UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MINERAÇÃO DE PADRÕES SEQUENCIAIS E GERAÇÃO DE REGRAS DE ASSOCIAÇÃO ENVOLVENDO TEMPORALIDADE

RAFAEL STOFFALETTE JOÃO

ORIENTADORA: PROFª. DRª. MARIA DO CARMO NICOLETTI

São Carlos - SP
Maio/2015

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MINERAÇÃO DE PADRÕES SEQUENCIAIS E GERAÇÃO DE REGRAS DE ASSOCIAÇÃO ENVOLVENDO TEMPORALIDADE

RAFAEL STOFFALETTE JOÃO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Inteligência Artificial.
Orientadora: Drª. Maria do Carmo Nicoletti.

São Carlos - SP
Maio/2015

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

J62mp

João, Rafael Stoffalette.
Mineração de padrões sequenciais e gereação de regras de associação envolvendo temporalidade / Rafael Stoffalette João. -- São Carlos : UFSCar, 2015.
168 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2015.

1. Data mining (Mineração de dados). 2. Regras de associação. 3. Tratamento de temporalidade. 4. Mineração de padrões sequenciais. I. Título.

CDD: 006.312 (20ª)

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Rafael Stoffalette João, realizada em 07/05/2015:

Profa. Dra. Maria do Carmo Nicoletti
UFSCar

Profa. Dra. Heloisa de Arruda Camargo
UFSCar

Prof. Dr. Mario Augusto de Souza Liziér
UFSCar

Prof. Dr. José Augusto Baranauskas
FFCLRP/USP

AGRADECIMENTOS

A Deus, por dar condição física e mental para superar os desafios correntes e perseverar nos meus objetivos.

À minha orientadora, professora Dr[a] Maria do Carmo Nicoletti, por ter se dedicado com tanto empenho a me ensinar; pela capacidade intelectual e seriedade profissional invejáveis, os quais me motivaram a buscar a melhoria constante do meu conhecimento e por reconhecer o meu trabalho nos momentos oportunos, o qual me motivou muito durante esse curso.

Aos professores: Dr[a]. Heloisa de Arruda Camargo, Dr. Mario Augusto de Souza Liziér e Dr. José Augusto Baranauskas por cederem seus preciosos tempos para analisar em detalhes minha dissertação, por contribuírem de forma significativa com sugestões de melhoria e por aceitarem participar da defesa realizada.

Aos meus pais, Antonio Carlos e Joana, por acreditarem sempre no êxito do meu trabalho, por me aconselharem nos meus impasses e me guiarem sempre pelos caminhos certos. Também por todo o sacrifício, em todos os aspectos, que fizeram para que eu pudesse alcançar meus objetivos e pela segurança que sempre tive em saber que me apoiariam em todos os meus planos.

Ao meu irmão, Renato, por ser um exemplo que sempre me inspirou durante toda a vida.

À minha namorada, Fernanda, por demonstrar tamanho companheirismo em todos os momentos difíceis, por compreender e suportar todas as vezes que eu estive ausente, por me incentivar com elogios e sentir orgulho de mim. Também por ceder seus horários livres para atuar como revisora do meu trabalho por diversas vezes.

Ao Departamento de Computação da UFSCar, por me oferecer uma oportunidade para obter o título de mestre, bem como subsidiar financeiramente minha publicação em uma conferência internacional.

À CAPES pelo incentivo, suporte financeiro, por acreditar no potencial desse estudo e na confiança do meu trabalho.

Aos professores do Programa de Pós Graduação em Ciência da Computação da UFSCar, que de alguma forma contribuíram para minha formação.

A todos meus amigos, pelo companheirismo e conselhos. Também por me proporcionarem importantes momentos de descontração.

À empresa supermercado Bandeiras em Osvaldo Cruz – SP pela importante contribuição ao permitir meu acesso aos seus registros de vendas, fornecendo a base de dados utilizadas nos experimentos realizados.

Enfim, agradeço a todos que contribuíram de forma direta ou indiretamente para a obtenção do título de mestre em Ciência da Computação.

RESUMO

A mineração de dados tem como objetivo principal a extração de informações úteis a partir de uma Base de Dados (BD). O processo de mineração viabiliza, também, a realização de análises dos dados (e.g, identificação de correlações, predições, relações cronológicas, etc.). No trabalho descrito nesta dissertação é proposta uma abordagem à extração de conhecimento temporal a partir de uma BD e detalha a implementação dessa abordagem por meio de um sistema computacional chamado S_MEMISP+AR. De maneira simplista, o sistema tem como principal tarefa realizar uma busca por padrões temporais em uma base de dados, com o objetivo de gerar regras de associação temporais entre elementos de padrões identificados. Ao final do processo, uma análise das relações temporais entre os intervalos de duração dos elementos que compõem os padrões é feita, com base nas relações binárias descritas pelo formalismo da Álgebra Intervalar de Allen. O sistema computacional S_MEMISP+AR e o algoritmo que o sistema implementa são subsidiados pelas propostas Apriori, ARMADA e MEMISP. Foram realizados três experimentos distintos, adotando duas abordagens diferentes de uso do S_MEMISP+AR, utilizando uma base de dados contendo registros de venda de produtos disponibilizados em um supermercado. Tais experimentos foram apresentados como forma de evidenciar que cada uma das abordagens, além de inferir novo conhecimento sobre o domínio de dados e corroborar resultados que reforçam o conhecimento implícito já existente sobre os dados, promovem, de maneira global, o refinamento e extensão do conhecimento sobre os dados.

Palavras-chave: S_MEMIS+AR, MEMISP, ARMADA, APRIORI, mineração de padrões sequenciais, regras de associação, tratamento de temporalidade.

ABSTRACT

Data mining aims at extracting useful information from a Database (DB). The mining process enables, also, to analyze the data (e.g. correlations, predictions, chronological relationships, etc.). The work described in this document proposes an approach to deal with temporal knowledge extraction from a DB and describes the implementation of this approach, as the computational system called S_MEMIS+AR. The system focuses on the process of finding frequent temporal patterns in a DB and generating temporal association rules, based on the elements contained in the frequent patterns identified. At the end of the process performs an analysis of the temporal relationships between time intervals associated with the elements contained in each pattern using the binary relationships described by the Allen´s Interval Algebra. Both, the S_MEMISP+AR and the algorithm that the system implements, were subsidized by the Apriori, the MEMISP and the ARMADA approaches. Three experiments considering two different approaches were conducted with the S_MEMISP+AR, using a DB of sale records of products available in a supermarket. Such experiments were conducted to show that each proposed approach, besides inferring new knowledge about the data domain and corroborating results that reinforce the implicit knowledge about the data, also promotes, in a global way, the refinement and extension of the knowledge about the data.

Keywords: S_MEMIS+AR, MEMISP, ARMADA, APRIORI, sequential-pattern data mining, association rules, temporal reasoning.

LISTA DE ALGORITMOS

Algoritmo 2.1 – Pseudocódigo do algoritmo Apriori com geração de regras 44

Algoritmo 2.2 – Pseudocódigo do procedimento *gera_itemsets_candidatos* 45

Algoritmo 2.3 – Pseudocódigo do procedimento *valida_candidatos_frequentes* 47

Algoritmo 2.4 – Pseudocódigo do procedimento para geração de regras de associação baseadas nos *itemsets* frequentes identificados pelo algoritmo Apriori 55

Algoritmo 3.1 –Pseudocódigo do algoritmo para verificação se, dadas duas sequências s_1 e s_2, $s_1 \sqsubseteq s_2$ 67

Algoritmo 3.2 – Pseudocódigo do algoritmo MEMISP 80

Algoritmo 4.1 – Pseudocódigo do procedimento *generate_richer_temporal_assoc_rules* utilizado para geração das regras de associação entre os padrões temporais identificados pelo algoritmo ARMADA 100

Algoritmo 5.1 – Procedimento *memisp_rules*. Geração de regras de associação a partir do conjunto de sequências frequentes obtida pelo MEMISP 105

LISTA DE FIGURAS

Figura 1.1 – Algoritmos de mineração de padrões sequenciais. Cronologia e suas principais influências 24

Figura 2.1 – Fluxograma alto nível do algoritmo Apriori 43

Figura 2.2 – Inserção inicial dos três primeiros *itemsets* na raiz da *árvore hash* ... 49

Figura 2.3 – Criação de um novo nó folha (ligado a seu pai via ramo rotulado 1) e transferência dos *itemsets* da raiz para o nó folha criado 49

Figura 2.4 – Criação de um novo nó folha (com profundidade 3) e transferência do primeiro *itemset* (1,2,4), armazenado em seu pai, para ele 50

Figura 2.5 – Criação de um novo nó folha (com profundidade 3) e transferência do *itemset* (1,3,6,) armazenado em seu pai, para ele 50

Figura 2.6 – Criação de um novo nó folha (com profundidade 3) e armazenamento do *itemset* (1,4,5) no nó folha criado 51

