta característica, esta implementação irá naturalmente apresentar um tempo de execução maior frente às demais.

4.1.4 Apriori-Cuda-Single-Memcpy

Esta implementação apresenta como principal característica a execução do cálculo de suporte (terceiro passo) em GPU, porém reduzindo a quantidade de transferências de

conjuntos de dados bitmap à apenas uma. Um esquema de funcionamento desta implementação é apresentado na Figura 14.

A árvore de prefixos é residente em memória, e com o intuito de reduzir a quantidade de transferências de vetores de candidatos entre *host* e GPU, a navegação realizada nesta é feita em largura.

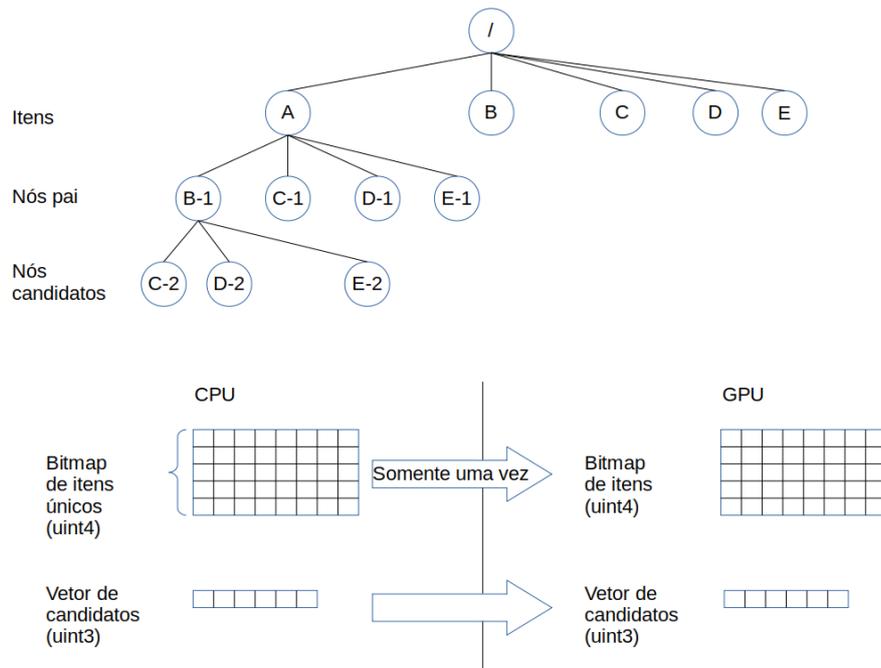


Figura 14 – Esquema lógico da implementação Apriori-Cuda-Single-Memcpy, em que a identificação de conjuntos frequentes é feita executando o cálculo do valor de suporte em GPU realizando apenas uma única transferência de estrutura bitmap. Fonte: Criado pelo autor.

Somente o conjunto de dados bitmap original, que está relacionado aos itens frequentes únicos, e o vetor de índices dos candidatos da iteração atual estarão residentes na memória da GPU. Com exceção de conjuntos vizinhos (apresentam o mesmo conjunto pai), a cada cálculo de candidatos deverão ser realizadas diversas operações **AND** para obter a linha correspondente ao conjunto a ser calculado o valor de suporte.

É importante notar que nesta implementação o elemento restritivo deixa de ser a memória principal, e torna-se a memória da GPU.

Com o objetivo de melhorar a taxa de ocupação da GPU, esta implementação optou pela estratégia de alocação de *threads* por partes, conforme apresentado na Figura 8.

A Figura 15 apresenta o algoritmo desta implementação.

O propósito principal desta aplicação é acelerar o cálculo do valor de suporte pelo uso da GPU e eliminar a penalidade do custo de transferência de dados entre *host* e GPU. A desvantagem, entretanto, é que a restrição torna-se a quantidade de memória da GPU, que geralmente é inferior à do *host*.

Entrada	: Conjunto de dados, valor de suporte mínimo
Saída	: Arquivo contendo conjuntos frequentes e respectivos valores de suporte
1	Realiza a leitura do arquivo de conjunto de dados;
2	Identifica os possíveis elementos distintos, e calcula os respectivos valores de suporte;
3	Identifica elementos únicos frequentes e descarta os demais;
4	Escreve os elementos frequentes e seus valores de suporte no arquivo de saída;
5	Transforma em paralelo o conjunto de dados de entrada em uma representação bitmap residente em memória principal considerando apenas os elementos frequentes;
6	Envia o conjunto de dados bitmap para a GPU;
7	A partir dos elementos frequentes, de maneira balanceada, gera possíveis candidatos com dois elementos;
8	Enquanto <i>existir candidatos gerados</i> faça
9	Envia o vetor de candidatos para cálculo para a GPU;
10	Calcula as linhas bitmap relativas aos candidatos em GPU;
11	Obtém o valor de suporte através da soma de bits verdadeiros em GPU;
12	Envia o vetor de valores de suporte para o <i>host</i> ;
13	Identifica os candidatos frequentes e descarta os demais;
14	Escreve os candidatos frequentes e seus valores de suporte no arquivo de saída;
15	A partir dos candidatos frequentes, gera candidatos em paralelo com cardinalidade superior, seguindo a abordagem em largura de navegação pela árvore de prefixos;
16	fim

Figura 15 – Algoritmo Apriori-Cuda-Single-Memcpy.

4.2 Etapa 2: Mineração de regras negativas

Conforme descrito na seção de fundamentação teórica, existem diversas abordagens para geração de regras negativas, podendo ser exatas ou baseadas em alguma medida estatística.

Entende-se, portanto, que a quantidade de regras geradas por cada uma delas não necessariamente será a mesma, ou seja, a saída das diversas implementações não é igual.

Como o trabalho está focado em uso de recursos computacionais durante o processo de mineração, será adotada a abordagem exata e contemplando mecanismos de poda descrita por (CORNELIS et al., 2006), e denominada PNAR. Neste estudo, os autores propõem duas definições de regras negativas, e iremos utilizar a definição em que dada uma regra positiva $X \rightarrow Y$, existem três possíveis casos de regras negativas a serem analisados: $X \rightarrow \bar{Y}$, $\bar{X} \rightarrow Y$ e $\bar{X} \rightarrow \bar{Y}$.

No método PNAR, para que uma determinada regra de associação $C_1 \Rightarrow C_2$, $C_1 \in$

$\{X, \neg X\}, C_2 \in \{Y, \neg Y\}, X, Y \subset I, X \cap Y = \emptyset$ seja considerada forte, esta deverá atender os seguintes critérios:

1. $\text{supp}(C_1 \Rightarrow C_2) \geq ms$
2. $\text{supp}(X) \geq ms, \text{supp}(Y) \geq ms$
3. $\text{conf}(C_1 \Rightarrow C_2) \geq mc$
4. $C_1 \Rightarrow C_2$ seja mínimo em relação ao suporte de seus conjuntos de itens negativos

onde ms é o valor de suporte mínimo e mc é o valor de confiança mínima.

Este quarto critério refere-se ao mecanismo de poda empregado pelos autores, que implica que se $C_1 = \neg X$, então não irá existir um conjunto $X' \subset X$ tal que $\text{supp}(\neg X' \Rightarrow C_2) \geq ms$. Este critério indica que somente devem ser consideradas regras de “fronteira” do conjunto de possíveis regras de associação negativas, pois as demais regras serão redundantes e com pouco interesse. A origem deste critério está relacionada com a propriedade anti-monotônica de valores de suporte de conjuntos de itens negativos.

Nesta definição os valores de suporte são calculados de maneira similar ao caso somente positivo, discriminando esta etapa e também o processo de geração de regras em quatro partes, conforme representado na Figura 16.

Durante nossa implementação, serão identificados subprocessos do método onde será realizado o processo de paralelização, de forma que a mineração de regras de associação seja acelerada.

De maneira resumida, o procedimento geral do método PNAR é o seguinte:

1. Identificar todos os conjuntos de itens frequentes somente positivos
2. Identificar todos os conjuntos de itens frequentes negativos da parte P_2
3. Identificar todos os conjuntos de itens frequentes negativos da parte P_3
4. Identificar todos os conjuntos de itens frequentes negativos da parte P_4
5. Gerar todas as regras de associação positivas e negativas fortes

A primeira destas partes (P_1) refere-se ao caso de conjuntos somente positivos, e para esta mineração de conjuntos frequentes serão utilizadas as abordagens comentadas na seção anterior.

Para a mineração da segunda parte (P_2), o método PNAR obtém estes conjuntos frequentes diretamente do resultado somente positivo, conforme descrito no estudo:

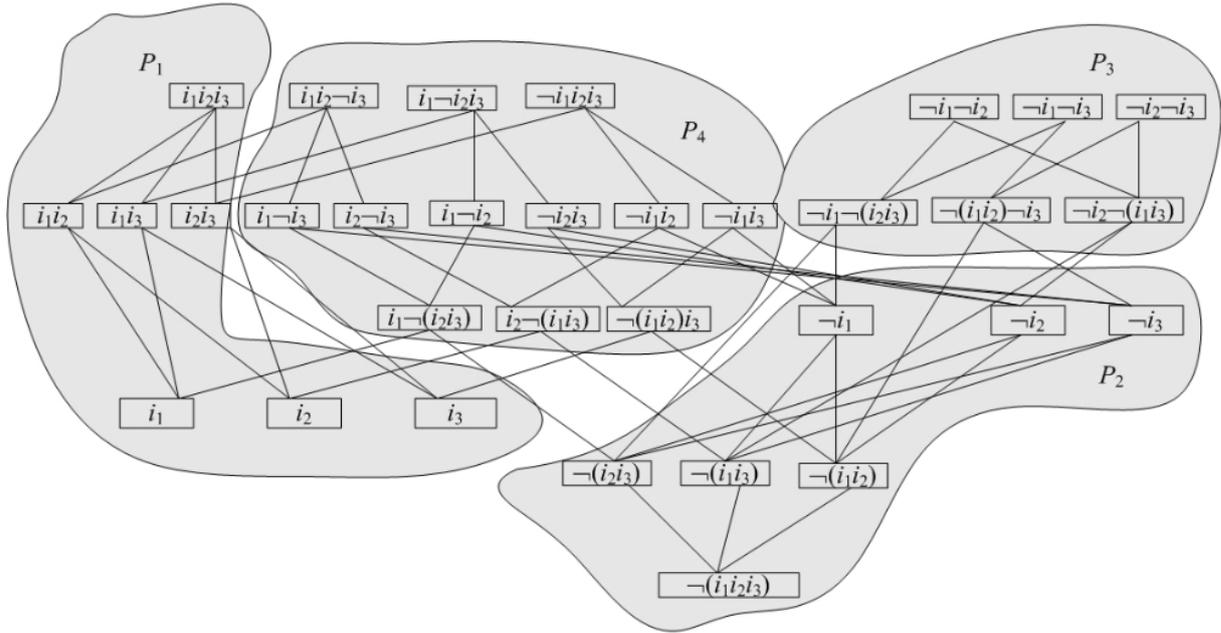


Figura 16 – Partição do processo de mineração de regras de associação em quatro partes. Fonte: Adaptado de (CORNELIS et al., 2006).

Para todos os conjuntos de itens I frequentes em P_1 , insira a versão negativa de I no conjunto de itens frequentes em P_2 se $1 - \text{supp}(I) \geq ms$, onde ms é o valor de suporte mínimo.

Esta é uma etapa simples do método, em que não se justifica realizar sua paralelização. Observe que a quantidade de conjuntos frequentes identificados nesta segunda parte depende do número de conjuntos frequentes identificados na parte somente positiva, e além disso, somente serão identificados conjuntos frequentes caso o valor de suporte mínimo definido seja inferior à 50 %.

Para realizar a mineração das partes P_3 e P_4 , os autores propõem dois algoritmos que serão descritos na sequência. Entretanto, antes de comentá-los, iremos expor a seguinte notação que foi empregada no estudo do PNAR:

- X, Y : conjuntos de itens positivos
- I, I' : conjuntos de itens, que podem ser positivos ou negativos
- $L(P_i)$: conjunto de itens frequentes da parte i
- $L(P_i)_k$: conjunto de itens frequentes de cardinalidade k da parte i
- $L(P_4)_{k,p}$: durante a mineração da parte 4, conjunto de itens frequentes com cardinalidade negativa k e cardinalidade positiva p

- $C(P_i)$: conjunto de itens candidatos na parte i (analogamente: $C(P_i)_k, C(P_4)_{k,p}$)
- $S(P_i)$: conjunto de itens positivos para os quais será necessário realizar o cálculo de suporte (analogamente: $S(P_i)_k, S(P_4)_{k,p}$)
- S : conjunto contendo subconjuntos de itens positivos para os quais o valor de suporte é conhecido. Este conjunto é composto pelos subconjuntos $L(P_1)$, $C(P_1)$ e $S(P_i)$

Na mineração dos conjuntos frequentes da terceira parte P_3 , os autores apresentam um algoritmo que utiliza resultados das partes anteriores e minera conjuntos frequentes apenas negativos, conforme apresentado na Figura 17.

```

Entrada  :  $L(P_1)$ ,  $S$ , Valor de suporte mínimo ( $ms$ )
Saída   :  $L(P_3)$ 

1  $L(P_3) = \emptyset, C(P_3) = \emptyset, S(P_3) = \emptyset;$ 
2  $C(P_3)_2 = \{\neg\{i_1\}\neg\{i_2\} | i_1, i_2 \in L(P_1)_1, i_1 \neq i_2\};$ 
3 Para  $\{k = 2; C(P_3)_k \neq \emptyset; k++\}$  faça
4   Para todo  $I = \neg X \neg Y \in C(P_3)_k$  faça
5     Se  $supp(I) \geq ms$  então
6       | Insira  $I$  no conjunto  $L(P_3)_k;$ 
7     senão
8       Para todo  $i \in L(P_1)_1, i \notin XY$  faça
9         |  $Cand = \{\neg(X\{i\})\neg Y\};$ 
10        Se  $X\{i\} \in L(P_1)$  e  $\nexists(\neg X' \neg Y') \in L(P_3), X' \subseteq X\{i\}, Y' \subseteq Y$  então
11          |  $C(P_3)_{k+1} = C(P_3)_{k+1} \cup Cand;$ 
12        fim
13         $Cand = \{\neg X \neg(Y\{i\})\};$ 
14        Se  $Y\{i\} \in L(P_1)$  e  $\nexists(\neg X' \neg Y') \in L(P_3), X' \subseteq X, Y' \subseteq Y\{i\}$  então
15          |  $C(P_3)_{k+1} = C(P_3)_{k+1} \cup Cand;$ 
16        fim
17        Se  $Cand \neq \emptyset, XY\{i\} \notin S, (\nexists I' \subset XY\{i\})(supp(I') = 0)$  então
18          | Insira  $XY\{i\}$  no conjunto  $S(P_3)_{k+1};$ 
19        fim
20      fim
21    fim
22  fim
23  Calcule em paralelo o valor de suporte dos conjuntos presentes em  $S(P_3)_{k+1};$ 
24   $S = S \cup S(P_3)_{k+1};$ 
25 fim

```

Figura 17 – Algoritmo para obtenção dos conjuntos frequentes da terceira parte do método PNAR. Fonte: Adaptado de (CORNELIS et al., 2006).