Figura 2.7 – Árvore hash que armazena o conjunto C_3 = {(1,2,4) (1,2,5) (1,3,6) (1,4,5) (1,5,9) (2,3,4) (3,4,5) (3,5,6) (3,5,7) (3,6,7) (3,6,8) (4,5,7) (4,5,8) (5,6,7) (6,8,9)}. Adaptada de (TAN et al., 2005) 51

Figura 2.8 – Busca pelo *itemset* (1,2,5) na árvore *hash* que armazena o conjunto C_3 = {(1,2,4) (1,2,5) (1,3,6) (1,4,5) (1,5,9) (2,3,4) (3,4,5) (3,5,6) (3,5,7) (3,6,7) (3,6,8) (4,5,7) (4,5,8) (5,6,7) (6,8,9)} 53

Figura 3.1 – Primeira etapa do procedimento *Partition-And-Validation* para mineração de bases de dados maiores que a memória interna disponível 76

Figura 3.2 – Segunda etapa do procedimento *Partition-And-Validation* para mineração de bases de dados maiores que a memória interna disponível 77

Figura 3.3 – Listas de índices dos padrões 79

Figura 4.1 – Representação pictórica do intervalo de estado (b_s, s, f_s) no espaço temporal 89

Figura 4.2 – Exemplo de uma sequência de três intervalos de estado e a matriz $R_{3\times3}$ associada 91

Figura 4.3 – Base de dados BD_3, utilizada para apresentação do algoritmo ARMADA. Extraída de (WINARKO; RODDICK, 2007) 97

Figura 4.4 – Listas de índices <s_1>-idx e <s_1s_2>-idx geradas a partir da execução do algoritmo ARMADA na base de dados BD_3 98

Figura 5.1 – Estruturas do MEMISP+AR. Geração de regras a partir da execução do MEMISP 104

Figura 5.2 – Representação das estruturas de dados que armazenam os itens, *itemsets* e sequências 107

Figura 5.3 – Estruturas de dados que armazenam a sequência s=<(a)(a,c,d)> ... 108

Figura 5.4 – Esquematização da estrutura de dados que armazena a MDB, base de dados na memória RAM 109

Figura 5.5 – MDB: Base de dados na memória interna do computador 110

Figura 5.6 – Padrões frequentes identificados pelo MEMISP+AR (*min_sup* = 0,5) 111

Figura 5.7 – Padrões frequentes identificados pelo MEMISP+AR na BDOC (*min_sup* = 0,25) 114

Figura 5.8 – Padrões frequentes identificados pelo MEMISP+AR na BDOC (*min_sup* = 0,2) 114

Figura 5.9 – Padrões frequentes identificados pelo MEMISP+AR na BDOC (*min_sup* = 0,18) 115

Figura 6.1 – Trecho extraído da BDO referente a uma compra registrada por Ca 122

Figura 6.2 – Gráfico comparativo quantitativo de padrões identificados, contendo três ou mais itens de compras por dia da semana, bem como aos padrões globais identificados, para *min_sup* = 0,05 136

Figura 6.3 – Nova estrutura de dados que armazena um *itemset* contemplando os pontos temporais de início e fim em cada característica 142

Figura 6.4 – Na parte superior é mostrada a representação pictórica dos intervalos temporais de uma ocorrência do padrão #ID = 2 (Sequência $<(2,C_1,5)(1,C_4,5)(2,C_5,5)(0,C_{13},4)>$). Na parte inferior é mostrada a matriz (M_{4x4}) que representa as relações dos intervalos em termos da AIA 145

LISTA DE TABELAS

Tabela 1.1 – Mapeamento vertical de bits. Implementando pelo algoritmo SPAM (AYRES et al., 2002) 31

Tabela 1.2 – ITEM_IS_EXIST_TABLE. Implementada pelo algoritmo LAPIN-SPAM para uma transação da BD, t = (a,c,d)(b,c,d,e)(a,c)(a,b)(e,f) 33

Tabela 1.3 – Algoritmos relevantes utilizados para identificação de padrões sequenciais. Ano em que foram propostos, seus autores, influências e principais características. * Algoritmos baseados na metodologia de geração e poda de candidatos, ** Algoritmos baseados na metodologia de crescimento de padrões..... 39

Tabela 2.1 – Base de dados BD de transações (vendas de um supermercado) ... 56

Tabela 2.2 – Contagem de suporte dos itens da BD para formar o conjunto L_1. Todos os itens que comparecem em 40% ou mais das transações da BD são considerados frequentes..... 56

Tabela 2.3 – Pré-candidatos gerados a partir da junção de L_1 com ele mesmo (k=1) 57

Tabela 2.4 – Passagem k = 2 do Apriori. Conjunto de candidatos C_2 gerados a partir de L_1 e respectivos valores de suporte. Os frequentes compõem o conjunto L_2..... 58

Tabela 2.5 – Passagem k = 3. Junção do conjunto de *itemsets* frequentes L_2, com ele mesmo, gerar os pré-candidatos que possivelmente irão compor C_3..... 59

Tabela 2.6 – Passagem k = 3 do Apriori. Conjunto de candidatos C_3 gerados a partir de L_2 e respectivos valores de suporte. Os frequentes compõem o conjunto L_3..... 60

Tabela 2.7 – Conjunto de candidatos C_4 gerados a partir de L_3 com seus respectivos valores de suporte. Os frequentes compõem o conjunto L_4. 60

Tabela 2.8 – Conjunto de candidatos C_5, contendo apenas um *itemset* candidato, gerado a partir de L_4. Também seu respectivo valor de suporte que o valida como o único *itemset* frequente do conjunto L_5 61

Tabela 2.9 – Conjunto de regras de associação geradas a partir do único *itemset* frequente pertencente a L_5 = (a, c, l, m, p). Também, seus respectivos valores de confiança e validação quando a regra de associação for considerada forte..... 62

Tabela 3.1 – Exemplos de uso do Algoritmo 3.1 para a verificação de $s_1 \sqsubseteq s_2$ considerando $s_1=<a_1, a_2, ..., a_n>$ e $s_2=<b_1, b_2, ..., b_m>$, $n \leq m$ 66

Tabela 3.2 – Primeira iteração da execução do algoritmo para validar se $s_1 \sqsubseteq s_2$, em que $s_1=<(a,b)(c)(d,e,f)>$ e $s_2=<(a)(a,c,d)(a,b)(c,d)(d,e,f,g)>$. A variável t representa a iteração atual 68

Tabela 3.3 – Segunda iteração da execução do Algoritmo 3.1 para validar $s_1=<(a,b)(c)(d,e,f)>$ como subsequência da sequência $s_2=<(a)(a,c,d)(a,b)(c,d)(d,e,f,g)>$ 69

Tabela 3.4 – Terceira iteração da execução do Algoritmo 3.1 para validar $s_1=<(a,b)(c)(d,e,f)>$ como subsequência da sequência $s_2=<(a)(a,c,d)(a,b)(c,d)(d,e,f,g)>$ 70

Tabela 3.5 – Primeira iteração da execução do algoritmo para validar $s_1=<(a)(b)(c,d)>$ como subsequência da sequência $s_2=<(a,b)(b,c,d)(e)>$ 70

Tabela 3.6 – Segunda iteração da execução do algoritmo para validar $s_1=<(a)(b)(c,d)>$ como subsequência da sequência $s_2=<(a,b)(b,c,d)(e)>$ 71

Tabela 3.7 – Terceira iteração da execução do algoritmo para validar $s_1=<(a)(b)(c,d)>$ como subsequência da sequência $s_2=<(a,b)(b,c,d)(e)>$. 71

Tabela 3.8 – Base de Dados (D_1) de Sequências 72

Tabela 3.9 – D_2: Base de Dados de Sequências 72

Tabela 3.10 – Sequências de D_2 que contém $s_i=<(b)(a)>$ 72

Tabela 3.11– Padrões sequenciais com relação à base de sequências D_2 73

Tabela 4.1 – As cinco relações temporais (binárias) básicas de Allen. Intervalo A = [a−,a+] (a− < a+) e B= [b−, b+] (b− < b+), sendo que {a−,a+, b−,b+} ⊆ $\Re$. A segunda coluna exibe as correspondentes condições default associadas a cada uma das cinco relações e a terceira coluna mostra uma representação pictórica geral associada a cada uma delas 83

Tabela 4.2 – Subdivisão da relação *during* em três novas relações: *starts*, *finishes* e uma nova relação *during*. Notação e convenção seguem aquelas estabelecidas para a Tabela 4.1 84

Tabela 4.3 – Relações temporais básicas, suas correspondentes relações reversas e a relação de equivalência entre cada um dos pares 84

Tabela 4.4 – Relações temporais básicas entre períodos temporais i_1 e i_2 e suas respectivas representações em termos da relação primitiva *meets* .. 85

Tabela 4.5 – Relações binárias básicas da lógica temporal de Allen (1983), expressas na representação de Höppner, para intervalos de estado (b_s, s, f_s) 90

Tabela 4.6 – Classes de equivalência identificadas em TP(S)..... 95

Tabela 5.1 – BD_2: Base de Dados de Sequências, extraída de (LIN; LEE, 2002)..... 109