Nesta etapa são utilizadas as informações de conjuntos positivos frequentes ($L(P_1)$) e conjuntos positivos calculados que foram obtidas nas etapas anteriores (S), e como saída será obtido o conjunto de itens frequentes $L(P_3)$.

Esta etapa em si gera candidatos negativos a partir das possíveis combinações de conjuntos frequentes positivos, desde que atendam o critério de suporte mínimo.

Caso um candidato não seja frequente, ele poderá gerar novos candidatos com cardinalidade superior (linhas 9 e 13) agregando itens únicos positivos frequentes (i) que ainda não façam parte do conjunto candidato. Estes candidatos gerados deverão atender algumas condições de poda, que são especificadas pelas linhas 10, 14 e 17.

Após isto, para alguns destes candidatos gerados poderá ser necessário conhecer o valor de suporte de conjuntos de itens positivos ($S(P_3)_{k+1}$), que serão calculados posteriormente. Repare que o único trecho que foi paralelizado desta etapa é o cálculo do valor de suporte dos conjuntos, dado sua relevância no tempo de execução.

A mineração da parte P_4 é a última etapa envolvendo a identificação de conjuntos frequentes. Nesta, o objetivo é identificar conjuntos que contenham ambos itens positivos e negativos através do algoritmo apresentado na Figura 18, que utiliza resultados das partes anteriores.

Semelhante à mineração da parte 3, esta etapa utiliza informações obtidas em etapas anteriores, como a de conjuntos positivos frequentes ($L(P_1)$) e conjuntos positivos calculados (S). A saída do algoritmo será o conjunto de itens frequentes $L(P_4)$.

Inicialmente são gerados candidatos a partir dos conjuntos frequentes positivos, e posteriormente a geração de candidatos com cardinalidade superior é feita a partir dos que foram identificados como frequentes. Repare que estes candidatos poderão ter cardinalidade incrementada de suas partes negativa (linha 3) e positiva (linha 4).

A geração de candidatos pela expansão da parte positiva (linhas 10 à 19) apresenta um destaque na composição do tempo de execução, o que implicou na paralelização deste trecho para acelerar o processo de mineração. Por outro lado, a geração de candidatos considerando a expansão da parte negativa (linhas 23 à 27) não apresenta relevância no tempo de execução total, não sendo interessante sua paralelização.

Outro trecho que também foi paralelizado é o de cálculo do valor de suporte de subconjuntos positivos (linhas 20 e 21), que é uma tarefa dispendiosa computacionalmente.

Para atender os critérios de geração de regras de “fronteira”, são aplicados mecanismos de poda para evitar gerar candidatos desnecessários, conforme é visto nas linhas 13, 15 e 24.

Após a identificação de conjuntos frequentes para cada uma das partes, a próxima etapa do método é realizar a geração de regras. Nesta etapa final, cada um dos conjuntos é analisado separadamente e são geradas as regras de associação (conjunto R), conforme representado na Figura 19.

```

Entrada :  $L(P_1)$ ,  $S$ , Valor de suporte mínimo ( $ms$ )
Saída :  $L(P_4)$ 

1  $L(P_4) = \emptyset$ ,  $C(P_4) = \emptyset$ ,  $S(P_4) = \emptyset$ ;
2  $C(P_4)_{1,1} = \{\neg\{i_1\}\{i_2\} | i_1, i_2 \in L(P_1)_1, i_1 \neq i_2\}$ ;
3 Para  $\{k = 1; C(P_4)_{k,1} \neq \emptyset; k++\}$  faça
4   Para  $\{p = 1; C(P_4)_{k,p} \neq \emptyset; p++\}$  faça
5     Para todo  $I \in C(P_4)_{k,p}$  faça
6       Se  $supp(I) \geq ms$  então
7         | Insira  $I$  no conjunto  $L(P_4)_{k,p}$ ;
8       fim
9     fim
10    Para toda possível união  $I_1, I_2 \in L(P_4)_{k,p}$  faça em paralelo
11       $X = I_1.negative, Y = I_1.positive \cup I_2.positive$ ;
12       $I = \neg XY$ ;
13      Se  $(\nexists X' \subset X)(supp(\neg X'Y) \geq ms)$  e  $(\nexists Y' \subset Y)(supp(\neg XY') < ms)$ 
14        então
15          | Insira  $I$  no conjunto  $C(P_4)_{k,p+1}$ ;
16          Se  $XY \notin S$  e  $\nexists I' \subset XY, supp(I') = 0$  então
17            | Insira  $XY$  no conjunto  $S(P_4)_{k,p+1}$ ;
18            fim
19          fim
20        Calcule em paralelo o valor de suporte dos conjuntos presentes em
21           $S(P_4)_{k,p+1}$ ;
22           $S = S \cup S(P_4)_{k,p+1}$ ;
23        fim
24      Para todo  $X \in L(P_1)_{k+1}, i \in L(P_1)_1$  faça
25        | Se  $(\nexists X' \subset X)(\neg X'\{i\} \in L(P_4))$  então
26          |  $C(P_4)_{k+1,1} = C(P_4)_{k+1,1} \cup \neg X\{i\}$ ;
27          fim
28      fim

```

Figura 18 – Algoritmo para obtenção dos conjuntos frequentes da quarta parte do método PNA. Fonte: Adaptado de (CORNELIS et al., 2006).

A geração de regras para as duas primeiras partes seguem o processo descrito na Fundamentação teórica, em que é empregada uma árvore de geração de regras que incrementa o termo consequente a cada iteração (Figura 5).

Por se tratar de uma etapa simples, não foi aplicada paralelização nesta, dado sua pouca relevância no tempo de execução total.

Um ponto importante a ser destacado nesta implementação, é que por definição a geração de regras negativas está limitada à quantidade de regras positivas identificadas, o que de certa forma representa um limite na geração de regras, e portanto de conhecimento

Entrada : $L(P_1)$, $L(P_2)$, $L(P_3)$ e $L(P_4)$, Valor de confiança mínima (mc)
Saída : R

- 1 Gera todas as regras de associação positivas e insere no conjunto R ;
- 2 Gera todas as regras de associação negativas referentes aos conjuntos frequentes da parte 2 ($L(P_2)$) e insere no conjunto R ;
- 3 **Para todo** $I = \neg X \neg Y \in L(P_3)$ **faça**
- 4 **Se** $conf(\neg X \Rightarrow \neg Y) \geq mc$ **então**
- 5 | Insira $\neg X \Rightarrow \neg Y$ no conjunto R ;
- 6 **fim**
- 7 **fim**
- 8 **Para todo** $I = \neg XY \in L(P_4)$ **faça**
- 9 **Se** $conf(X \Rightarrow \neg Y) \geq mc$ **então**
- 10 | Insira $X \Rightarrow \neg Y$ no conjunto R ;
- 11 **fim**
- 12 **Se** $conf(\neg X \Rightarrow Y) \geq mc$ **então**
- 13 | Insira $\neg X \Rightarrow Y$ no conjunto R ;
- 14 **fim**
- 15 **fim**

Figura 19 – Algoritmo para geração de regras de associação do método PNAAR. Fonte: Adaptado de (CORNELIS et al., 2006).

obtido do conjunto de dados.

4.3 Conjuntos de dados

Para avaliação das implementações desenvolvidas foram utilizados conjuntos de dados públicos disponíveis no repositório *Frequent Itemset Mining Dataset* (FIMI) (JR.; GOETHALS; ZAKI, 2005) e também conjuntos de dados sintéticos gerados através da ferramenta IBM Quest (SOFTLAND, 2010).

Estes conjuntos de dados apresentam a estrutura de transações descrita na seção de fundamentação teórica, e foram escolhidos pois, durante o processo de revisão sistemática, foi constatada sua utilização por diversos estudos. Entende-se que a seleção destes conjuntos de dados implica à este trabalho uma característica de reprodutibilidade dos experimentos realizados e base de comparação com outros estudos.

Para classificação de conjuntos de dados são utilizadas algumas métricas como quantidade de transações, quantidade de itens distintos e tamanho médio das transações.

4.3.1 Repositório FIMI de conjuntos de dados

O repositório FIMI apresenta diversos conjuntos de dados para realização de experimentos. Neste projeto de mestrado, foram selecionados seis conjuntos, os quais serão

Tabela 3 – Características dos conjuntos de dados do repositório FIMI utilizados. Fonte: Criado pelo autor.

Conjunto de dados	Transações	Itens	Tamanho médio das transações
chess	3,196	75	37
pumsb	49,046	2113	74
kosarak	80,769	7,116	50
accidents	340,183	468	33
webdocs	1,692,082	5,267,656	177
retail	88,162	16,470	13

descritos a seguir e também pelos dados contidos na Tabela 3.

- **chess:** é um conjunto de dados originalmente presente no repositório da Universidade da Califórnia em Irvine (UCI), mas que foi convertido para o formato de transações utilizado em mineração de regras de associação por Roberto Bayardo. Este conjunto de dados apresenta uma característica de ser denso.
- **pumsb:** é um conjunto de dados que contém informações do censo para população e habitação (PUMS - Public Use Microdata Sample) e foi introduzido no repositório FIMI por Roberto Bayardo. Este conjunto de dados apresenta a característica de ser denso.
- **kosarak:** trata-se de um conjunto de dados de registro de cliques de usuários de um portal de notícias húngaro. Foi elaborado por Ferenc Bodon e sua característica é de um conjunto de dados esparsos.
- **accidents:** é um conjunto de dados doado por Karolien Geurts que apresenta dados anônimos de acidentes de trânsito. Trata-se de um conjunto de dados classificado entre denso e esparsos.
- **webdocs:** é um conjunto de dados elaborado por Claudio Lucchese, Salvatore Orlando, Raffaele Perego e Fabrizio Silvestri, que utilizaram documentos web escritos em linguagem HTML. Durante sua elaboração, foram removidas tags HTML e palavras comuns (*stopwords*). Trata-se de um conjunto de dados grande e esparsos.
- **retail:** este conjunto descreve dados anônimos de compras de um mercado belga durante três períodos não consecutivos. É considerado um conjunto de dados esparsos.

4.3.2 Ferramenta IBM Quest para geração de conjuntos de dados

IBM Quest é uma ferramenta para geração de conjuntos de dados que simulam transações de usuários em um ambiente de varejo. Através dela é possível variar parâme-

Tabela 4 – Características dos conjuntos de dados sintéticos utilizados. Fonte: Criado pelo autor.

Conjunto de dados	Transações	Itens	Tamanho médio das transações
T10000L200N1000	10^4	1000	200
T100000L200N1000	10^5	1000	200
T1000000L200N1000	10^6	1000	200
T10000000L200N1000	10^7	1000	200
T100000000L200N1000	10^8	1000	200
T10000L300N1000	10^4	1000	300
T100000L300N1000	10^5	1000	300
T1000000L300N1000	10^6	1000	300
T10000000L300N1000	10^7	1000	300
T100000000L300N1000	10^8	1000	300

tros do conjunto de dados como quantidade de transações, número médio de itens por transação e quantidade de itens distintos.

Além de gerar conjuntos de dados para a tarefa de identificação de padrões frequentes, esta ferramenta também permite criar conjuntos para a tarefa de mineração de sequências frequentes.

Durante o projeto de mestrado, a ferramenta tem sido utilizada para geração de conjuntos de dados maiores que os disponíveis publicamente. A Tabela 4 apresenta as características dos conjuntos de dados gerados com esta ferramenta.

5 Experimentos e resultados

Nesta seção são apresentados os resultados obtidos. Os experimentos foram divididos em três grupos com os seguintes propósitos:

1. Comparar as representações de dados mencionadas em razão de uso de memória
2. Comparar implementações de identificação de conjuntos frequentes
3. Avaliar características do método desenvolvido para mineração de regras negativas

5.1 Representações de dados

Neste experimento buscou-se avaliar o impacto no uso de memória ao utilizar representações bitmap para conjuntos de dados de transações. Na Tabela 5 é apresentado um comparativo de uso de memória considerando cinco situações:

1. Representação por transações (original)
2. Representação por bitmap simples (sem compressão)
3. Representação por bitmap Roaring
4. Representação por bitmap simples com mecanismo de poda de elementos únicos não frequentes
5. Representação por bitmap Roaring com mecanismo de poda de elementos únicos não frequentes

O mecanismo de poda mencionado consiste em remover linhas da representação bitmap que estejam relacionadas à elementos únicos não frequentes, que é uma das primeiras tarefas realizada nas implementações desenvolvidas. Neste experimentos utilizamos o valor de suporte mínimo de 50 %.

Esses resultados indicam o problema de alto uso de memória ao converter conjuntos de dados esparsos em representação de bitmap simples. Nesses casos, o emprego de uma representação de bitmap com compressão reduz a demanda de memória.

Além disso, o uso do mecanismo de poda permite reduzir drasticamente o tamanho dos conjuntos de dados considerados, de forma que conjuntos maiores possam ser minerados. É claramente visível que esta redução será tão maior quanto seja o valor definido do suporte mínimo (ZAKI; JR, 2014).

Tabela 5 – Comparativo de tamanhos de conjuntos de dados.

Conjunto de dados	Original	Bitmap simples	Roaring	Poda+Bitmap simples	Poda+Roaring
chess	335 KB	234 KB (69.8 %)	285 KB (85.0 %)	115.4 KB (34.4 %)	187 KB (55.8 %)
pumsb	16 MB	98.83 MB (617.6 %)	7.71 MB (48.1 %)	2.43 MB (15.1 %)	416.8 KB (2.6 %)
kosarak	31 MB	38.05 GB (122,741.9 %)	26.78 MB (86,3 %)	587.28 KB (1.8 %)	128.13 KB (0.4 %)
accidents	33.9 MB	151.8 MB (447.7 %)	27.5 MB (81.1 %)	7.7 MB (22.7 %)	1.13 MB (3.3 %)
webdocs	1.38 GB	8.1 TB (586,956.5 %)	657.5 MB (47.6 %)	7.41 MB (0.5 %)	1.01 MB (0.07 %)
retail	4.0 MB	1.35 GB (33750 %)	3.01 MB (75.2 %)	49.4 KB (1.2 %)	16.02 KB (0.4 %)
T10000L200N1000	7.4 MB	9.2 MB (124.3 %)	3.95 MB (53.3 %)	625 KB (8.4 %)	513.06 KB (6.9 %)
T100000L200N1000	74 MB	92.02 MB (124.3 %)	39.6 MB (53.5 %)	6.10 MB (8.2 %)	1.00 MB (1.3 %)
T1000000L200N1000	739 MB	920.29 MB (124.5 %)	396.0 MB (53.5 %)	61.03 MB (8.2 %)	8.00 MB (1.0 %)
T10000000L200N1000	7.3 GB	8.98 GB (123.0 %)	3.86 GB (52.8 %)	610.35 MB (8.3 %)	76.57 MB (1.0 %)
T100000000L200N1000	73 GB	89.87 GB (123.1 %)	38.6 GB (52.8 %)	5.96 GB (8.1 %)	763.7 MB (1.0 %)
T10000L300N1000	12 MB	9.2 MB (76.6 %)	5.85 MB (48.7 %)	1.92 MB (16.0 %)	1.58 MB (13.1 %)
T100000L300N1000	111 MB	92.02 MB (82.9 %)	58.59 MB (52.7 %)	19.16 MB (17.2 %)	3.14 MB (2.8 %)
T1000000L300N1000	1.1 GB	920.29 MB (83.6 %)	585.80 MB (53.2 %)	191.68 MB (17.4 %)	25.15 MB (2.2 %)
T10000000L300N1000	11 GB	8.98 GB (81.6 %)	5.72 GB (52.0 %)	1.87 GB (17.0 %)	240.49 MB (2.1 %)
T100000000L300N1000	109 GB	89.87 GB (82.4 %)	57.21 GB (52.4 %)	18.71 GB (17.1 %)	2.34 GB (2.1 %)

A característica do conjunto de dados de entrada também determina quão eficiente será sua taxa de compressão por meio do mecanismo de poda. De forma geral, conjuntos de dados esparsos apresentarão uma melhor taxa de compressão quando comparados a conjuntos de dados densos.