Tabela 5.2 – Trecho extraído da BDOC após pré-processamento 113

Tabela 5.3 – Conjunto de regras de associação geradas a partir da sequência frequente s = <(carnebovina,frango,paes)>. Também, seus respectivos valores de confiança e validação quando a regra de associação for considerada forte, admitindo-se um valor de *min_conf* = 0,8..... 116

Tabela 5.4 – Trecho extraído da $BDOC_2$ após pré-processamento 117

Tabela 5.5 – Subsequências frequentes identificadas. Parte do resultado da execução do algoritmo MEMISP+AR na $BDOC_2$ (min_sup = 0,8) .. 118

Tabela 5.6 – Conjunto de regras de associação geradas a partir da sequência frequente s = <(carnebovina,frango,paes,refrigerante)>. Também, seus respectivos valores de confiança e validação quando a regra de associação for considerada forte, admitindo-se um valor de *min_conf* = 0,8..... 119

Tabela 6.1 – Trecho da BDO gerada pelo *script_I*, após a primeira etapa do pré-processamento 123

Tabela 6.2 – Trecho da listagem lexicográfica da BDO, feita na etapa 2 do *script_I*, parte do S_MEMISP+AR..... 125

Tabela 6.3 – Trecho da BDO após seu pré-processamento 127

Tabela 6.4 – Itens frequentes identificados na BDO via S_MEMISP+AR (min_sup = 0,05)..... 128

Tabela 6.5 – Padrões frequentes, contendo 2 itens, identificados na BDO via S_MEMISP+AR, para min_sup = 0,05 130

Tabela 6.6 – Padrões frequentes, contendo mais de 2 itens, identificados na BDO via S_MEMISP+AR, para min_sup = 0,05..... 130

Tabela 6.7 – Conjunto de combinações geradas a partir do padrão frequente #ID = 10, com seus respectivos valores de confiança. Serão consideradas regras aquelas combinações com conf ≥ 0,035 131

Tabela 6.8 – Padrões identificados em dias específicos da semana. Os padrões identificados com (*) são recorrentes durante a semana, i.e., ocorrem em 4 dias ou mais..... 134

Tabela 6.9 – Tabela 6.9 – Conjunto de combinações geradas a partir dos padrões frequente identificados, com seus respectivos valores de confiança. Serão consideradas regras de associação aquelas combinações com valor de confiança ≥ 0,035..... 137

Tabela 6.10 – Conjunto de características intrínsecas sobre o domínio representado pela BDO..... 139

Tabela 6.11 – Base de dados de características intrínsecas (BDO-C), construída a partir da BDO 141

Tabela 6.12 – Trecho da saída do *script_II*. Sequência de características, reescritas no formalismo de Höppner. Cada uma das ternas representa uma característica intrínseca que ocorre em um dia da semana; no caso, a segunda-feira 143

Tabela 6.13 – Conjunto de padrões identificados via S_MEMISP+AR na BDO-C, com seus respectivos valores de suporte. São considerados padrões frequentes aqueles que sobrepõem min_sup = 0,428 144

Tabela 6.14 – Conjunto de combinações geradas a partir do padrão frequente #ID = 2, com seus respectivos valores de confiança. Serão consideradas regras de associação, aquelas combinações com conf ≥ 0,6 147

LISTA DE ABREVIATURAS E SIGLAS

AIA – *Álgebra Intervalar de Allen*

ARMADA – *An algorithm for discovering Richer relative teMporal Association rules from temporal DAta*

BD – *Base de Dados*

BDO – *Base de Dados Original*

BDO-C – *Base de Dados de Características*

BIDE – *Bi-Directional extension*

FreeSpan – *Frequent pattern-projected Sequential PAtterN mining*

GSP – *Generalized Sequential Pattern*

IA – *Inteligência Artificial*

IBM-DM – *IBM Intelligent Data Miner*

LAPIN-SPAM – *Last Position Induction Sequential Pattern Mining*

LTI – *Lógica Temporal Intervalar*

MDB – *Memory Database*

MEMISP – *Memory Indexing for Sequential Pattern Mining*

MEMISP+AR – *MEMory Indexing for Sequential Pattern mining + Association Rules*

MFS+ – *Mining Frequence Sequences*

PrefixSpan – *Prefix-projected Sequential PAtterN mining*

SGBD – *Sistema Gerenciador de Banco de Dados*

SPADE – *Sequential Pattern Discovery using Equivalence classes*

SPAM – *Sequential Pattern Mining*

SPaRSe – *Sequential Pattern Mining with Restricted Search*

TT – *Teoria de Temporalidade*

SUMÁRIO

CAPÍTULO 1 – MINERAÇÃO DE PADRÕES SEQUENCIAIS – PRINCIPAIS CARACTERÍSTICAS E ALGORITMOS RELEVANTES 20

1.1 Mineração de Padrões Sequenciais 20

1.2 Uma Breve Descrição dos Principais Algoritmos Relacionados à Mineração de Padrões Sequenciais e de suas Características 22

1.2.1 Algoritmo Apriori 25

1.2.2 Algoritmo GSP 27

1.2.3 Algoritmo FreeSpan 27

1.2.4 Algoritmo PrefixSpan 28

1.2.5 Algoritmo MEMISP 29

1.2.6 Algoritmo SPADE 29

1.2.7 Algoritmo SPAM 30

1.2.8 Algoritmo LAPIN-SPAM 32

1.2.9 Algoritmo SPaRSe 34

1.2.10 Algoritmo BIDE 35

1.2.11 Algoritmo MFS+ 36

1.2.12 Algoritmo ARMADA 37

CAPÍTULO 2 – O ALGORITMO APRIORI 40

2.1 Considerações Iniciais 40

2.2 Descrição do Algoritmo Apriori 41

2.3 Considerações sobre a Estrutura Árvore *hash* 47

2.4 Geração de Regras de Associação 53

2.5 Um Exemplo de Uso do Apriori 55

CAPÍTULO 3 –A PROPOSTA MEMISP (MEMORY INDEXING FOR SEQUENTIAL PATTERN MINING) 63

3.1 Notação e Conceitos Básicos 63

3.2 MEMISP (Memory Indexing for Sequential Pattern Mining) 74

3.2.1 A técnica Partition-and-Validation 75

3.2.2 Passos do Algoritmo 77

CAPÍTULO 4 – ALGORITMO ARMADA - INCORPORAÇÃO DO ASPECTO TEMPORAL 81

4.1 Considerações Iniciais 81

4.2 A Álgebra Intervalar de Allen 82

4.3 O Algoritmo ARMADA 86

CAPÍTULO 5 – A EXTENSÃO MEMISP+AR: (MEMORY INDEXING FOR SEQUENTIAL PATTERN MINING + ASSOCIATION RULES) 102

5.1 Considerações Iniciais 102

5.2 Sobre o Algoritmo MEMISP+AR 103

5.3 Implementação do MEMISP+AR - Estruturas de Dados e Procedimentos Utilizados 106

5.4 MEMISP+AR × MEMISP 109

5.5 MEMISP+AR - O Processo de Geração de Regras Associativas Considerando Duas Abordagens 111

5.5.1 Sobre o Processo de Identificação de Padrões do MEMISP+AR Considerando Sequências com um Único *Itemset* - Estudo de Caso I 112

5.5.2 Sobre o Processo de Identificação de Padrões do MEMISP+AR Considerando Sequências com Vários *Itemsets* - Estudo de Caso II 116

5.6 Dificuldades de Implementação 120

CAPÍTULO 6 – O USO DO S_MEMISP+AR EM DADOS REAIS – EXPERIMENTOS REALIZADOS E ANÁLISES DE RESULTADOS.....121

6.1 Considerações Iniciais 121

6.2 A BDO – Base de Dados Original Utilizada 122

6.3 Experimento Inicial, Resultados e Análises 127

6.4 O Aspecto Temporal e a BDO.....131

6.4.1 Abordagem 1 — Padrões Temporais Relacionados a Dias da Semana 132

6.4.2 Abordagem 2 — Padrões Temporais Relacionados a Características Intrínsecas 137

6.5 Comentários Adicionais sobre os Três Experimentos Realizados, com Ênfase em Aspectos Comparativos 148

CAPÍTULO 7 – CONCLUSÕES E TRABALHOS FUTUROS 152

7.1 Um Histórico do Desenvolvimento do Trabalho de Pesquisa 152

7.2 Conclusões.....157
7.3 Propostas e Possibilidades Futuras.....158
REFERÊNCIAS.....159
APÊNDICE A.....164
APÊNDICE B.....167

APRESENTAÇÃO

Essa dissertação de mestrado tem foco na investigação de algoritmos de mineração de padrões sequenciais com posterior geração de regras de associação, levando em consideração aspectos temporais inerentes aos dados. No trabalho realizado foram abordados algoritmos das categorias identificadas como (1) geração e poda de candidatos e (2) crescimento de padrões. Dentre os muitos algoritmos encontrados na literatura e brevemente abordados nesta dissertação, os algoritmos MEMISP (LIN; LEE, 2002) e ARMADA (WINARKO; RODDIK, 2007), devido à características discutidas nos capítulos 3 e 4, foram investigados em detalhe. Com base nessa investigação, um novo algoritmo, o MEMISP+AR, que realiza mineração de padrões sequenciais temporais com geração de regras de associação foi proposto. Sua implementação, como o sistema computacional S_MEMISP+AR, foi desenvolvida e utilizada em experimentos usando tanto dados sintéticos quanto dados reais. O trabalho realizado, descrito nesta dissertação, está organizado como segue.

No Capítulo 1 é contextualizada a área de mineração de padrões sequencias e apresenta uma breve revisão dos principais algoritmos na área.

No Capítulo 2 é abordado em detalhes um dos primeiros e mais populares algoritmos para mineração de padrões sequenciais e geração de regras de associação, conhecido como algoritmo Apriori (AGRAWAL; SRIKANT, 1994).

No Capítulo 3 é apresentado e discutido o algoritmo de mineração de padrões sequencias MEMISP(*MEMory Indexing for Sequential Pattern mining*) (LIN; LEE, 2002), um dos mais recentes algoritmos publicados, que promove rapidez de execução devido à estratégia empregada i.e., a de minerar dados em memória.

No Capítulo 4 é abordado o algoritmo ARMADA (WINARKO; RODDICK, 2007) que, juntamente com o MEMISP, serviram de base para a pesquisa conduzida. Inicialmente no capítulo, a Álgebra Intervalar de Allen (AIA) (ALLEN, 1981), um formalismo lógico adequado para a representação e inferência de conhecimento temporal e utilizado pelo algoritmo ARMADA, é apresentada.

No Capítulo 5 é apresentada em detalhes a motivação para o desenvolvimento do algoritmo MEMISP+AR, como uma extensão do algoritmo MEMISP; tal extensão incorpora tratamento de dados temporais (similarmente ao

ARMADA), bem como geração de regras de associação. No capítulo, tanto o sistema computacional S_MEMISP+AR, que implementa o algoritmo proposto, quanto alguns experimentos realizados com vistas à validação da proposta, em dados artificiais e reais, são apresentados.

No Capítulo 6 é descrito o domínio de dados reais utilizado em alguns dos experimentos e reporta os resultados dos experimentos realizados usando o S_MEMISP+AR no domínio considerado.

No Capítulo 7 a finalização da dissertação é apresentada, com um relato cronológico sequencial das várias etapas cumpridas para o desenvolvimento da pesquisa, resumindo os tópicos investigados, comentando sobre as dificuldades encontradas, resultados obtidos, contribuições que a proposta pode trazer e possíveis linhas para a continuidade do trabalho.

Capítulo 1

MINERAÇÃO DE PADRÕES SEQUENCIAIS - PRINCIPAIS CARACTERÍSTICAS E ALGORITMOS RELEVANTES

O processo de mineração de padrões tem como principal objetivo a tarefa de facilitar a descoberta da maior quantidade de informações importantes que podem ser armazenadas em uma base de dados. Para tal, cada domínio requer uma configuração de algoritmo, com técnicas e abordagens que melhor atendam as necessidades. Este capítulo descreve alguns dos principais trabalhos propostos nas área.

1.1 A Mineração de Padrões Sequenciais

Com o crescente volume de informações armazenadas em bases de dados, fica evidente a necessidade, cada vez maior, de técnicas e ferramentas que possam manipular e analisar tais dados, de forma que informações importantes sobre eles possam ser evidenciadas e, assim, contribuam mais efetivamente com o refinamento de qualquer domínio de conhecimento. A mineração de dados tem como foco a extração da maior quantidade de informações úteis, a partir de uma base de dados (BD). O processo de mineração viabiliza, também, a realização de analises (e.g, identificação de correlações, predições, relações cronológicas, etc.), sobre os dados contidos na BD.

O problema caracterizado como *descobrir padrões frequentes em dados sequenciais* é o foco de muitos trabalhos envolvendo mineração de dados, devido à sua alta aplicabilidade. Tal problema, descrito de uma forma simplista, consiste em analisar fragmentos compostos por itens da base de dados, que se repetem com alta frequência na própria BD. Quando tratados computacionalmente, tais fragmentos (na maioria das referências identificados como *itemsets*), geralmente, são gerados a partir dos itens da base de dados, por processos combinatórios (como faz o algoritmo SPADE (ZAKI, 2001)). Isso, de certa forma, promove o aumento no tempo de processamento de vários algoritmos que mineram padrões sequenciais, devido à grande quantidade de fragmentos gerados .

A identificação de padrões frequentes em bases de dados foi, inicialmente, abordada por Agrawal et al. (1993), com a definição formal da extração de regras de associação entre itens. O assunto é amplamente abordado atualmente por ter uma aplicabilidade alta em muitas áreas distintas, tais como: comércio, administração de equipes, medicina, etc. Inicialmente, o foco da extração de regras, proposto por Agrawal, foi a aplicação em transações de comércio, a fim de identificar as relações entre os produtos de uma venda, para aumentar a quantidade de produtos vendidos e controlar melhor a reposição de estoques. Atualmente, propostas, tais como aquelas que descrevem o algoritmo SPADE (ZAKI, 2001) e o algoritmo ARMADA (WINARKO; RODDICK, 2007), focam no problema da temporalidade associada às sequências contidas nas bases de dados. Por meio da incorporação de aspectos temporais, torna-se possível a inferência de regras que, além de evidenciar a relação entre itens, também fornecem informações sobre padrões que ocorrem em intervalos de tempo (duração em que os eventos ocorrem).

O principal objetivo da busca de padrões temporais, que é o assunto tema desse trabalho de pesquisa, é associar as informações provenientes da mineração dos padrões com o fator tempo, de tal forma que os padrões encontrados tem relação direta com inúmeros aspectos variáveis tais como: sazonalidade de mercados, períodos tendenciosos, condições climáticas, recorrência de eventos, entre outras. É importante ressaltar, como será visto ao longo do trabalho descrito nesta dissertação, que as definições formais relativas aos vários conceitos envolvidos na caracterização do problema de encontrar padrões frequentes em dados sequenciais, não têm um consenso rígido e tampouco uma notação padronizada nos vários trabalhos acadêmicos que tratam do assunto. Isso, de certa

forma, promove uma desorientação quando da tentativa de comparar e analisar as diferentes propostas encontradas.

1.2 Uma Breve Descrição dos Principais Algoritmos Relacionados à Mineração de Padrões Sequenciais e de suas Características

Como pode ser evidenciado pela literatura, existem vários algoritmos para a busca e identificação de padrões sequenciais em bases de dados de sequências que, normalmente, conservam muitas características dos algoritmos em que foram baseados. Em uma visão mais ampla, tais algoritmos podem ser classificados em 2 principais grupos, segundo suas abordagens: (1) Abordagem de geração e teste de candidatos e (2) Métodos de crescimento de padrões. Em (1) estão os algoritmos que executam múltiplas varreduras na base de dados. Em cada uma de suas iterações é gerado um conjunto chamado de conjunto de candidatos, os quais são analisados, um a um, e suas frequências calculadas para validar se efetivamente ocorrem com frequência na base de dados, i.e., se são padrões da BD. Os candidatos validados como frequentes, são utilizados como base para a geração de novos candidatos na próxima iteração do algoritmo. Os algoritmos contidos em (2) geralmente fazem uso da recursão para combinar padrões, efetivamente frequentes, contidos na base de dados e gerar padrões de tamanhos maiores. A identificação dos padrões frequentes, nos algoritmos da abordagem de crescimento de padrões, é feita em grupos independentes. Cada um dos grupos gerados contém uma quantidade de padrões locais que são, posteriormente, analisados para serem validados como padrões frequentes da base de dados.

Muitos algoritmos com foco em mineração de padrões sequenciais possuem características bem parecidas pois são variações que conservam características de um mesmo algoritmo. A Figura 1.1 apresenta alguns dos algoritmos relevantes, dispostos em ordem cronológica e organizados de maneira a evidenciar a relação de influência na proposta de vários deles.

Dentre os algoritmos apresentados como referência na mineração de padrões sequenciais, estão alguns dos mais utilizados. Dentre eles destacam-se: (1) os baseados na metodologia de geração e teste de candidatos: Apriori (AGRAWAL;

SRIKANT, 1994), GSP (Generalized Sequential Patterns) (SRIKANT;AGRAWAL, 1996), SPADE (Sequential PAttern Discovery using Equivalence classes) (ZAKI, 2001), SPAM (Sequential PAttern Mining) (AYRES et al., 2002), SPaRSe (Sequential Pattern Mining with Restricted Search) (ANTUNES; OLIVEIRA, 2004), LAPIN-SPAM (LAst Position INduction Sequential PAttern Mining) (YANG; KITSUREGAWA, 2005) e MFS+ (Mining Frequent Sequences plus) (KAO et al., 2005) e (2) os baseados na metodologia de crescimento de padrões: FreeSpan (Frequent pattern-projected Sequential pattern mining) (HAN et al., 2000), PrefixSpan (Prefix-projected Sequential pattern mining) (PEI et al., 2001), MEMISP (Memory Indexing for Sequential Pattern mining) (LIN; LEE, 2002), BIDE (BI-Directional Extension) (WANG; HAN, 2004) e ARMADA (WINARKO; RODDICK, 2007). A partir de 2007 trabalhos encontrados na literatura focalizam a mineração da Web em aplicações tais como levantamento de páginas visitadas (VEERAMALAI et al., *2010)* e (BONDE; GORE, 2014) e análise de transações comerciais via Web (OMARI et al., 2007), (YANG; FONG, 2009) e (SIDDIQUI et al., 2014).