Neste experimento foi avaliada a compressão Roaring devido ao fato de ter sido utilizada na implementação do Apriori-Roaring-Parallel. É importante enfatizar que nesta implementação, a representação de conjunto de dados comprimido é o principal componente da demanda de uso de memória, e portanto, determinar esse valor para um certo conjunto de dados especifica o requisito computacional do tamanho da memória para a execução da implementação.

5.2 Identificação de conjuntos frequentes

Para verificar o desempenho do método de identificação de conjuntos frequentes desenvolvido, foram realizados experimentos em três análises. A primeira delas avalia o pico de uso de memória das implementações, identificando em qual cenário é mais adequado o uso de cada uma delas.

Em uma segunda análise foram realizados experimentos com os mesmos cenários desta primeira, porém foram comparados os tempos de execução. Nesta são consideradas cinco execuções de cada configuração, expondo nos gráficos os valores médios.

Por fim, uma terceira análise buscou identificar a escalabilidade das implementações que apresentam paralelismo em memória compartilhada (Apriori-Roaring-Parallel e Apriori-Simd-Parallel) ao variar a quantidade de *threads* para execução.

Durante estas análises, foram utilizados valores de suporte mínimo que, ou eram

limites para a mineração de cada conjunto de dados para o ambiente computacional utilizado, ou proporcionavam destaque para os trechos em que foi realizada a paralelização.

Para efeito de comparação, as implementações de Borgelt dos algoritmos Apriori (BORGELT; KRUSE, 2002), Eclat (BORGELT, 2003) e FP-Growth (BORGELT, 2005) foram utilizadas. Estas são implementações otimizadas dos métodos originais, em que Borgelt empregou mecanismos tais como o uso de árvores de prefixos para representação dos candidatos e cálculo de suporte utilizando busca binária (BORGELT, 2003).

Em relação ao ambiente computacional dos experimentos, para as duas primeiras análises mencionadas, foi utilizado um servidor HPE ProLiant ML30 Gen9, equipado com um processador Intel Xeon E3-1220v6 de 3.00GHz e 8MB de cache, 8GB de memória RAM DDR4-2400MHz, placa de vídeo Nvidia Quadro P620 2GB GDDR5 128b e sistema operacional CentOS Linux 7. Todas as aplicações foram escritas em linguagem C/C++/OpenMP/Cuda, e a compilação foi realizada utilizando o GCC versão 4.8.5.

Já para a terceira análise, devido à necessidade de mais núcleos de processamento, foi utilizada a estrutura do NCC/GridUNESP que consiste de um cluster computacional com lâminas contendo núcleos Intel Xeon E5-2680v4 de 2.40GHz e 64GB de memória RAM. A compilação do Apriori-Roaring-Parallel e Apriori-Simd-Parallel foi realizada utilizando o Intel Compiler 2017.

5.2.1 Pico de uso de memória

Os resultados obtidos nesta análise são apresentados nas Figuras 20, 21, 22 e 23.

Na Figura 20 são exibidos os resultados comparativos do pico de uso de memória das implementações para conjuntos de dados sintéticos esparsos. Já na Figura 21 são apresentados os mesmos dados mas com o uso de memória das implementações de Borgelt para comparação.

As Figuras 22 e 23 exibem resultados similares, porém considerando a mineração de conjuntos de dados sintéticos densos.

Nesta análise utilizamos apenas conjuntos de dados sintéticos devido ao fato de poder estudar cenários de tamanhos (pequeno, médio, grande) e características (esparso, denso) diferentes. Somente utilizando os conjuntos de dados FIMI não conseguiríamos analisar todos estes cenários.

É possível observar através destes resultados que nem todas as implementações conseguem executar em todos os cenários, seja por conta de memória disponível insuficiente ou limite de tempo de execução. No caso das implementações de Borgelt, foi constatado que elas apresentam uma limitação em uso de memória para mineração de conjuntos de dados médios e grandes.

Considerando as implementações desenvolvidas, observa-se que o Apriori-Roaring-Parallel apresenta um limite de mineração de conjuntos de dados grandes esparsos superior às demais, devido ao seu baixo uso de memória. Já em questão de tempo de execução, esta implementação possui uma restrição em conjuntos de dados grandes densos, dada a maior demanda por tempo de processamento.

Para facilitar o entendimento dos gráficos, e também resumir os resultados, foi estabelecida a Tabela 6 que faz a recomendação ou não de determinada implementação para cada cenário.

Através da tabela comparativa, podemos confirmar algumas características a respeito do uso de memória das implementações desenvolvidas:

- **Apriori-Bitmap-Cuda:** para conjuntos de dados pequenos, utilizar a GPU implicar em demandar mais memória devido ao custo de utilizar a biblioteca Cuda. Para os conjuntos médios e grandes este custo é minimizado, e torna-se interessante a aplicação desta implementação
- **Apriori-Simd-Parallel:** foi observado que esta implementação apresenta uma demanda crescente de memória de acordo com o tamanho do conjunto de dados, com o pior caso para conjuntos grandes. Logo é recomendada sua utilização para conjuntos pequenos e médios
- **Apriori-Roaring-Parallel:** a utilização da compressão Roaring de bitmaps atribui à esta implementação a característica de ser a que menos demanda memória, sendo portanto recomendada em todos os cenários
- **Apriori-Cuda-Single-Memcpy:** esta implementação é apenas indicada para mineração de conjuntos esparsos de tamanho médio. Para minerar conjuntos pequenos, esta apresenta o custo de utilizar a biblioteca Cuda, o que a coloca em desvantagem frente às outras implementações. Já para conjuntos grandes e médios densos, devido à característica de a quantidade de memória da GPU ser o fator limitante, esta implementação não é recomendada

Um ponto importante a destacar é que apenas as implementações desenvolvidas foram capazes de executar a mineração dos conjuntos de dados grandes, reforçando assim o fato comentado de que o uso de memória está cada vez mais sendo o fator limitante neste tipo de mineração de dados.

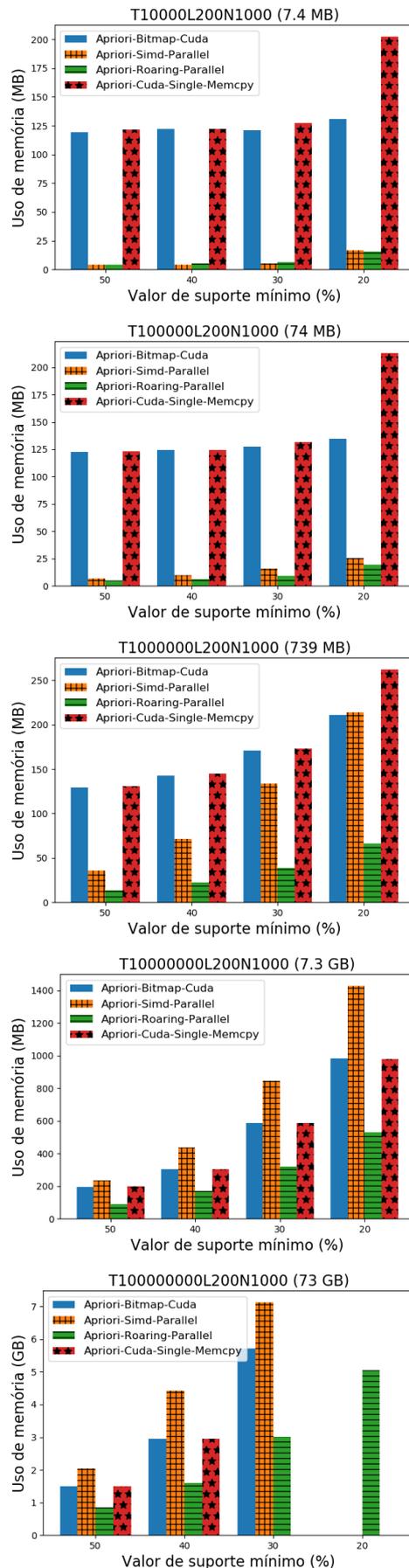


Figura 20 – Análise de pico de uso de memória das implementações do método de identificação de conjuntos frequentes para conjuntos de dados esparsos.

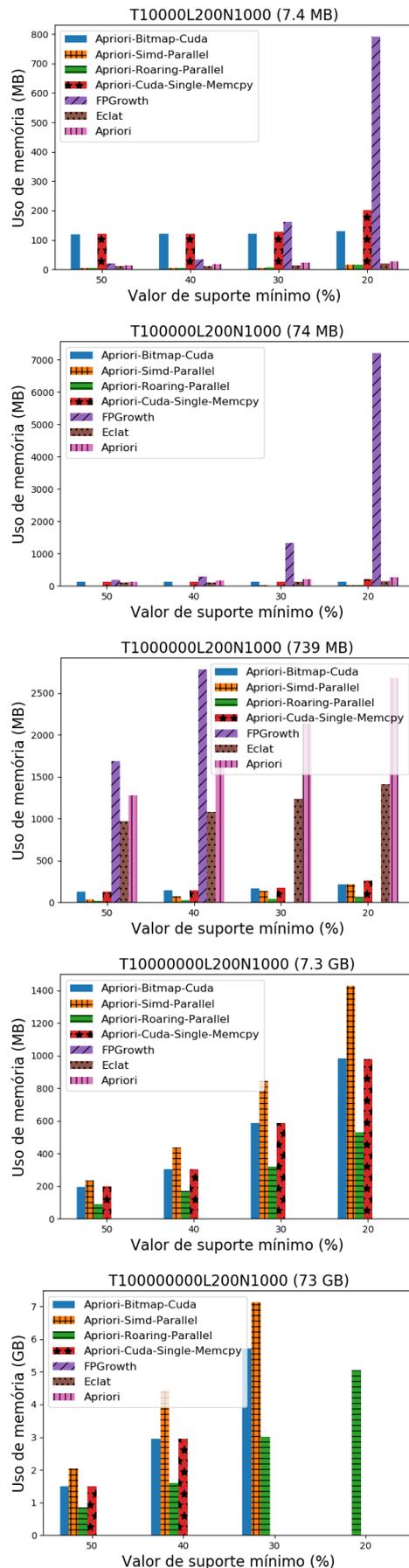


Figura 21 – Análise de pico de uso de memória das implementações do método de identificação de conjuntos frequentes com o comparativo das implementações de Borgelt para conjuntos de dados esparsos.

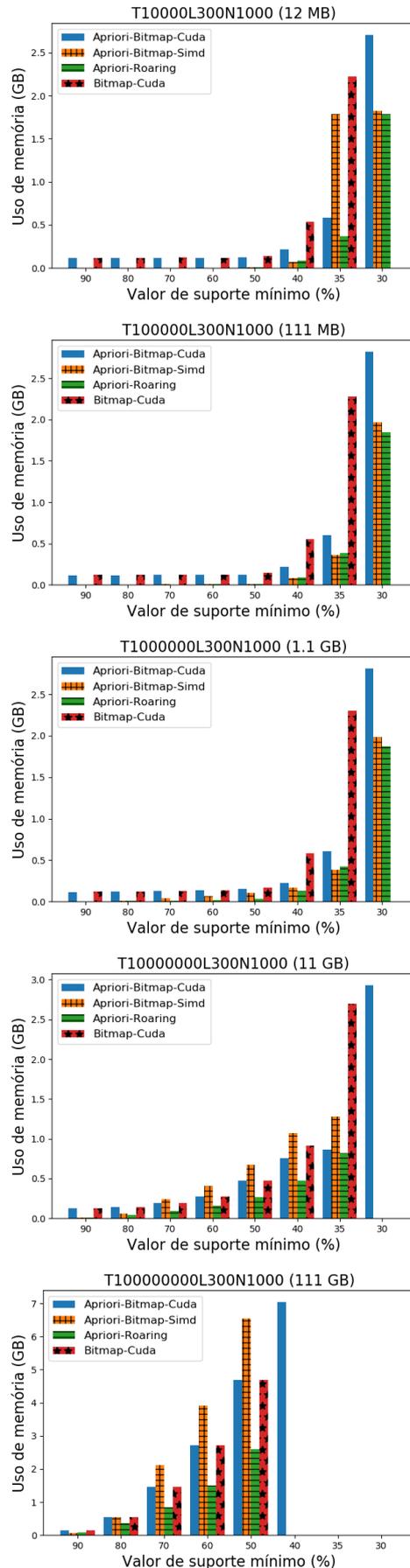


Figura 22 – Análise de pico de uso de memória das implementações do método de identificação de conjuntos frequentes para conjuntos de dados densos.

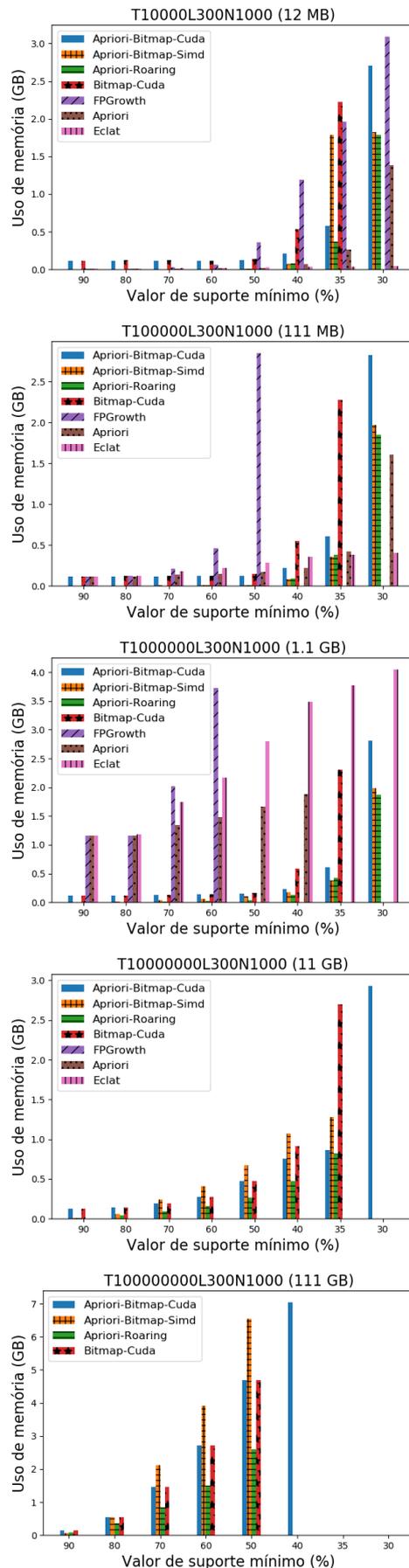


Figura 23 – Análise de pico de uso de memória das implementações do método de identificação de conjuntos frequentes com o comparativo das implementações de Borgelt para conjuntos de dados densos.

Tabela 6 – Cenários de indicação de uso das implementações de identificação de conjuntos frequentes de acordo com uso de memória.

	Apriori Bitmap Cuda	Apriori Simd Paralel	Apriori Roaring Paralel	Apriori Cuda Single Memcpy	FP-Growth	Apriori	Eclat
Pequeno esperso (1 MB~1 GB)	Não indicado	Indicado	Indicado	Não indicado	Indicado	Indicado	Indicado
Pequeno denso (1 MB~1 GB)	Não indicado	Indicado	Indicado	Não indicado	Indicado	Indicado	Indicado
Médio esperso (1 GB~10 GB)	Indicado	Indicado	Indicado	Indicado	Não indicado	Não indicado	Não indicado
Médio denso (1 GB~10 GB)	Indicado	Indicado	Indicado	Não indicado	Não indicado	Não indicado	Não indicado
Grande esperso (>10 GB)	Indicado	Não indicado	Indicado	Não indicado	Não indicado	Não indicado	Não indicado
Grande denso (>10 GB)	Indicado	Não indicado	Indicado	Não indicado	Não indicado	Não indicado	Não indicado

5.2.2 Tempo de execução

Os resultados desta análise foram obtidos através das mesmas execuções da análise anterior, porém com foco no tempo de execução. Estes resultados são apresentados nas Figuras 24, 25, 26 e 27.

De forma semelhante, os resultados dos gráficos foram resumidos na Tabela 7 para recomendar ou não determinada implementação para cada cenário.

Através da tabela comparativa, podemos confirmar algumas características a respeito do tempo de execução das implementações desenvolvidas:

- **Apriori-Bitmap-Cuda:** devido ao custo de constantes transferências de memória (linhas bitmap) entre *host* e GPU, esta implementação é somente indicada para conjuntos de dados grandes, onde destaca-se o tempo de cálculo do valor de suporte
- **Apriori-Simd-Paralel:** esta implementação explora os recursos do processador do *host* (vetorização e paralelização em memória compartilhada) sem apresentar um custo de transferência de memória, e portanto é recomendada em todos os cenários
- **Apriori-Roaring-Paralel:** devido à utilização da compressão Roaring de bitmaps, esta implementação apresenta um penalidade no tempo de execução, apresentando o pior resultado frente às outras implementações
- **Apriori-Cuda-Single-Memcpy:** esta implementação, semelhante à Apriori-Bitmap-Cuda, realiza o cálculo do valor de suporte em GPU, e adicionalmente não possui o custo de transferências constantes de memória, sendo portanto recomendada para quase todos os cenários

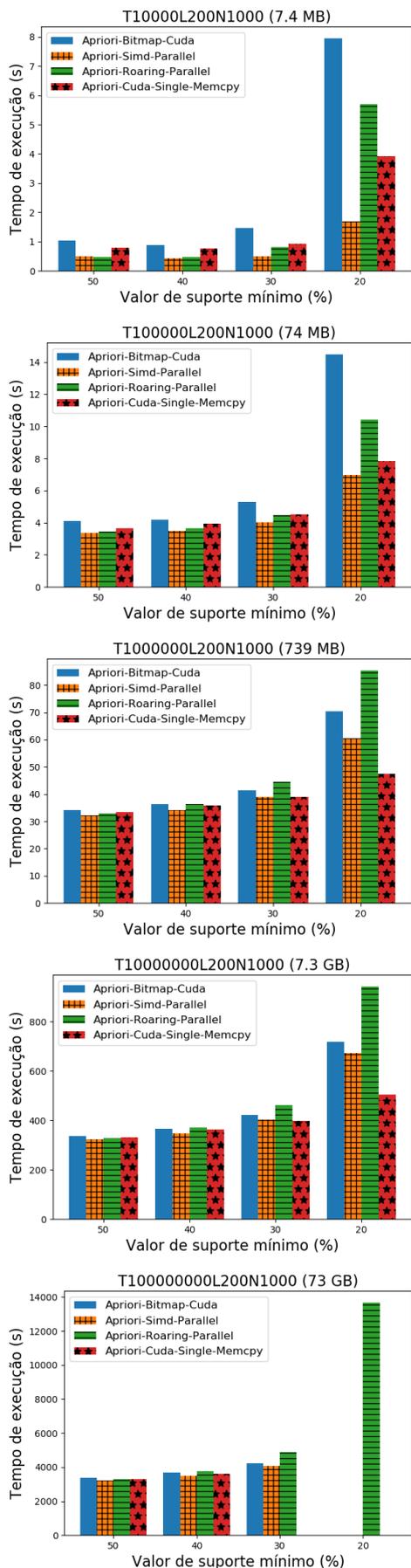


Figura 24 – Análise de tempo de execução das implementações do método de identificação de conjuntos frequentes para conjuntos de dados esparsos.

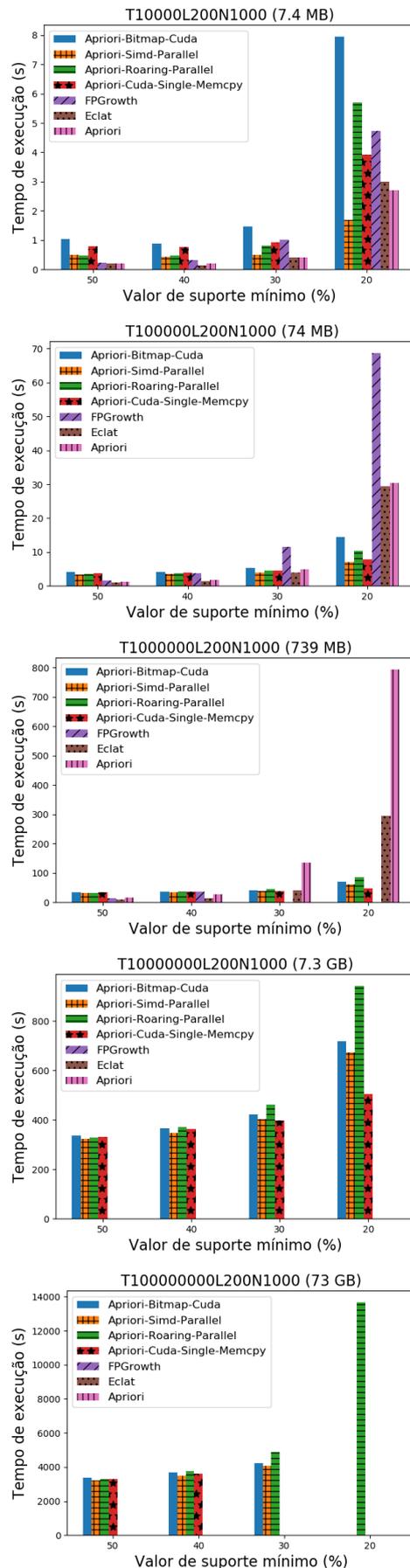


Figura 25 – Análise de tempo de execução das implementações do método de identificação de conjuntos frequentes com o comparativo das implementações de Borgelt para conjuntos de dados esparsos.

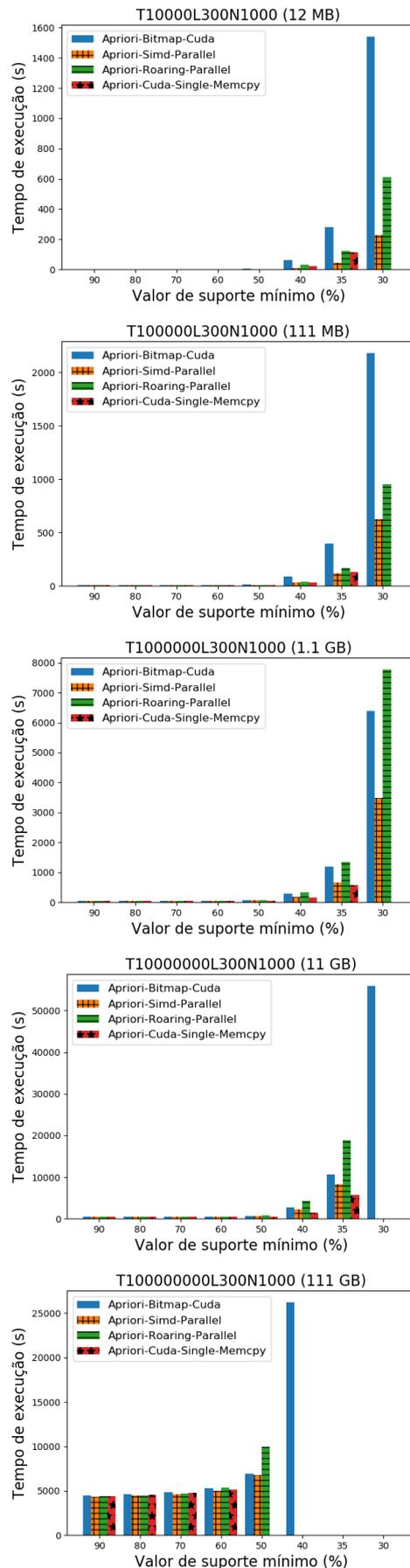


Figura 26 – Análise de tempo de execução das implementações do método de identificação de conjuntos frequentes para conjuntos de dados densos.

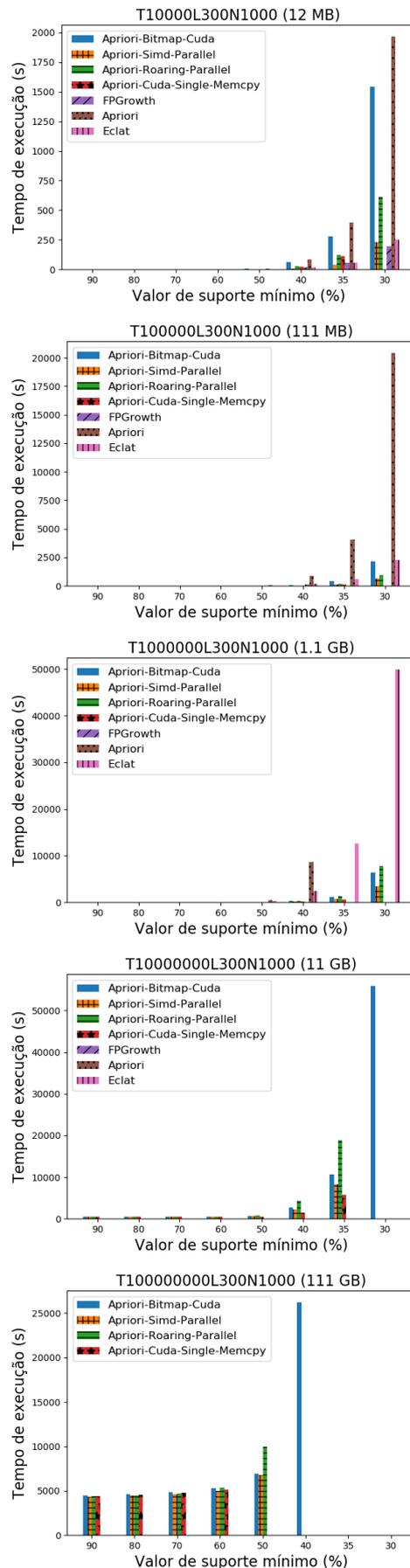


Figura 27 – Análise de tempo de execução das implementações do método de identificação de conjuntos frequentes com o comparativo das implementações de Borgelt para conjuntos de dados densos.

Tabela 7 – Cenários de indicação de uso das implementações de identificação de conjuntos frequentes de acordo com tempo de execução.

	Apriori Bitmap Cuda	Apriori Simd Parallel	Apriori Roaring Parallel	Apriori Cuda Single Memcpy	FP-Growth	Apriori	Eclat
Pequeno esparsos (1 MB~1 GB)	Não indicado	Indicado	Não indicado	Não indicado	Indicado	Não indicado	Indicado
Pequeno denso (1 MB~1 GB)	Não indicado	Indicado	Indicado	Indicado	Indicado	Não indicado	Indicado
Médio esparsos (1 GB~10 GB)	Não indicado	Indicado	Não indicado	Indicado	Não indicado	Não indicado	Não indicado
Médio denso (1 GB~10 GB)	Não indicado	Indicado	Não indicado	Indicado	Não indicado	Não indicado	Não indicado
Grande esparsos (>10 GB)	Indicado	Indicado	Não indicado	Não indicado	Não indicado	Não indicado	Não indicado
Grande denso (>10 GB)	Indicado	Indicado	Não indicado	Indicado	Não indicado	Não indicado	Não indicado

5.2.3 Escalabilidade

Neste experimento foi realizada uma análise do *speedup* das implementações Apriori-Roaring-Parallel e Apriori-Simd-Parallel em razão do número de elementos de processamento.

Como cenário do experimento, foi utilizado o conjunto de dados sintético esparsos médio (T1000000L200N1000) com um valor de suporte mínimo definido para 10 %. Estas definições dão destaque no tempo de execução para os trechos de cálculo do valor de suporte dos conjuntos e geração de candidatos.

Os resultados são expostos nas Figuras 28 e 29, com o valor médio de dez execuções para cada valor de quantidade de *threads*.

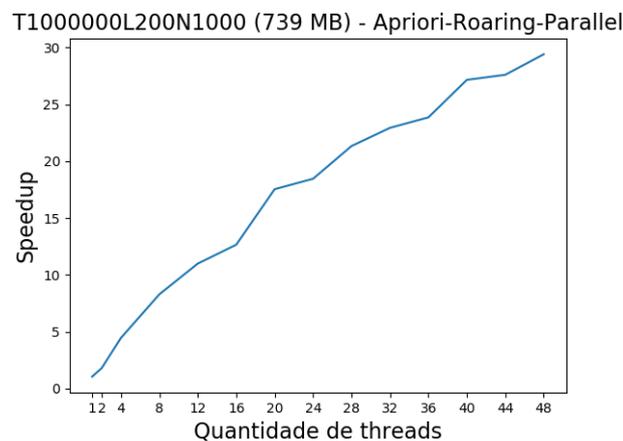


Figura 28 – Análise de escalabilidade da implementação Apriori-Roaring-Parallel.

Pelos resultados obtidos, percebe-se que a implementação do Apriori-Roaring-Parallel é escalável em relação à quantidade de elementos de processamento em uso, o que a torna adequada às situações em que se deseja acelerar a mineração de dados através do incremento de *hardware*.