Na literatura podem ser encontrados vários algoritmos que estão voltados à mineração de padrões sequenciais; os evidenciados durante o levantamento bibliográfico, entretanto, são aqueles que foram mais amplamente usados em aplicações práticas e que têm tido maior relevância na área. Tais trabalhos estão listados na Figura 1.1; nela as linhas tracejadas representam os algoritmos da metodologia de geração e teste de candidatos (baseados no Apriori) e as linhas contínuas representam os algoritmos da metodologia de crescimento de padrões. Como o algoritmo ARMADA, um dos focos deste trabalho, é influenciado por algoritmos pertencentes a ambas categorias, é apontado por uma seta diferenciada das demais.

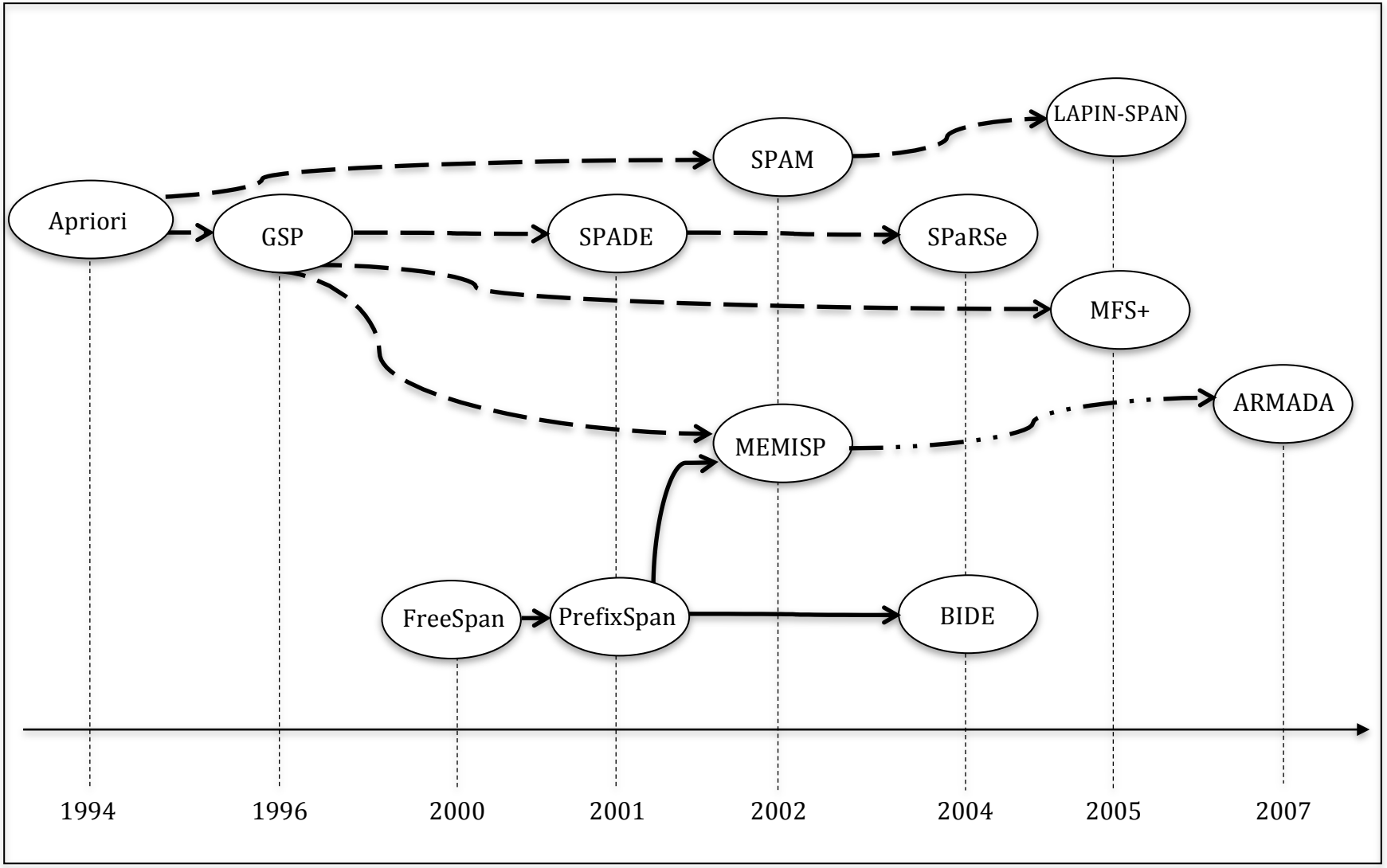


***Figura 1.1 – Algoritmos de mineração de padrões sequenciais*. Cronologia e suas principais influências.**

A Definição 1.1 a seguir introduz notação e alguns conceitos necessários à uma breve apresentação dos principais algoritmos relacionados à mineração de padrões sequenciais, que a segue.

Definição 1.1

(1) Seja um conjunto de *itens* notado por $I=\{i_1, i_2, ..., i_N\}$.

(2) Um *itemset e* é um subconjunto do conjunto de itens, i.e., $e \subseteq I$.

(3) Uma *sequência* s, escrita da forma $s=<e_1e_2...e_n>$, é uma lista ordenada na qual e_j é um *itemset*, i.e., $e_j \subseteq I$, j = 1, ..., n.

(4) Cada *itemset* e_j (j = 1,..., n) da sequência é escrito como $e_j = (i_{j_1}, i_{j_2}, ..., i_{j_m})$, em que $i_{j_k} \in I$ (k=1, ..., m). A notação $|e_j|$ representa o número de itens que definem o *itemset* e_j e, portanto, para o *itemset* $e_j = (i_{j_1}, i_{j_2}, ..., i_{j_m})$, $|e_j| = m$.

(5) Um item $i_k \in I$ (k = 1, ..., N) pode ocorrer, no máximo, uma vez em um particular *itemset* de uma determinada sequência. Um mesmo item i_k pode, entretanto, ocorrer múltiplas vezes em uma sequência, uma vez que pode ocorrer em vários dos *itemsets* que definem a sequência.

(6) Os *itemsets* e_i, $e_j \subseteq I$ (i, j = 1, ..., n) em uma sequência não necessariamente compartilham os mesmos itens e não necessariamente têm a mesma cardinalidade. Pode pois acontecer que: $e_i \cap e_j = \varnothing$, $i \neq j$ ou $|e_i| \neq |e_j|$, $i \neq j$.

(7) Sem perda de generalidade, assume-se que os itens que definem um *itemset* estão em ordem lexicográfica;

(8) Note que, apesar de ser um conjunto, a notação adotada para representar um *itemset* e_j é $e_j = (i_{j_1}, i_{j_2}, ..., i_{j_n})$, em virtude de (7).

1.2.1 Algoritmo Apriori

O algoritmo Apriori, proposto por Agrawal e Srikant (1994), é uma das principais referências na extração de padrões, tanto pelo fato de ter sido um dos primeiros na mineração de padrões, quanto pela sua facilidade de compreensão, além da eficiência na identificação de padrões em bases de dados de transações comerciais. O algoritmo é melhor compreendido quando abordado nas suas três fases consecutivas de execução, descritas a seguir.

Na primeira fase o algoritmo busca os itens frequentes usando, para isso, um limite mínimo pré-definido para o valor de frequência mínima exigida para considerar um item frequente. Após todos itens frequentes terem sido identificados, o Apriori gera combinações de itens, que passam a ser referenciados como candidatos (a *itemsets*), por meio da combinação dos itens efetivamente frequentes.

Na segunda fase os candidatos são efetivamente validados como frequentes e, então, passam a servir de base para a geração de novos candidatos – nessa fase o algoritmo entra em um ciclo repetitivo de geração e validação de candidatos que finaliza apenas quando não são mais identificados novos padrões.

A terceira fase do algoritmo cuida da geração das regras de associação, que tentam evidenciar uma associação entre os itens que comparecem em um *itemset* frequente.

Uma das características mais relevantes do algoritmo Apriori está relacionada à propriedade conhecida como *antimonotonia da relação*, também referida como *propriedade Apriori*. De acordo com essa propriedade, dado um *itemset* e_1, que é parte de outro *itemset*, e_2, (i.e., e_2 contém todos os itens de e_1) e, ainda, e_1 não possui frequência maior que o limite pré-estabelecido para um *itemset* ser considerado frequente, o *itemset* e_2 também não pode ser considerado frequente. A propriedade Apriori garante que nenhum *itemset* que não é frequente pode ser estendido com o objetivo de gerar um novo *itemset*, de tamanho maior, frequente. Assim o algoritmo ganha em velocidade de processamento, uma vez que reduz o número de varreduras da base de dados para contagem de frequência dos possíveis *itemsets* frequentes.