A implementação do Apriori-Simd-Parallel, por outro lado, se mostrou escalável até um valor de cerca de 16 *threads*, sendo que após isto não foram obtidos ganhos com

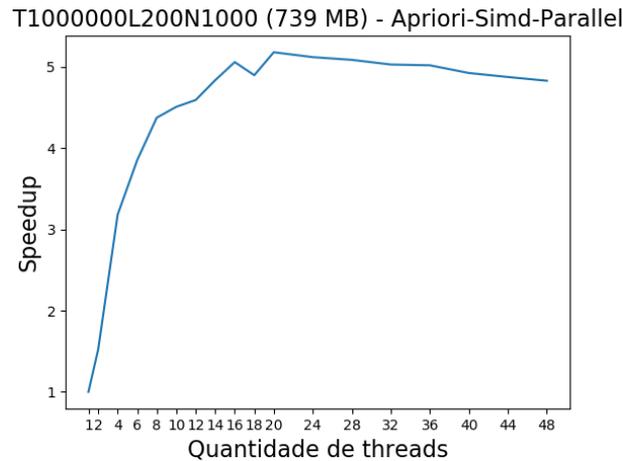


Figura 29 – Análise de escalabilidade da implementação Apriori-Simd-Parallel.

o incremento da quantidade de *threads*. Por se tratar de uma implementação que explora o paralelismo em memória compartilhada, entende-se que o limite de *threads* para escalabilidade é compatível com o disponível atualmente pelos diversos fabricantes de processadores.

Além disso, foi observada uma flutuação dos valores obtidos de *speedup* em torno do valor limite a partir de 16 *threads*. Este fato se deve ao ambiente computacional utilizado, em que existe uma independência de *threads* alocadas por processo, mas um compartilhamento dos demais recursos, como disco e memória principal.

Um ponto a comentar sobre este experimento é que seu enfoque é em um cenário que destaca o cálculo de suporte de conjuntos. Para outros cenários, como por exemplo o de um conjunto de dados grande, outros trechos se destacam, como o que realiza as leituras do conjunto de dados original. No caso do exemplo hipotético, o *speedup* observado será diferente, e dependente de outros fatores como, por exemplo, tecnologias de recuperação de dados em disco e de armazenamento (disco rígido vs. estado sólido).

5.3 Mineração de regras negativas

Para verificar o desempenho do método de mineração de regras negativas desenvolvido, foram realizados experimentos em quatro análises. A primeira delas avalia o pico de uso de memória das implementações, identificando os limites de aplicação do método.

Em uma segunda análise foram realizados experimentos com os mesmos cenários desta primeira, porém foram comparados os tempos de execução. Nesta são consideradas cinco execuções de cada configuração, expondo nos gráficos os valores médios.

Uma terceira análise buscou identificar a quantidade de regras de associação geradas, discriminando a contribuição de cada uma das quatro partes do método.

Por fim, a quarta análise buscou identificar a escalabilidade do método ao variar a quantidade de *threads* para execução.

Durante estas análises, foram utilizados valores de suporte mínimo e confiança mínima que, ou eram limites para a mineração de cada conjunto de dados para o ambiente computacional utilizado, ou proporcionavam destaque para os trechos em que foi realizada a paralelização.

As três primeiras análises mencionadas foram realizadas em um servidor HPE ProLiant ML30 Gen9, equipado com um processador Intel Xeon E3-1220v6 de 3.00GHz e 8MB de cache, 8GB de memória RAM DDR4-2400MHz, placa de vídeo Nvidia Quadro P620 2GB GDDR5 128b e sistema operacional CentOS Linux 7. O método foi escrito em linguagem C/C++/OpenMP/Cuda, e a compilação foi realizada utilizando o GCC versão 4.8.5.

Semelhante aos demais experimentos, devido à necessidade de mais núcleos de processamento, para a quarta análise foi utilizada a estrutura do NCC/GridUNESP que consiste de um cluster computacional com lâminas contendo núcleos Intel Xeon E5-2680v4 de 2.40GHz e 64GB de memória RAM. A compilação neste ambiente foi realizada utilizando o Intel Compiler 2017.

Neste experimento foram utilizados apenas conjuntos de dados esparsos, tanto do repositório FIMI quanto sintéticos. O uso de conjuntos de dados densos demandou mais memória que o disponível nos ambientes de experimentos, o que indica uma restrição do método desenvolvido.

É importante destacar que, por definição do método PNAR, a geração de regras negativas é dependente da quantidade de regras positivas identificadas, e além disso somente são geradas regras negativas ao selecionar valores de suporte mínimo inferiores à 50 %.

Para estes experimentos, os valores selecionados de suporte mínimo e confiança mínima foram aqueles que representam regiões de interesse para cada conjunto de dados.

5.3.1 Pico de uso de memória

Os resultados obtidos nesta análise são apresentados nas Figuras 30 e 31.

Na Figura 30 são exibidos os resultados comparativos para conjuntos de dados do repositório FIMI, enquanto na Figura 31 são apresentados os resultados para os conjuntos de dados sintéticos.

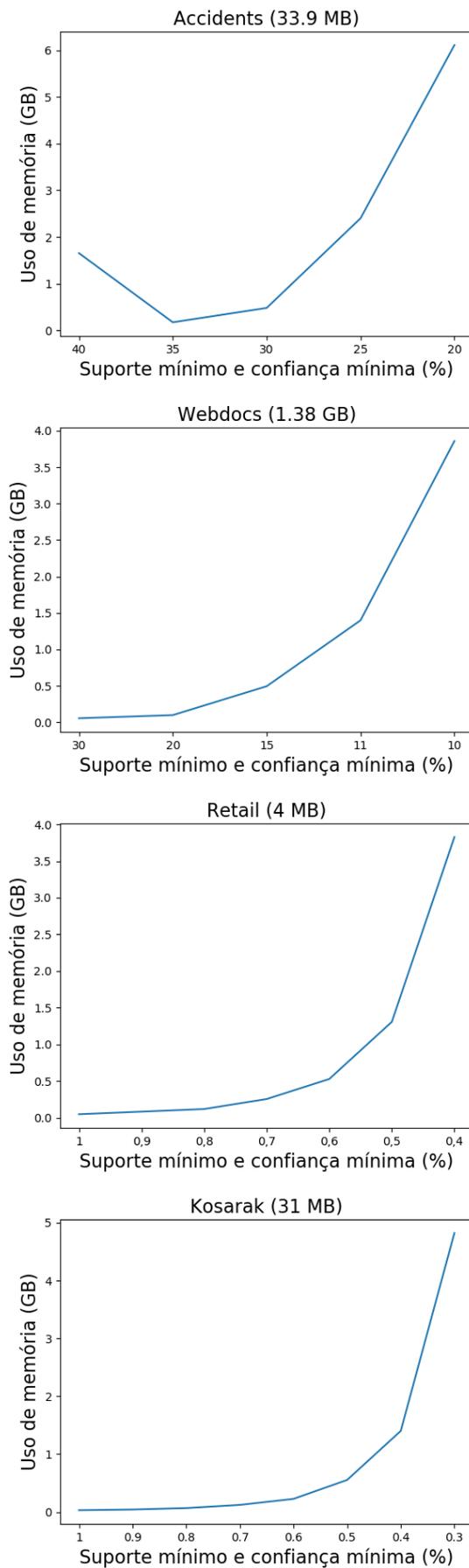


Figura 30 – Análise de pico de uso de memória do método desenvolvido de identificação de regras de associação negativas utilizando conjuntos de dados do repositório FIMI.

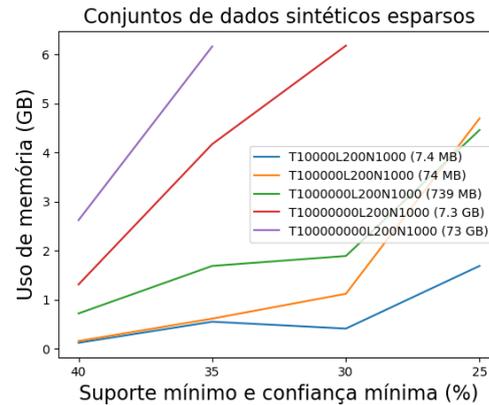


Figura 31 – Análise de pico de uso de memória do método desenvolvido de identificação de regras de associação negativas utilizando conjuntos de dados sintéticos.

Através dos gráficos é possível identificar as seguintes características do método a respeito do uso de memória:

- O uso de memória apresenta um comportamento exponencial em relação aos valores de suporte mínimo e confiança mínima
- Dentre as diversas etapas do método desenvolvido, a maior contribuição para o uso de memória ao selecionar valores baixos de suporte mínimo e confiança mínima se deve à quarta parte do algoritmo PNAR
- Para valores de suporte mínimo próximos à 50 %, há um aumento na quantidade de uso de memória devido ao processo de mineração da terceira parte do algoritmo PNAR, o que é indicado pela análise do conjunto de dados accidents e também apresentado na Figura 32
- O método consegue minerar conjuntos de dados grandes, porém com um valor de suporte mínimo limite, como pode ser visto no resultado dos conjuntos de dados sintéticos

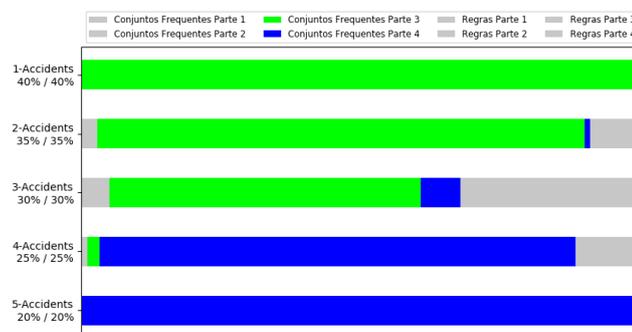


Figura 32 – Análise de participação das etapas três e quatro do método desenvolvido na composição do tempo de execução, considerando o conjunto de dados accidents e valores de suporte mínimo e confiança mínima variando de 20% à 40%.

5.3.2 Tempo de execução

Os resultados desta análise foram obtidos através das mesmas execuções da análise anterior, porém com foco no tempo de execução. Estes resultados são apresentados nas Figuras 34 e 33.

Através dos gráficos é possível identificar as seguintes características do método a respeito do tempo de execução:

- Similar ao uso de memória, a característica do tempo de execução também apresenta um comportamento exponencial
- A maior contribuição para o tempo de execução ao selecionar valores baixos de suporte mínimo e confiança mínima se deve à quarta parte do método desenvolvido
- O método consegue minerar conjuntos de dados grandes, porém com um valor de suporte mínimo limite, como pode ser visto no resultado dos conjuntos de dados sintéticos

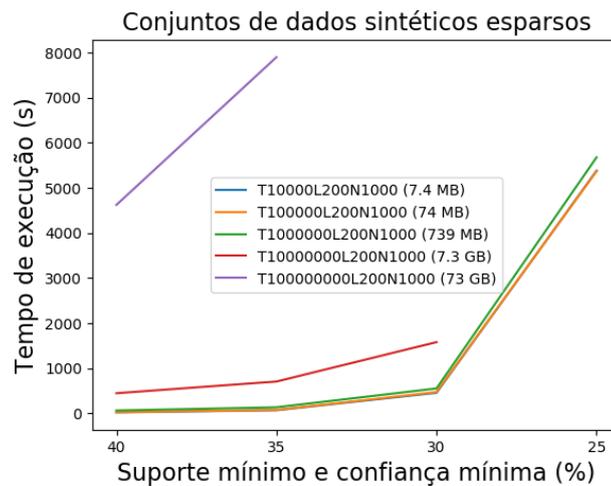


Figura 33 – Análise de tempo de execução do método desenvolvido de identificação de regras de associação negativas utilizando conjuntos de dados sintéticos.

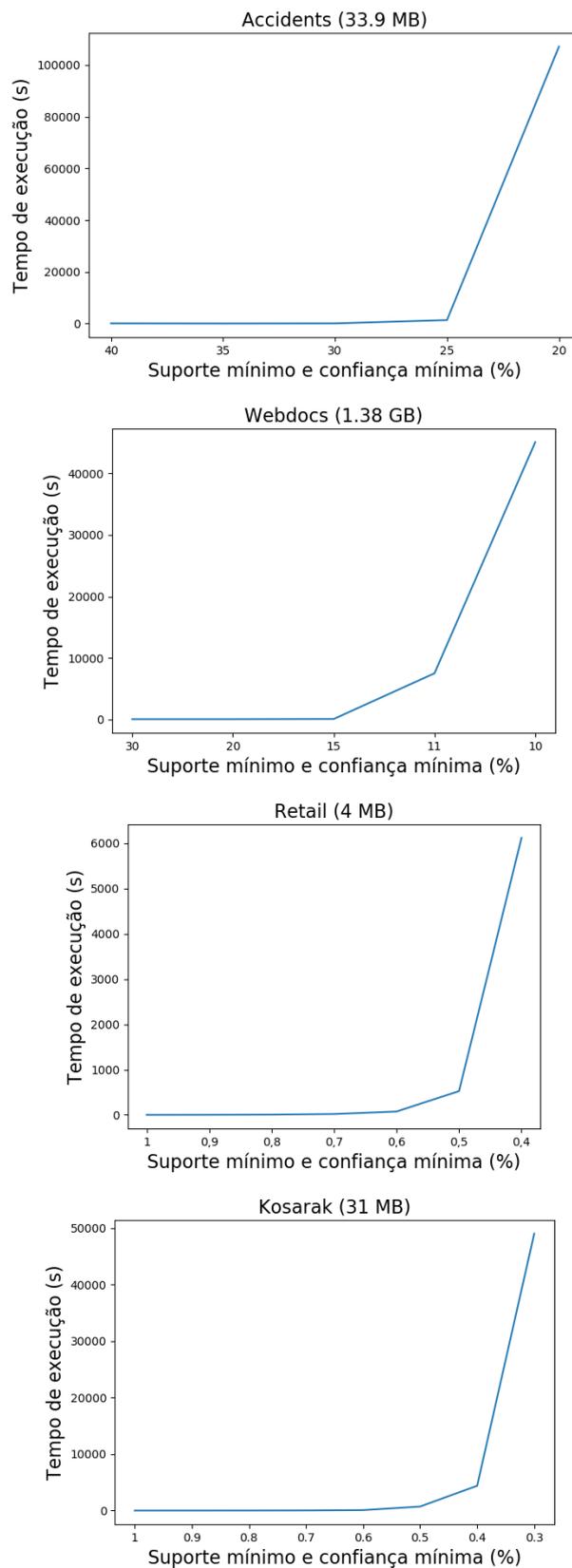


Figura 34 – Análise de tempo de execução do método desenvolvido de identificação de regras de associação negativas utilizando conjuntos de dados do repositório FIMI.

5.3.3 Regras geradas

Esta análise buscou identificar a relação entre regras positivas e negativas geradas, identificando a contribuição de cada parte do método neste processo. Os resultados apresentando apenas a discriminação entre regras positivas e negativas são expostos nas Figuras 35 e 37. Já nas Figuras 36 e 38 são expostos os resultados da contribuição de cada parte do método.

Nesta análise foi selecionado um valor de interesse de suporte mínimo, e feita a variação do valor de confiança mínima para alterar a quantidade de regras geradas.

Através dos gráficos é possível identificar as seguintes características do método a respeito da geração de regras:

- De forma geral, a quantidade de regras negativas geradas é muito maior que a quantidade de regras positivas
- Por conta do método PNAR gerar regras negativas a partir de positivas, existe um limite de geração de regras negativas ao selecionar valores pequenos de confiança mínima
- A contribuição de regras pelas partes do método depende da característica do conjuntos de dados, não existindo uma predominância constante de uma parte sobre outra
- Com a diminuição do valor de confiança mínima, a quantidade de regras negativas geradas pela parte 2 tende a se aproximar da quantidade de regras positivas geradas
- Com a diminuição do valor de confiança mínima, a quantidade de regras negativas geradas tende a crescer de forma mais acelerada que a quantidade de regras positivas geradas

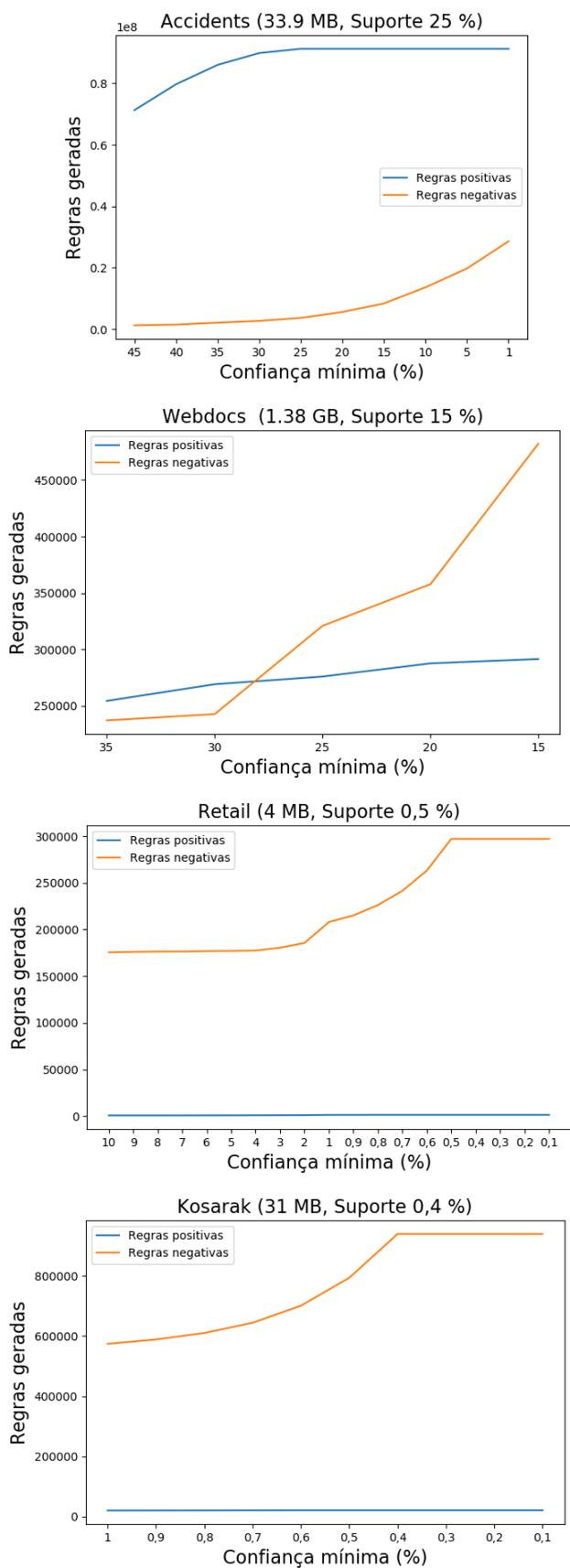


Figura 35 – Análise de quantidade de regras geradas pelo método desenvolvido utilizando conjuntos de dados do repositório FIMI, representando valores totais.

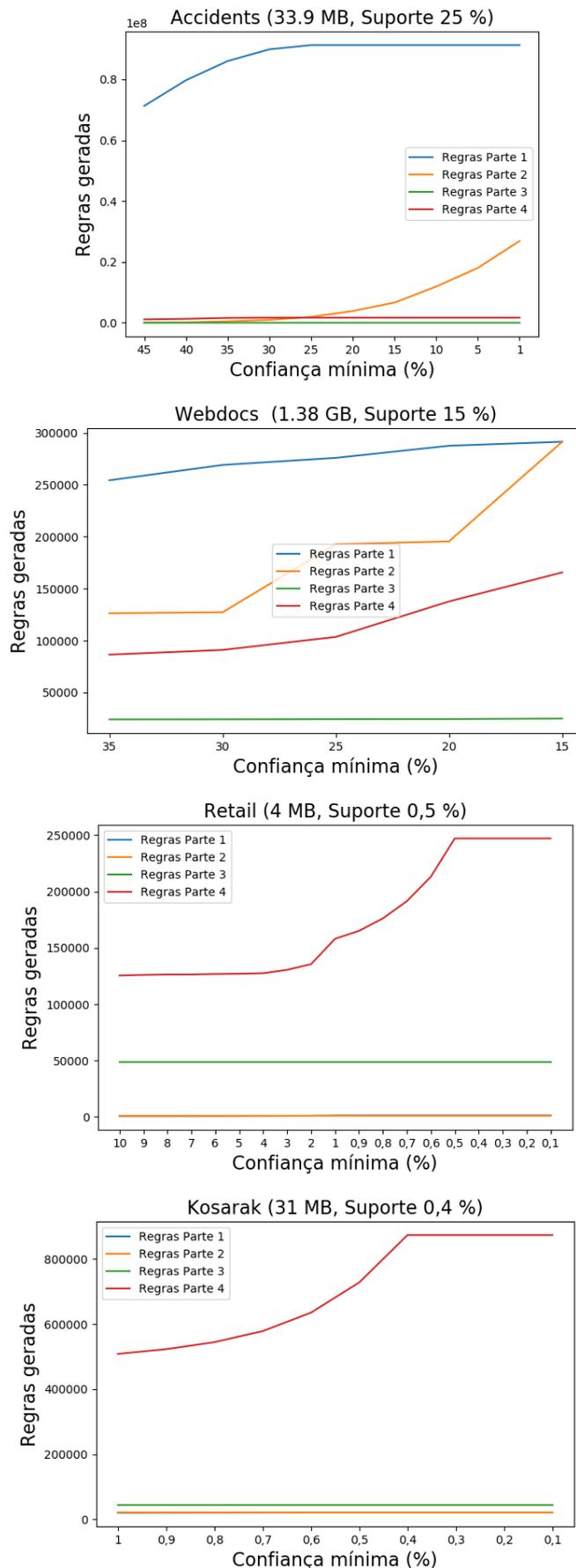


Figura 36 – Análise de quantidade de regras geradas pelo método desenvolvido utilizando conjuntos de dados do repositório FIMI, representando contribuição de partes do método.

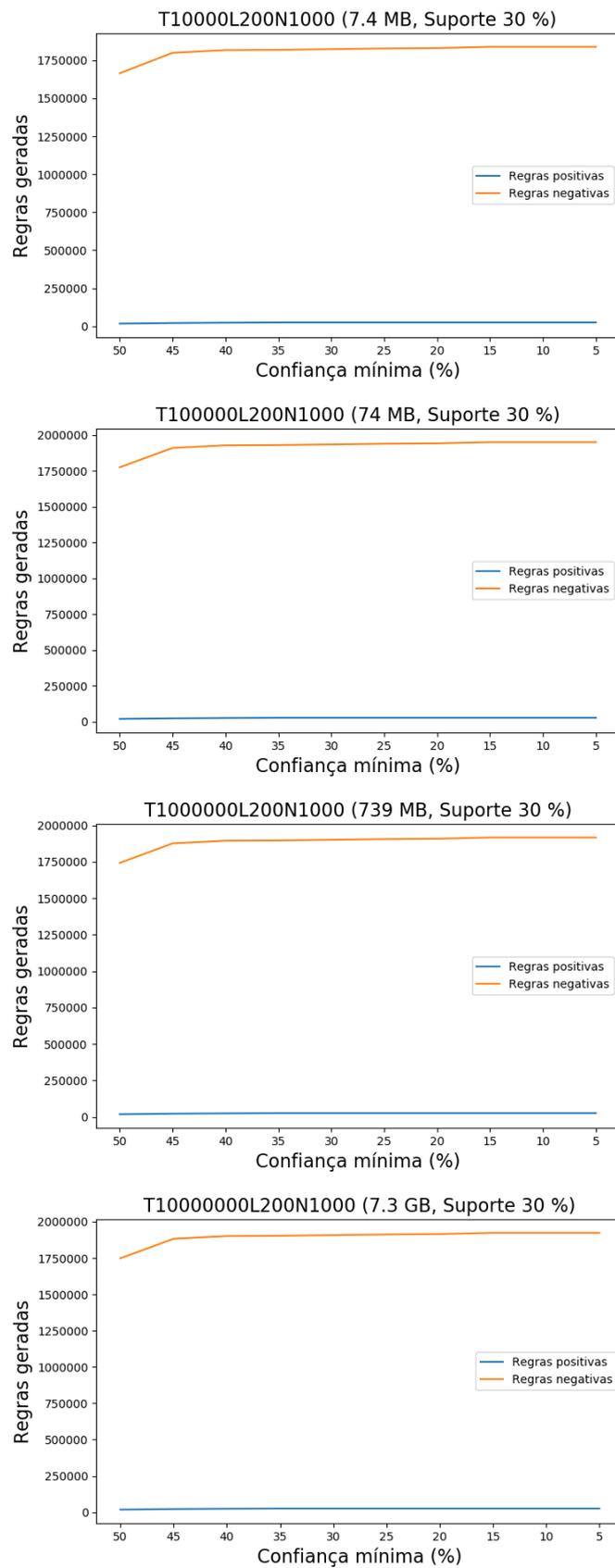


Figura 37 – Análise de quantidade de regras geradas pelo método desenvolvido utilizando conjuntos de dados sintéticos, representando valores totais.

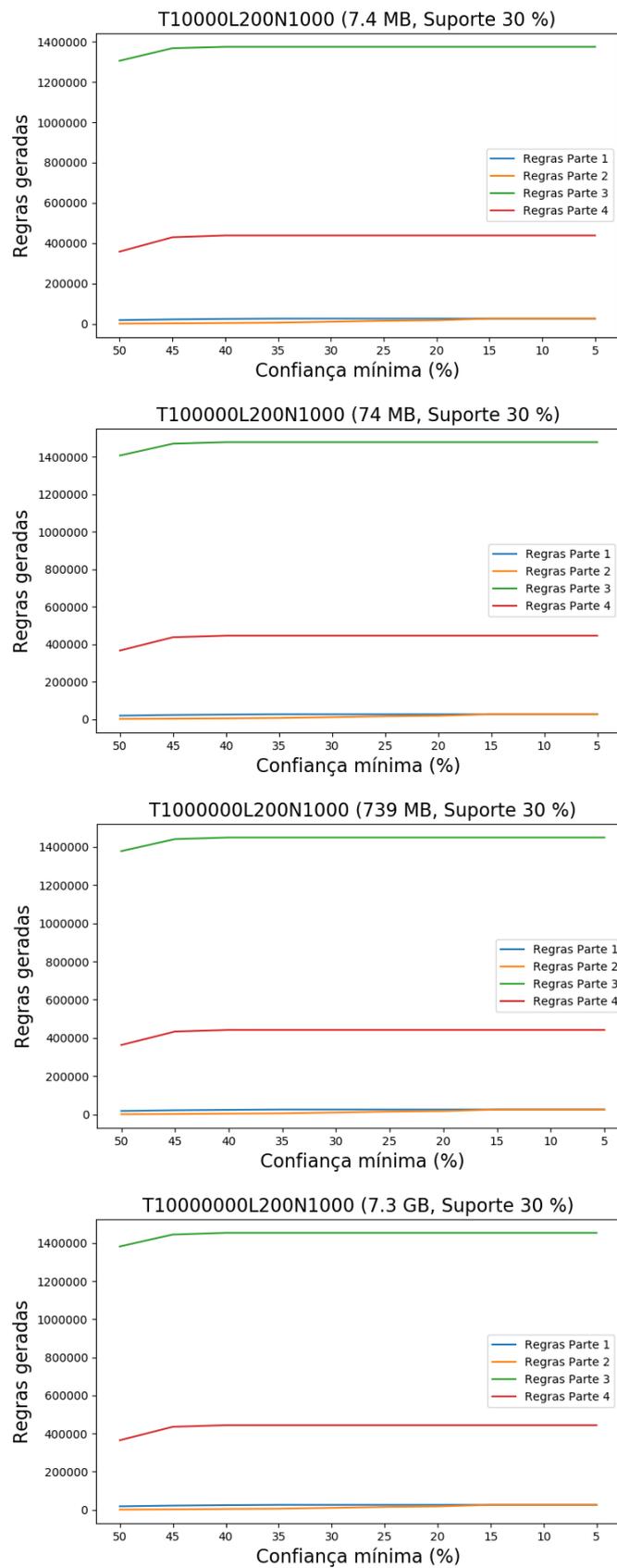


Figura 38 – Análise de quantidade de regras geradas pelo método desenvolvido utilizando conjuntos de dados sintéticos, representando contribuição de partes do método.

5.3.4 Escalabilidade

O objetivo desta última análise é compreender a característica de escalabilidade do método desenvolvido. Para isto, foram realizadas execuções considerando valores de suporte mínimo e confiança mínima nos quais as partes paralelizadas apresentaram destaque na composição do tempo de execução. Entre estas execuções foi alterada a quantidade de *threads* para comparação do tempo de execução.

Na Figura 39 são apresentados os resultados das execuções com os conjuntos de dados do repositório FIMI, enquanto as Figuras 40 e 41 contemplam os resultados dos conjuntos de dados sintéticos.