Algoritmos inspirados no algoritmo original Apriori apresentam resultados ruins quando usados em bases de dados que possuem padrões constituídos por *itemsets* com um número alto de diferentes itens; melhores resultados são, geralmente, obtidos em situações que envolvem padrões que apresentam um número baixo de itens diferentes. Ainda, em casos em que a presença de itens frequentes é muito alta ou, então, o valor estipulado para considerar se um item é frequente ou não, for muito baixo, o algoritmo pode ficar lento por ter que manipular uma quantidade muito grande de candidatos ou, então, executar um número expressivo de buscas. O algoritmo Apriori será revisitado no Capítulo 2, em detalhes.

1.2.2 Algoritmo GSP

O algoritmo GSP (*Generalized Sequential Patterns*), proposto por Srikant e Agrawal (1996), pode ser considerado uma variante bem próxima do algoritmo original Apriori. Também é um dos algoritmos mais utilizados em comparações com novas propostas. A grande diferença entre o GSP e o Apriori está relacionada à etapa de criação e eliminação dos candidatos; enquanto o Apriori gera apenas os candidatos formados com o *itemset*, o algoritmo GSP gera os *itemsets* candidatos com itens separados entre si e, também, candidatos com um *itemset* contendo todos os itens. Em uma execução do algoritmo GSP se dois itens são considerados frequentes, não é gerado apenas um candidato combinando os itens (como no algoritmo Apriori) mas sim três candidatos: um contendo os dois itens como um único *itemset* e outros dois considerando cada item como um *itemset* unitário, alternando a ordem de ocorrência de cada *itemset*.

Assim como o Apriori, ao aplicar a propriedade Apriori na geração de candidatos, o GSP implementa técnicas que reduzem a quantidade de candidatos gerados que, muitas vezes, são redundantes. Essa etapa é conhecida como a fase de *poda dos candidatos*.

Na fase de poda o algoritmo GSP é mais refinado que o Apriori pois, enquanto o Apriori elimina um *itemset* inteiro (na dependência de sua frequência ser menor ou igual a um limitante pré-definido), o GSP analisa a frequência de todas as possibilidades de *itemsets* de tamanhos menores que podem ser obtidos a partir do candidato. Cada candidato é manipulado de forma que um de seus itens é removido e o *itemset* (contendo os itens restantes) é validado quanto à sua frequência em relação ao limite mínimo pré-definido. Quando um dos *itemsets* gerados não é validado como frequente, o candidato é eliminado utilizando a propriedade Apriori.

1.2.3 Algoritmo FreeSpan

O algoritmo FreeSpan (*Frequent pattern-projected Sequential PAtterN mining*), proposto por Han et al. (2000), é um dos primeiros algoritmos que seguem a abordagem de crescimento de padrões ao invés da geração e teste de candidatos, como faz o Apriori. O algoritmo analisa uma quantidade bem menor de combinações de subsequências e, consequentemente, seu tempo de execução é menor (em

comparação com aquele do Apriori). Na abordagem de crescimento de padrões frequentes adotada pelo FreeSpan, são levadas em consideração:

- A geração de candidatos é substituída por uma análise da contagem de frequência apenas de *itemsets* relevantes.
- a metodologia de *dividir-e-conquistar*. Tanto a base de dados quanto o conjunto de *itemsets* a serem examinados são particionados.
- A redução do tamanho da base de dados torna possível colocar uma quantidade substancial de dados na memória interna.

O objetivo principal do algoritmo FreeSpan é, a partir de grupos contendo itens frequentes já identificados, durante a primeira etapa do algoritmo, buscar novos *itemsets* frequentes, em um processo recursivo. Cada um dos itens frequentes serve de base para um processo conhecido como projeção da base de dados (BD), em que somente se busca novos padrões na parcela da BD que possui ocorrência(s) do padrão selecionado. Dessa forma, a varredura é feita em subdivisões relacionadas ao item frequente. Os padrões, locais, de tamanho maior vão sendo descobertos à medida que o algoritmo FreeSpan executa chamadas recursivas considerando os *itemsets* frequentes das subdivisões obtidas.

Em media, o algoritmo FreeSpan, executa apenas 3 varreduras da base de dados para identificar todos os padrões existentes. Ao realizar a projeção de bases de dados, o algoritmo analisa uma quantidade bem menor de combinações de subsequências e, consequentemente, executa em menos tempo se comparado ao algoritmo Apriori.

1.2.4 Algoritmo PrefixSpan

O PrefixSpan (Prefix-projected Sequential pattern mining), proposto por Pei et al. (2001), é um dos algoritmos mais eficientes e utilizados para mineração de padrões sequenciais. O PrefixSpan pode ser considerado uma versão que herdou quase todas as características do algoritmo que o inspirou, o FreeSpan.

O algoritmo PrefixSpan difere do FreeSpan ao utilizar os *itemsets* com maior frequência, extraindo seus prefixos, para projetar um grupo de *itemsets* que tem uma grande possibilidade de ocorrência. A partir dos prefixos extraídos, novos *itemsets* frequentes, de tamanho maior são buscados na BD. Essa característica do algoritmo é conhecida como *projeção de prefixos* e é utilizada para reduzir o volume

de dados manipulados. Cada partição projetada é uma base de dados onde o item inicial é prefixo de todos os padrões existentes na partição.

A motivação do algoritmo PrefixSpan de utilizar prefixos para encontrar novos padrões se deve ao fato que qualquer sequência frequente pode sempre ser encontrada pelo acréscimo de um sufixo a um prefixo frequente já identificado.

1.2.5 Algoritmo MEMISP

O algoritmo MEMISP (MEMory Indexing for Sequential Pattern mining), proposto por Lin e Lee (2002), e baseado nos algoritmos GSP e PrefixSpan, será apenas brevemente comentado aqui, uma vez que é tratado com detalhes no Capítulo 3. O MEMISP executa apenas uma única varredura na base de dados, copiando-a para a memória interna (referenciada por MDB) e encontra padrões diretamente da memória, sem a necessidade de gerar candidatos.

Bases de dados com tamanho que ultrapassam a capacidade da memória, são possíveis de serem mineradas por meio da divisão da base em subgrupos que caibam na memória, por meio da implementação de uma estratégia conhecida como *partition-and-validation*.

O MEMISP se mostra muito superior aos algoritmos GSP e PrefixSpan quando o limite mínimo de frequência estabelecido é baixo, i.e., quando se busca identificar apenas as sequências com alta frequência na base de dados.

A principal característica do algoritmo MEMISP é a de utilizar a abordagem de indexação de memória por meio da criação de uma tabela de índices, que associa índices à posição das sequências frequentes na base de dados.

1.2.6 Algoritmo SPADE

O algoritmo SPADE (Sequential PAttern Discovery using Equivalence classes), proposto por Zaki (2001), é baseado no algoritmo GSP; no SPADE, entretanto, o GSP é implementado de forma recursiva e com a incorporação da temporalidade associada à base de dados. O SPADE considera os itemsets frequentes já identificados anteriormente e analisa apenas as sequencias da BD que

possuem relação com tais *itemsets*, diferentemente do GSP que realiza uma varredura completa pela base de dados para cada passagem do algoritmo.

No primeiro passo do SPADE é calculada a frequência das 1-sequências (sequência de comprimento 1, contendo apenas um item), em uma única varredura da base de dados. Os itens frequentes identificados são então dispostos em uma estrutura na memória interna denominada ID-list. O passo seguinte consiste na combinação dos itens dispostos em pares, formando 2-sequências que são, então, utilizadas para o novo cálculo de frequência; essa característica é referenciada como propriedade combinatória. O segundo passo pode também ser executado com apenas uma varredura. Para executar as varreduras da base de dados, o algoritmo SPADE pode ser implementado utilizando tanto a técnica de busca em largura quanto a busca em profundidade.

Outro aspecto relevante é o de considerar as relações existentes entre os eventos, i.e., cada *itemset* da base de dados possui dois pontos temporais a ele associado, que identificam o seu início e fim, respectivamente. A informação relativa à temporalidade é, também, armazenada na ID-list. Assim cada *itemset* pode ser visto como um evento que aconteceu em um certo intervalo de tempo.

1.2.7 Algoritmo SPAM

O algoritmo SPAM (*Sequential PAttern Mining*), proposto por (AYRES et al., 2002) é mais uma referência entre os algoritmos de mineração de padrões sequenciais que seguem a metodologia adotada pelo algoritmo Apriori. O SPAM realiza uma busca em profundidade nas transações de uma BD a fim de identificar padrões na forma de sequências de *itemsets*.