Através dos gráficos é possível identificar as seguintes características do método a respeito da escalabilidade:

- Para a quantidade de *threads* utilizada no experimento o método se mostrou escalável
- Conjuntos de dados com uma característica de esparsidade mais acentuada apresentaram um *speedup* superior
- Para conjuntos de dados maiores, é observado um *speedup* menor devido ao ganho de relevância na composição do tempo de execução de outras etapas do método, como por exemplo a leitura em disco do conjunto de dados
- Durante os experimentos foram utilizados discos rígidos, o que impacta consideravelmente o *speedup*. Caso fossem utilizados dispositivos de armazenamento em estado sólido (SSD), seria esperado que esta característica fosse aprimorada
- Embora o método não tenha sido paralelizado em todos os trechos, a escalabilidade obtida indica que foram paralelizados os trechos de relevância para o tempo de execução
- O método desenvolvido é adequado às situações em que se deseja acelerar a mineração de dados através do incremento de *hardware*

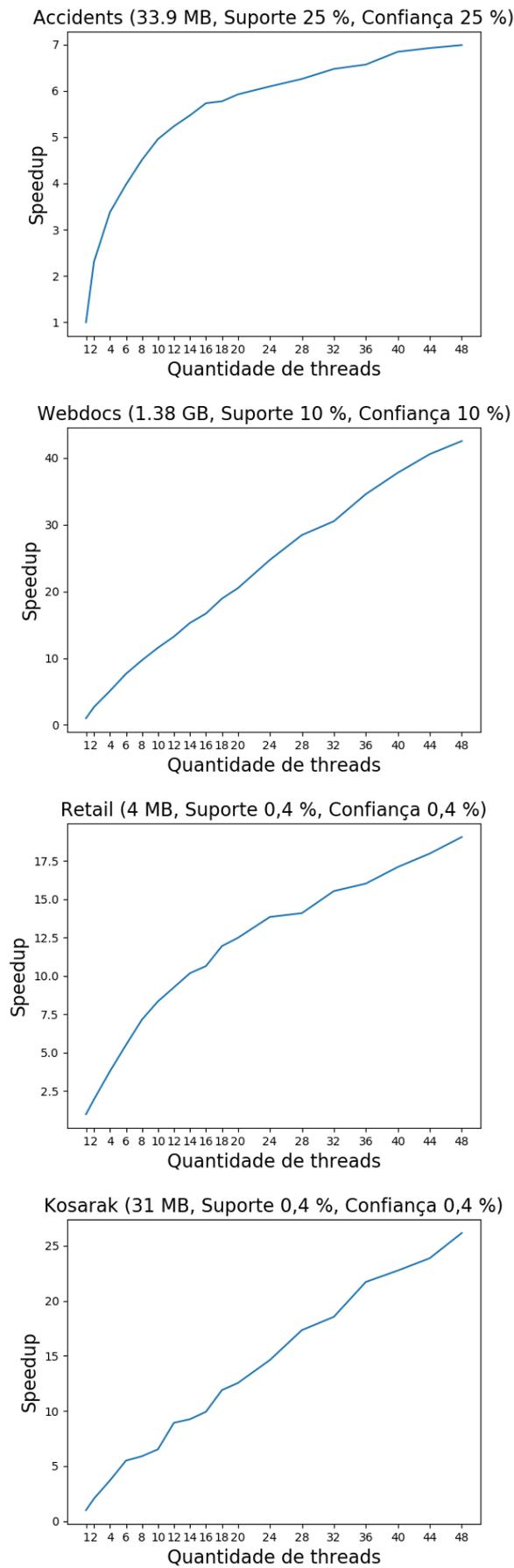


Figura 39 – Análise de escalabilidade do método desenvolvido utilizando conjuntos de dados do repositório FIMI.

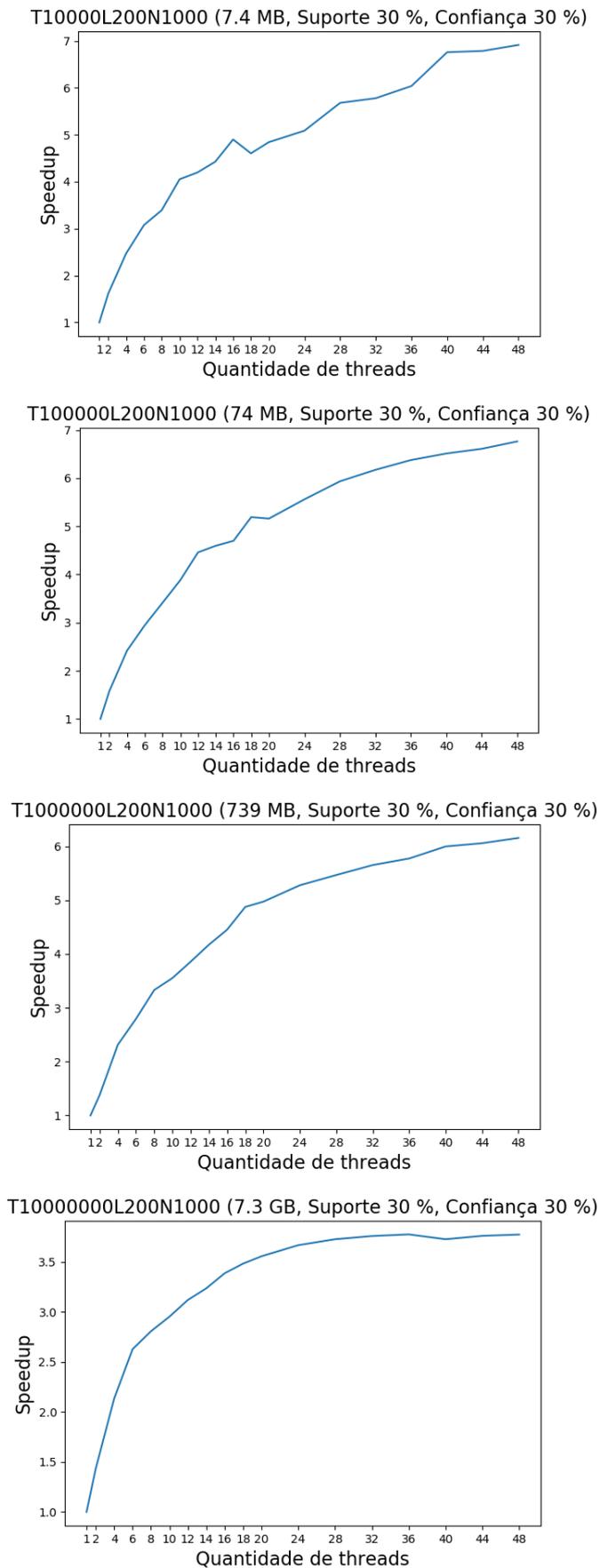


Figura 40 – Análise de escalabilidade do método desenvolvido utilizando conjuntos de dados sintéticos, com representação discriminada.

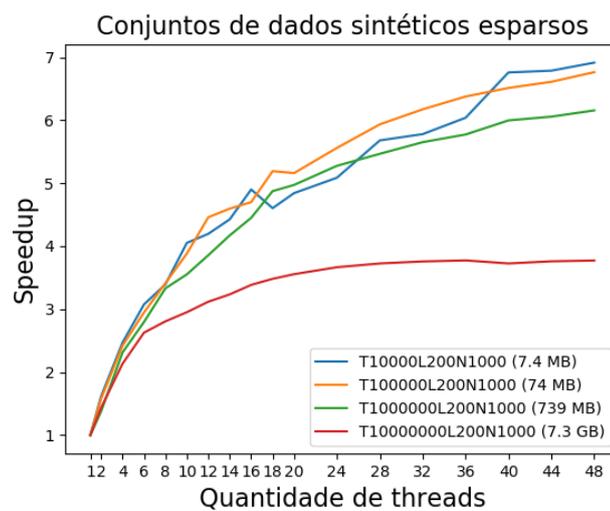


Figura 41 – Análise de escalabilidade do método desenvolvido utilizando conjuntos de dados sintéticos, com representação agrupada.

6 Conclusões e trabalhos futuros

Identificação de padrões frequentes e regras de associação se apresenta como uma tarefa de mineração de dados com o objetivo de determinar relações consistentes entre elementos. As principais abordagens existentes se baseiam na ocorrência de elementos, que é um cenário em que para certas condições de tamanho de conjunto de dados e valor de suporte mínimo já se apresenta como um problema de difícil resolução.

Pela análise destas abordagens, foi possível identificar que algumas delas são passíveis de serem paralelizadas, o que permite executar esta tarefa em menor tempo ou que condições mais críticas possam ser analisadas. A paralelização, entretanto, exige que os dados sejam armazenados em representações uniformes para evitar cenários de divergência de elementos de processamento. De forma a atender este requisito, foram estudadas as representações bitmap e suas variações contemplando compressão, com destaque para o método de compressão Roaring.

Deste estudo, foi obtido um método contemplando quatro implementações que fazem uso de paralelismo em memória compartilhada, vetorização e computação heterogênea. Este possui como principal proposta realizar a identificação de conjuntos frequentes adotando a melhor estratégia para cada cenário, que irá depender da característica do conjuntos de dados a ser analisado e do valor de suporte mínimo selecionado.

O método permite trabalhar com situações em que deseja-se acelerar o processo de mineração, com o uso de GPU ou vetorização e paralelismo em memória compartilhada, ou situações em que deseja-se analisar conjuntos de dados muito grandes, que é realizado com o emprego de estruturas bitmap comprimidas.

Posteriormente à tarefa de identificar conjuntos frequentes, deverão ser geradas regras de associação, que é de fato a maneira de extrair conhecimento de conjuntos de dados. Abordagens tradicionais consideram apenas a ocorrência de elementos, o que foi expandido ao explorar o problema englobando também a ausência destes, com a geração de regras negativas. Ao considerar este tipo de regras, poderão ser identificadas associações de grande interesse e conhecimento até então desconhecido para o cientista de dados ou analista. Este tipo de tarefa, entretanto, tem como principal característica a explosão da quantidade de regras geradas, que por meio de investigação científica foi possível levantar diversas abordagens para seu tratamento.

Como resultado desta investigação, foi desenvolvido um método baseado na abordagem PNAR de geração de regras negativas, que é exato e emprega mecanismo de poda para reduzir a quantidade de regras geradas. O intuito do mecanismo de poda é tanto acelerar o processo de mineração quanto remover regras redundantes e de pouco interesse.

O método desenvolvido utiliza a etapa de identificação de conjuntos frequentes desenvolvida inicialmente, somado à uma implementação paralela do método PNAR. Nesta implementação, os trechos que apresentaram destaque na composição do tempo de execução em análises sequenciais foram acelerados com paralelismo, conforme constatado através de experimentos.

Por fim, é importante destacar que o uso de plataformas heterogêneas e métodos paralelos não apresenta o objetivo único de executar a tarefa em menor tempo, mas sim uma preocupação em permitir que problemas em cenários mais críticos possam ser analisados. Um destes cenários em questão é o caso de regras negativas, que foi o desafio principal do presente trabalho.

6.1 Contribuições

Este trabalho teve início com a elaboração de uma revisão da literatura, de forma a selecionar os trabalhos recentes acerca do tema de pesquisa.

Posteriormente foram analisadas representações de conjuntos de dados alternativas à tradicional representação por transações. Nesta análise foi adotado um enfoque em selecionar representações que fossem adequadas ao desenvolvimento de um método que explore o paralelismo, e também em representações que permitam utilizar menos memória durante o processo de mineração.

Considerando a primeira etapa do processo de mineração, que é a identificação de conjuntos frequentes, foram desenvolvidas quatro abordagens que se destacam em diferentes cenários. Estas abordagens se limitaram à explorar o paralelismo em contextos de memória compartilhada e computação heterogênea.

Para a segunda etapa do processo de mineração, que é a geração de regras negativas, foi realizada a implementação de um método paralelo. Este método é baseado em um estudo que reduz o espaço de busca de regras através do emprego de mecanismos de poda.

De forma geral, a contribuição deste trabalho foi a proposta de um novo método de geração de regras negativas que explora o paralelismo e é escalável.

6.2 Trabalhos futuros

Neste trabalho foi definido um foco em acelerar o processo de mineração de regras negativas explorando o paralelismo em memória compartilhada e plataformas heterogêneas. Através dos resultados foi observado que o método implementado é escalável. Deste fato, entende-se portanto, que um possível trabalho futuro seria realizar a implementação

deste método em uma arquitetura distribuída, com foco em paralelismo de dados, o que irá permitir trabalhar com uma quantidade de nós de processamento superior aos que foram realizados os experimentos.

Um outro desdobramento deste trabalho de pesquisa seria empregar compressão de representações bitmap em GPU. Com o uso de uma compressão deste tipo, conjuntos de dados maiores poderiam ser minerados utilizando uma abordagem como a Apriori Cuda Single Memcpy, em que o fator limitante é o tamanho da memória da GPU.

Por fim, durante a etapa de geração de regras negativas foi selecionado como base um estudo que emprega mecanismos de poda. Seria interessante alterar esta etapa para a adoção de outras abordagens, de forma a verificar a variação no desempenho e na quantidade de regras geradas.

Referências

- AGGARWAL, C. C.; YU, P. S. A New Framework For Itemset Generation. In: *Proceedings of the 17th Symposium on Principles of Database Systems*. [S.l.]: ACM Press, 1998. p. 18–24. ISBN 0-89791-996-3. Citado na página 18.
- AGGARWAL, C. C.; YU, P. S. Mining associations with the collective strength approach. *IEEE Transactions on Knowledge and Data Engineering*, v. 13, n. 6, p. 863–873, Nov 2001. ISSN 2326-3865. Citado 2 vezes nas páginas 13 e 28.
- AGRAWAL, J. et al. Set-pso-based approach for mining positive and negative association rules. *Knowledge and Information Systems*, v. 45, p. 453–471, 2014. Citado na página 14.
- AGRAWAL, R.; SHAFER, J. C. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, v. 8, n. 6, p. 962–969, Dec 1996. ISSN 2326-3865. Citado na página 14.
- AGRAWAL, R.; SRIKANT, R. Fast algorithms for mining association rules in large databases. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994. (VLDB '94), p. 487–499. ISBN 1-55860-153-8. Disponível em: <<http://dl.acm.org/citation.cfm?id=645920.672836>>. Citado na página 21.
- ANTONIE, M.-L.; ZAIANE, O. R. An associative classifier based on positive and negative rules. In: *Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. New York, NY, USA: Association for Computing Machinery, 2004. (DMKD '04), p. 64–69. ISBN 158113908X. Disponível em: <<https://doi.org/10.1145/1008694.1008705>>. Citado 2 vezes nas páginas 13 e 28.
- ANTONIE, M.-L.; ZAIANE, O. R. Mining positive and negative association rules: An approach for confined rules. In: BOULICAUT, J.-F. et al. (Ed.). *Knowledge Discovery in Databases: PKDD 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 27–38. ISBN 978-3-540-30116-5. Citado na página 18.
- AROOUR, K.; BELKAHLA, A. Frequent pattern-growth algorithm on multi-core cpu and gpu processors. *Journal of Computing and Information Technology*, v. 22, p. 159–169, 01 2014. Citado na página 14.
- ARYABARZAN, N.; MINAEI-BIDGOLI, B.; TESHNEHLAB, M. negfin: An efficient algorithm for fast mining frequent itemsets. *Expert Systems with Applications*, v. 105, p. 129 – 143, 2018. ISSN 0957-4174. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S095741741830188X>>. Citado na página 17.
- BAGUI, S.; DHAR, P. C. Mining positive and negative association rules in hadoop's mapreduce environment. In: *Proceedings of the ACMSE 2018 Conference*. New York, NY, USA: ACM, 2018. (ACMSE '18), p. 33:1–33:1. ISBN 978-1-4503-5696-1. Citado na página 14.