A principal característica que distingue o SPAM dos demais algoritmos é o fato do SPAM introduzir um novo conceito, denominado *mapeamento vertical de bits*, para ajudar o processo de identificação dos padrões na BD. Um mapeamento vertical de bits é uma matriz de valores binários que identificam a ocorrência (ou não) de cada um dos item frequentes nas transações da BD. Cada uma das colunas da matriz referencia um item lido na primeira varredura da BD e cada uma das linhas da matriz representa uma transação da BD.

No primeiro passo de execução do SPAM, é gerado um mapeamento vertical de bits listando, nas colunas da matriz, cada um dos itens identificados na varredura

e sua relação de ocorrência nas transações (ou de conjuntos de transações), identificadas nas linhas da matriz. Os valores contidos nos elementos da matriz são números binários que correspondem à ocorrência do item da coluna na transação da linha; bit 1 quando o item é identificado na transação e 0 quando o item não é identificado.

Por exemplo, considere o conjunto dos itens I = {a, b, c, d, e, f, g, h, i}, identificados na primeira varredura da BD e o conjunto de transações, nela contidas, T= {1, 2, 3, 4, 5, 6}. A Tabela 1.1 apresenta uma simulação de um mapeamento vertical de bits para os itens do conjunto I, levando em conta T.

A leitura da BD inicia-se selecionando, uma a uma, todas as transações da BD para que possam ser analisadas quanto aos itens que nelas ocorrem. Uma nova linha é incluída na matriz de mapeamento vertical de bits a cada transação lida e os itens que ocorrem na transação selecionada são buscados nas colunas da matriz; quando o item é encontrado na coluna, o elemento da matriz armazena o bit 1 e quando não é encontrado, uma nova coluna é gerada, referente àquele item da transação e o elemento da matriz armazena o bit 1. Todos os demais elementos da linha armazenam o bit 0, indicando que nenhum dos demais itens representados nas colunas foi encontrado na transação.

Na tabela é possível ver informações tais como: o item “a” ocorre em todas as transações da BD, o item “c” nas transações 1, 3 e 5, o item “g” em todas as transações a partir da transação 3 e o item “i” ocorre nas transações pares. Considere um limite mínimo de frequência com valor de 50% e o mapeamento vertical de bits apresentado na Tabela 1.1. É possível identificar que os itens “a”, “c”, “e”, “f”, “h” e “i” são itens considerados frequentes na BD.

Tabela 1.1 – Mapeamento vertical de bits. Implementando pelo algoritmo SPAM (AYRES et al., 2002).

ID Transação	**Item frequente**								
	a	b	c	d	e	f	g	h	i
1	1	0	1	0	1	1	0	1	0
2	1	0	0	0	0	0	0	1	1
3	1	0	1	1	1	1	1	1	0
4	1	0	0	0	0	1	0	1	1
5	1	1	1	0	1	1	0	0	0
6	1	0	0	1	1	1	1	0	1

Assim como o Apriori, o SPAM implementa um processo de geração e teste de candidatos. O processo de geração de candidatos pode ser realizado a partir de duas operações básicas: *S-step* e *I-step*; a primeira responsável por incluir um novo *itemset* contendo apenas um item ao final da sequência frequente já identificada e a segunda estende o último *itemset* da sequência frequente já identificada com um novo item. Tanto a S-step quanto a I-step são operações lógicas AND e, consequentemente, tornam o processo de geração de novos candidatos custoso devido à grande quantidade de combinações possíveis.

Uma vez gerados os novos candidatos, um novo mapeamento vertical de bits é realizado para cada uma das sequências geradas, da mesma forma como descrito anteriormente (ver Tabela 1.1) e a contagem de frequência das sequências é feita basicamente pela operação de contagens de bits com valor “1” do mapeamento. Ao final, as sequências frequentes são identificadas.

1.2.8 Algoritmo LAPIN-SPAM

LAPIN-SPAM (*LAst Position INduction Sequential PAttern Mining*) (YANG; KITSUREGAWA, 2005) implementa uma versão melhorada do algoritmo SPAM, ao substituir o mapeamento vertical de bits do SPAM por um novo mapeamento nomeado *ITEM_IS_EXIST_TABLE*. De forma próxima ao mapeamento vertical de bits, a tabela utilizada pelo LAPIN-SPAM armazena valores binários que referem-se à ocorrência dos itens na BD. A grande diferença está no fato que, em vez de simplesmente armazenar se o item ocorre, ou não, na transação da BD, a *ITEM_IS_EXIST_TABLE* contém a informação que descreve a posição da ultima ocorrência de um determinado item, i.e., consiste na criação de uma tabela durante a primeira varredura da DB, que é responsável por armazenar o bit 1 em todas as posições até a ultima posição em que o item ocorre e o bit 0 nas demais. Um exemplo de uma *ITEM_IS_EXIST_TABLE* pode ser visto na Tabela 1.2, em que a cada linha da tabela refere-se a posição na sequência que está sendo analisada e cada coluna refere-se a um item da transação. Considere ainda que a transação analisada de um determinado consumidor seja t=(a,c,d)(b,c,d,e)(a,c)(a,b)(e,f).

Analisando as colunas da tabela é possível observar até quais posições da transação um determinado item pode ocorrer. Por exemplo, a ultima ocorrência do item “a” na transação é na posição 4 (i.e., quarto *itemset*), assim como ocorre

também com o item “b”; já os itens “c” e “d” têm suas ultimas ocorrências nas posições 3 e 2, respectivamente. Os itens “e” e “f” são os únicos que têm sua ultima ocorrência no último *itemset* da sequência.

Tabela 1.2 – *ITEM_IS_EXIST_TABLE*, implementada pelo algoritmo LAPIN-SPAM para uma transação da BD, t = (a,c,d)(b,c,d,e)(a,c)(a,b)(e,f).

Posição	itens					
	a	b	c	d	e	f
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	0	1	1
4	1	1	0	0	1	1
5	0	0	0	0	1	1

Outra característica que pode ser observada em relação à *ITEM_IS_EXIST_TABLE*, como apontado por (YANG; KITSUREGAWA, 2005) é que cada linha da tabela é um vetor de bits que identifica quais itens ocorrem na transação até a posição atual. Essa característica é utilizada para a geração dos candidatos da seguinte forma: o LAPIN-SPAM utiliza uma técnica parecida com o PrefixSpan para projetar a base de dados de acordo com os candidatos gerados. Quando um novo candidato será gerado, a posição da última ocorrência do prefixo selecionado para ser estendido é comparada à *ITEM_IS_EXIST_TABLE.* O vetor de bits da linha referente à ultima posição do prefixo é selecionado e por meio dele é possível identificar quais itens podem, ou não, estender o prefixo analisado.

Por exemplo, se o prefixo (a,b) é selecionado à ser estendido e sua última ocorrência é na posição 4 (ver Tabela 1.2), a linha de numero 4 da *ITEM_IS_EXIST_TABLE* é selecionada e o vetor 110011 é identificado. Os bits 1 representam os itens que podem ocorrer após o prefixo selecionado. Nesse caso, os itens “a”, “b”, “e” e “f” são os únicos que podem ser utilizados para estender o prefixo, pois os itens “c” e “d” não ocorrem a partir da posição 4.

Enquanto o SPAM realiza uma quantidade substancial de operações lógicas AND para verificar se um candidato é frequente, o LAPIN-SPAM pode facilmente implementar o processo de validação de um padrão sequencial pela seguinte condição: se a ultima ocorrência de um item é menor que a última posição do prefixo atual analisado, o item não pode ser considerado para estender o prefixo e se tornar um candidato a padrão, pois não há possibilidade do item ocorrer futuramente.

Em cada iteração posterior é preciso apenas verificar a tabela construída para obter informações se um candidato ocorre antes da posição do prefixo atual, ou não. Dessa forma, e possível realizar o mesmo processo de validação em menos tempo.

O LAPIN-SPAM pode reduzir consideravelmente o espaço de busca durante o processo de mineração de dados. Segundo descrevem os autores, o algoritmo proposto LAPIN-SPAM supera o SPAM em pelo menos três vezes para quaisquer bases de dados utilizadas.

1.2.9 Algoritmo SPaRSe

O algoritmo SPaRSe (*Sequential Pattern Mining with Restricted Search*), proposto por Antunes e Oliveira (2004), é mais um algoritmo baseado no Apriori. O SPaRSe combina as técnicas de geração e teste de candidatos com a restrição do espaço de busca, por meio da projeção da BD. A execução do SPaRSe é feita de forma iterativa e, assim como o Apriori, após identificar itens frequentes, o algoritmo busca padrões de tamanho crescente, a cada iteração executada pelo algoritmo. A condição de parada do SPaRSe é a mesma do algoritmo Apriori i.e., o algoritmo termina quando não existem mais combinações possíveis a serem geradas como candidatos a padrões.

A ideia central do SPaRSe é manter uma lista de suporte de sequências, para cada um dos candidatos gerados e, então, verificar o valor do suporte somente nas sequências que contém todos os candidatos gerados. Assim como no Apriori, o SPaRSe faz uso da propriedade de antimonotonia de relações; caso uma sequência não contenha um padrão frequente, ela não é considerada.