- BAYARDO, R. J. Efficiently mining long patterns from databases. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 27, n. 2, p. 85–93, jun. 1998. ISSN 0163-5808. Disponível em: <<https://doi.org/10.1145/276305.276313>>. Citado na página 16.
- BEMARISIKA, P.; TOTOHASINA, A. Erapn, an algorithm for extraction positive and negative association rules in big data. In: ORDONEZ, C.; BELLATRECHE, L. (Ed.). *Big Data Analytics and Knowledge Discovery*. Cham: Springer International Publishing, 2018. p. 329–344. ISBN 978-3-319-98539-8. Citado 3 vezes nas páginas 14, 18 e 19.
- BIAN, P. et al. Nar-miner: Discovering negative association rules from code for bug detection. *ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, p. 411–422, 2018. Citado 4 vezes nas páginas 14, 18, 19 e 27.
- BORGELT, C. Efficient implementations of apriori and eclat. In: *Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL). CEUR Workshop Proceedings 90*. [S.l.: s.n.], 2003. p. 90. Citado 2 vezes nas páginas 32 e 50.
- BORGELT, C. An implementation of the fp-growth algorithm. In: *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*. New York, NY, USA: ACM, 2005. (OSDM '05), p. 1–5. ISBN 1-59593-210-0. Disponível em: <<http://doi.acm.org/10.1145/1133905.1133907>>. Citado 2 vezes nas páginas 32 e 50.
- BORGELT, C.; KRUSE, R. Induction of association rules: Apriori implementation. In: HÄRDLE, W.; RÖNZ, B. (Ed.). *Compstat*. Heidelberg: Physica-Verlag HD, 2002. p. 395–400. ISBN 978-3-642-57489-4. Citado 2 vezes nas páginas 32 e 50.
- BRAMER, M. *Principles of Data Mining*. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2013. ISBN 1447148835, 9781447148838. Citado na página 25.
- BRIN, S. et al. Dynamic itemset counting and implication rules for market basket data. In: *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 1997. (SIGMOD '97), p. 255–264. ISBN 0-89791-911-4. Disponível em: <<http://doi.acm.org/10.1145/253260.253325>>. Citado na página 26.
- CHON, K.-W.; KIM, M.-S. Ssdminer: A scalable and fast disk-based frequent pattern miner. In: LEE, W. et al. (Ed.). *Proceedings of the 7th International Conference on Emerging Databases*. Singapore: Springer Singapore, 2018. p. 99–110. ISBN 978-981-10-6520-0. Citado na página 17.
- COLANTONIO, A.; PIETRO, R. D. CONCISE: compressed 'n' composable integer set. *CoRR*, abs/1004.0403, 2010. Disponível em: <<http://arxiv.org/abs/1004.0403>>. Citado 2 vezes nas páginas 17 e 24.
- CORNELIS, C. et al. Mining positive and negative association rules from large databases. In: *2006 IEEE Conference on Cybernetics and Intelligent Systems*. [S.l.: s.n.], 2006. p. 1–6. ISSN 2326-8239. Citado 9 vezes nas páginas 7, 13, 18, 27, 39, 41, 42, 44 e 45.

- DELIEGE, F.; PEDERSEN, T. Position list word aligned hybrid: optimizing space and performance for compressed bitmaps. In: *Proceedings of the 13th International Conference on Extending Database Technology, EDBT '10*. United States: Association for Computing Machinery, 2010. p. 228–239. ISBN 978-1-60558-945-9. Citado 2 vezes nas páginas 17 e 24.
- DJENOURI, Y. et al. How to exploit high performance computing in population-based metaheuristics for solving association rule mining problem. *Distributed and Parallel Databases*, v. 36, n. 2, p. 369–397, 2018. Citado na página 14.
- ENGLIN, R. *Indirect association rule mining for crime data analysis*. Dissertação (Mestrado) — EWU Masters Thesis Collection, 2015. Citado na página 14.
- FADISHEI, H.; DOUSTIAN, S.; SAADATI, P. The merits of bitset compression techniques for mining association rules from big data. In: GRANDINETTI, L.; MIRTAHERI, S. L.; SHAHBAZIAN, R. (Ed.). *High-Performance Computing and Big Data Analysis*. Cham: Springer International Publishing, 2019. p. 119–131. ISBN 978-3-030-33495-6. Citado na página 17.
- FANG, W. et al. Frequent itemset mining on graphics processors. In: *Proceedings of the Fifth International Workshop on Data Management on New Hardware*. New York, NY, USA: ACM, 2009. (DaMoN '09), p. 34–42. ISBN 978-1-60558-701-1. Disponível em: <<http://doi.acm.org/10.1145/1565694.1565702>>. Citado 5 vezes nas páginas 6, 14, 23, 24 e 26.
- GAN, V. W. et al. Data mining in distributed environment: a survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, v. 7, p. e1216, 07 2017. Citado na página 13.
- GOETHALS, B. Memory issues in frequent itemset mining. In: *Proceedings of the 2004 ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2004. (SAC '04), p. 530–534. ISBN 1581138121. Disponível em: <<https://doi.org/10.1145/967900.968012>>. Citado na página 16.
- GUZUN, G. et al. A tunable compression framework for bitmap indices. *Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE)*, 03 2014. Citado 2 vezes nas páginas 17 e 24.
- HAN, J.; PEI, J.; YIN, Y. Mining frequent patterns without candidate generation. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2000. (SIGMOD '00), p. 1–12. ISBN 1-58113-217-4. Disponível em: <<http://doi.acm.org/10.1145/342009.335372>>. Citado na página 21.
- HUANG, Y.-S. et al. Accelerating parallel frequent itemset mining on graphics processors with sorting. In: HSU, C.-H. et al. (Ed.). *Network and Parallel Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 245–256. ISBN 978-3-642-40820-5. Citado na página 14.
- HäMÄLÄINEN, W. Kingfisher: An efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures. *Knowledge and Information Systems - KAIS*, v. 32, p. 1–32, 08 2011. Citado 3 vezes nas páginas 13, 18 e 28.

- ITKAR, S.; KULKARNI, U. An efficient and optimised frequent pattern mining using novel multipath-graph structure. *International Journal of Data Mining, Modelling and Management*, v. 9, p. 79, 01 2017. Citado na página 16.
- Jian Pei et al. H-mine: hyper-structure mining of frequent patterns in large databases. In: *Proceedings 2001 IEEE International Conference on Data Mining*. [S.l.: s.n.], 2001. p. 441–448. Citado na página 16.
- JR., R. J. B.; GOETHALS, B.; ZAKI, M. J. (Ed.). *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, v. 126 de *CEUR Workshop Proceedings*, (CEUR Workshop Proceedings, v. 126). CEUR-WS.org, 2005. Disponível em: <<http://ceur-ws.org/Vol-126>>. Citado na página 45.
- KIM, S. et al. Sbh: Super byte-aligned hybrid bitmap compression. *Information Systems*, v. 62, 07 2016. Citado 2 vezes nas páginas 17 e 24.
- KOH, Y. S.; PEARS, R. Efficiently finding negative association rules without support threshold. In: ORGUN, M. A.; THORNTON, J. (Ed.). *AI 2007: Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 710–714. ISBN 978-3-540-76928-6. Citado na página 18.
- LEMIRE, D.; KAI, G. S. Y.; KASER, O. Consistently faster and smaller compressed bitmaps with roaring. *CoRR*, abs/1603.06549, 2016. Disponível em: <<http://arxiv.org/abs/1603.06549>>. Citado 2 vezes nas páginas 17 e 24.
- LEMIRE, D.; KASER, O.; AOUICHE, K. Sorting improves word-aligned bitmap indexes. *CoRR*, abs/0901.3751, 2009. Disponível em: <<http://arxiv.org/abs/0901.3751>>. Citado 2 vezes nas páginas 17 e 24.
- LIU, G. et al. On computing, storing and querying frequent patterns. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2003. (KDD '03), p. 607–612. ISBN 1-58113-737-0. Disponível em: <<http://doi.acm.org/10.1145/956750.956827>>. Citado na página 17.
- LIU, L. et al. Optimization of frequent itemset mining on multiple-core processor. In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. [S.l.]: VLDB Endowment, 2007. (VLDB '07), p. 1275–1285. ISBN 9781595936493. Citado na página 14.
- MAHMOOD, S.; SHAHBAZ, M.; GUERGACHI, A. Negative and positive association rules mining from text using frequent and infrequent itemsets. *The Scientific World Journal*, v. 2014, 05 2014. Citado na página 14.
- MARTÍN, D. et al. Nicgar: A niching genetic algorithm to mine a diverse set of interesting quantitative association rules. *Information Sciences*, v. 355-356, p. 208–228, 2016. Citado 2 vezes nas páginas 19 e 28.
- MCNICHOLAS, P. D.; MURPHY, T. B.; O'REGAN, M. Standardising the lift of an association rule. *Comput. Stat. Data Anal.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, v. 52, n. 10, p. 4712–4721, jun. 2008. ISSN 0167-9473. Disponível em: <<http://dx.doi.org/10.1016/j.csda.2008.03.013>>. Citado 3 vezes nas páginas 13, 14 e 27.

- MI, J. et al. A multiobjective evolution algorithm based rule certainty updating strategy in big data environment. In: *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings*. [S.l.: s.n.], 2018. v. 2018-January, p. 1–6. Citado 4 vezes nas páginas 13, 14, 19 e 28.
- MOONESINGHE, H. D. K.; FODEH, S.; TAN, P.-N. Frequent closed itemset mining using prefix graphs with an efficient flow-based pruning strategy. In: *Proceedings of the Sixth International Conference on Data Mining*. USA: IEEE Computer Society, 2006. (ICDM '06), p. 426–435. ISBN 0769527019. Disponível em: <<https://doi.org/10.1109/ICDM.2006.74>>. Citado na página 16.
- MULA, W.; KURZ, N.; LEMIRE, D. Faster population counts using AVX2 instructions. *CoRR*, abs/1611.07612, 2016. Disponível em: <<http://arxiv.org/abs/1611.07612>>. Citado na página 24.
- PAN, F. et al. Carpenter: Finding closed patterns in long biological datasets. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2003. (KDD '03), p. 637–642. ISBN 1-58113-737-0. Disponível em: <<http://doi.acm.org/10.1145/956750.956832>>. Citado na página 17.
- PASQUIER, N. et al. Discovering frequent closed itemsets for association rules. In: *Proceedings of the 7th International Conference on Database Theory*. London, UK, UK: Springer-Verlag, 1999. (ICDT '99), p. 398–416. ISBN 3-540-65452-6. Disponível em: <<http://dl.acm.org/citation.cfm?id=645503.656256>>. Citado na página 16.
- PEI, J.; HAN, J.; MAO, R. Closet: An efficient algorithm for mining frequent closed itemsets. 04 2001. Citado na página 17.
- PIATETSKY-SHAPIRO, G. Discovery, analysis and presentation of strong rules. In: PIATETSKY-SHAPIRO, G.; FRAWLEY, W. J. (Ed.). *Knowledge Discovery in Databases*. [S.l.]: AAAI Press, 1991. p. 229–248. Citado na página 26.
- PIETERSE, V. et al. Performance of c++ bit-vector implementations. In: . [S.l.: s.n.], 2010. p. 242–250. Citado 2 vezes nas páginas 17 e 24.
- POHL, A. et al. An evaluation of current simd programming models for c++. In: *Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing*. New York, NY, USA: ACM, 2016. (WPMVP '16), p. 3:1–3:8. ISBN 978-1-4503-4060-1. Disponível em: <<http://doi.acm.org/10.1145/2870650.2870653>>. Citado na página 24.
- SAEED, A. A. et al. Compressed bitmaps based frequent itemsets mining on hadoop. In: *Proceedings of the 10th International Conference on Informatics and Systems*. New York, NY, USA: Association for Computing Machinery, 2016. (INFOS '16), p. 159–165. ISBN 9781450340625. Disponível em: <<https://doi.org/10.1145/2908446.2908457>>. Citado na página 17.
- SCHLEGEL, B.; GEMULLA, R.; LEHNER, W. Memory-efficient frequent-itemset mining. In: *Proceedings of the 14th International Conference on Extending Database Technology*. New York, NY, USA: Association for Computing Machinery, 2011. (EDBT/ICDT '11), p. 461–472. ISBN 9781450305280. Disponível em: <<https://doi.org/10.1145/1951365.1951420>>. Citado na página 16.

- SCHLEGEL, B. et al. Scalable frequent itemset mining on many-core processors. In: *Proceedings of the Ninth International Workshop on Data Management on New Hardware*. New York, NY, USA: Association for Computing Machinery, 2013. (DaMoN '13). ISBN 9781450321969. Disponível em: <<https://doi.org/10.1145/2485278.2485281>>. Citado na página 14.
- SILVERSTEIN, C.; BRIN, S.; MOTWANI, R. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, v. 2, n. 1, p. 39–68, Jan 1998. ISSN 1573-756X. Disponível em: <<https://doi.org/10.1023/A:1009713703947>>. Citado 3 vezes nas páginas 13, 18 e 28.
- SOFTLAND. *IBM Quest Synthetic Data Generator*. 2010. [Http://ibm-quest-synthetic-data-generator.soft112.com/](http://ibm-quest-synthetic-data-generator.soft112.com/). Acessado em 10/11/2019. Citado na página 45.
- THIRUVADY, D. R.; WEBB, G. I. Mining negative rules using grd. In: DAI, H.; SRIKANT, R.; ZHANG, C. (Ed.). *Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 161–165. ISBN 978-3-540-24775-3. Citado na página 18.
- VERMA, N.; SINGH, J. A comprehensive review from sequential association computing to hadoop-mapreduce parallel computing in a retail scenario. *Journal of Management Analytics*, v. 4, n. 4, p. 359–392, 2017. Citado na página 14.
- WANG, J.; HAN, J.; PEI, J. Closet+: Searching for the best strategies for mining frequent closed itemsets. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2003. (KDD '03), p. 236–245. ISBN 1-58113-737-0. Disponível em: <<http://doi.acm.org/10.1145/956750.956779>>. Citado na página 17.
- WANG, J. et al. An experimental study of bitmap compression vs. inverted list compression. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. New York, NY, USA: ACM, 2017. (SIGMOD '17), p. 993–1008. ISBN 978-1-4503-4197-4. Disponível em: <<http://doi.acm.org/10.1145/3035918.3064007>>. Citado 2 vezes nas páginas 17 e 25.
- WU, K. et al. Notes on design and implementation of compressed bit vectors. In: . [S.l.]: Lawrence Berkeley National Laboratory, 2001. LBNL/PUB-3161. Citado 2 vezes nas páginas 17 e 24.
- YIN, M. et al. An improvement of fp-growth association rule mining algorithm based on adjacency table. *MATEC Web of Conferences*, v. 189, p. 10012, 01 2018. Citado na página 16.
- ZAKI, M.; HSIAO, C.-J. Charm: An efficient algorithm for closed itemset mining. In: . [S.l.: s.n.], 2002. Citado na página 17.
- ZAKI, M. J. Scalable algorithms for association mining. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 12, n. 3, p. 372–390, maio 2000. ISSN 1041-4347. Disponível em: <<http://dx.doi.org/10.1109/69.846291>>. Citado na página 21.

ZAKI, M. J.; JR, W. M. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. New York, NY, USA: Cambridge University Press, 2014. ISBN 0521766338, 9780521766333. Citado 8 vezes nas páginas [6](#), [9](#), [13](#), [21](#), [22](#), [23](#), [27](#) e [48](#).