O principal procedimento do SPaRSe é muito próximo ao funcionamento do GPS, a diferença entre ambos está no fato que, enquanto o GSP gera um conjunto de candidatos e, então, valida se são efetivamente frequentes, o SPaRSe gera e testa um candidato de cada vez. O procedimento chamado *satisfies* calcula a frequência de cada candidato e retorna o resultado se o candidato é frequente ou não. Tal tarefa é similar à executada pela poda implementada no GSP.

A grande diferença, no entanto, entre o GSP e o SPaRSe, está no fato do SPaRSe implementar uma restrição no espaço de busca, similar à utilizada pelo algoritmo PrefixSpan. Cada padrão encontrado é associado a um conjunto de sequências em que o padrão ocorre. Tal conjunto é denominado *base de dados de*

suporte. Um novo candidato pode ter seu valor de suporte verificado apenas observando as intersecções na base de dados de suporte dos padrões que geraram o candidato. No algoritmo SPaRSe, um padrão não é apenas uma sequência que se repete, mas também contém a lista de ocorrências do padrão na BD.

1.2.10 Algoritmo BIDE

O algoritmo BIDE (*BI-Directional Extension*), de Wang e Han (2004), é um algoritmo inspirado no algoritmo SPADE que segue a abordagem do método de crescimento de padrões e, de acordo com os autores, é um algoritmo eficiente para descobrir os conjuntos de padrões sequenciais caracterizados como *fechados*. Um padrão sequencial fechado é um padrão sequencial de tal forma que não está contido em um outro padrão sequencial que possui o mesmo valor de suporte.

O BIDE contribui para a área de mineração de padrões sequenciais incorporando um novo paradigma, chamado *Extensão BI-Direcional*, que utiliza técnicas de ambos paradigmas já existentes, a extensão direcional para frente e para trás. A extensão direcional para frente é o termo utilizado para o crescimento dos padrões a partir de prefixos e, também, para verificar se o padrão sequencial é fechado; já a extensão direcional para trás geralmente é utilizada para podar o espaço de busca.

Nos últimos anos estudos tais como os descritos em (PASQUIER et al., 1999), (WANG et al., 2003), (YAN et al., 2003) e (ZAKI; HSIAO, 2003), têm apresentado argumentos convincentes de que minerar padrões sequenciais, em muitos casos, é melhor quando se consideram apenas os padrões sequenciais fechados, pois alcançam os mesmos resultados e em tempos menores.

Como concluem os autores Wang e Han (2004), o BIDE apresenta uma alta eficiência, com uso de menos memória, em muitos casos. O algoritmo também tem uma boa escalabilidade para qualquer que seja o tamanho da base de dados.

Ao encontrar uma novo padrão o BIDE faz uso do paradigma Bi-Direcional para verificar se o padrão é um padrão fechado, a fim de gerar um conjunto não redundante de sequências frequentes. O funcionamento do BIDE é bem próximo àquele da maioria dos algoritmos que se enquadram no método de crescimento de padrões. O BIDE executa uma varredura na BD e identifica os itens frequentes e, em seguida, faz uma projeção da base de dados para cada um dos itens frequentes

identificados. Cada item frequente se torna um prefixo para novos padrões a serem identificados.

Cada um dos padrões identificados durante a execução do algoritmo é utilizado como input do procedimento *BackScan*, que tem a tarefa de realizar uma poda, verificando se o padrão é, ou não, um padrão fechado. Recursivamente, o BIDE utiliza o padrão encontrado como um prefixo para identificar padrões de tamanho maiores, para cada novo prefixo. Primeiro é verificado se o prefixo pode ser podado e, caso não possa, é calculado o número de extensões que podem ser geradas na base de dados projetada. Então, após essa etapa o BIDE invoca a si mesmo recursivamente, utilizando como parâmetro o prefixo selecionado.

1.2.11 Algoritmo MFS+

O algoritmo MFS+ é uma variação melhorada do MFS (*Mining Frequent Sequences*); ambos os algoritmos foram propostos para mineração de padrões em bases de dados de transações, tais como compras de um consumidor em um supermercado. Basicamente o MFS+ é como um algoritmo em dois estágios bem próximo ao GSP que, entretanto, reduz consideravelmente a quantidade de varreduras na BD (KAO et al., 2005). No primeiro estágio, uma parte da BD é minerada a fim de obter uma estimativa grosseira das sequências frequentes. A partir de tal estimativa, a BD é varrida novamente para checar a existência dos padrões identificados e refinar a busca de novos padrões.

A grande diferença entre o MFS+ e o GSP está, exclusivamente, na tarefa de geração de candidatos. Enquanto o GSP, a cada iteração, gera um conjunto de candidatos incrementando os padrões já identificados em 1 item, ou seja, padrões de tamanho k são utilizados para gerar candidatos de tamanho k + 1, o MFS+ utiliza um conjunto de sequencias frequentes de tamanhos variados para gerar candidatos também de tamanhos variados. Tal abordagem é denominada *refinamento sucessivo de padrões*.

Inicialmente o MFS+ realiza uma estimativa grosseira dos padrões de tamanhos variados na BD, tarefa executada na BD. Posteriormente, se a BD é atualizada constantemente, novos padrões são buscados a partir da estimativa do passo anterior. Caso não exista uma estimativa previamente gerada, o algoritmo

GSP é utilizado em uma pequena parcela da BD (em torno de 10%) para gerar um conjunto de padrões que servirão como a estimativa inicial.

A partir do passo em que a estimativa inicial é feita, o MFS+ primeiro vare a BD novamente para validar quais padrões identificados são efetivamente frequentes. Um conjunto chamado MFSS é então criado, contendo todos os padrões validados e o procedimento de geração de candidatos é executado a partir do conjunto MFSS. Como resultado de tal processo é gerado um conjunto de candidatos de tamanhos variados.

Os candidatos gerados no passo anterior são então validados quanto a serem frequentes ou não na BD e os padrões frequentes são então utilizados para refinar o conjunto MFSS. O processo de geração e validação dos candidatos é interrompido quando não são gerados mais candidatos. Como comentam os autores em (KAO et al., 2005), o MFS+ é capaz de validar grandes sequências rapidamente, fato esse que supera consideravelmente o desempenho do GSP; a referência em questão apresenta resultados empíricos que mostram que o MFS+ identifica os mesmos padrões que o GSP, com a vantagem do primeiro não gerar candidatos desnecessários.

1.2.12 Algoritmo ARMADA

O algoritmo ARMADA (An algorithm for discovering Richer relative teMporal Association rules from temporal DAta), proposto por Winarko e Roddick (2007), é um dos algoritmos de mineração de padrões sequenciais mais recentes. É uma extensão do algoritmo MEMISP e tem dois objetivos específicos: (1) encontrar padrões temporais e (2) gerar regras de associação que correlacionam os eventos da base de dados levando em consideração, para a associação, os intervalos de duração de tais eventos. As relações entre os intervalos de tempo, consideradas pelo algoritmo, são baseadas na lógica temporal intervalar de Allen (discutida com maior riqueza de detalhes no Capítulo 4). Assim como o MEMISP, o algoritmo não gera candidatos e tampouco faz projeção de base de dados. Faz indexação de memória para organizar os padrões e suas respectivas posições na base de dados. Executa uma única varredura inicial na base e faz uma cópia para a memória interna, calculando a frequência de todas as 1-sequências.

A segunda etapa do algoritmo gera um conjunto de índices com o objetivo de associar cada uma das sequências frequentes à sua posição na base de dados. Na terceira e última etapa do algoritmo, novas buscas são feitas e os conjuntos de índices são atualizados de forma recursiva, a fim de descobrir todos os padrões frequentes existentes na base de dados. O algoritmo ARMADA, assim como o SPADE, busca padrões sequenciais temporais e, para tal, considera que *itemsets* são eventos (i.e., acontecimentos quaisquer, associados a seus respectivos tempos de duração). Eventos possuem dois pontos temporais a ele associados, determinando seu início e fim, respectivamente. Cada *itemset* ocorre em um intervalo de tempo, duração entre o ponto de início e o ponto do fim do *itemset*, assim como há um intervalo de tempo entre o ponto que determina o fim de um evento e o início de um evento seguinte.

Um novo valor de constante pré-definido, o *maximum_gap*, é introduzido pelo ARMADA, a fim de refinar a identificação de padrões temporais reduzindo o volume de eventos encontrados. O *maximum_gap* define a janela máxima permitida entre a ocorrência de dois eventos (*itemsets*) para que possam ser considerados frequentes.

Algoritmos que seguem a metodologia de crescimento de padrões geralmente são mais eficientes e escaláveis devido aos seguintes fatores:

- Se beneficiam da estratégia de dividir-e-conquistar;
- Fazem uso da memória principal – Processamento localizado, realizando apenas um único acesso ao BD;
- Refinam a propriedade Apriori, evitando gerar grande quantidade de candidatos.

Assim, os algoritmos que se baseiam na metodologia de crescimento de padrões são mais difundidos atualmente pois são preferíveis àqueles que utilizam métodos baseados no algoritmo Apriori.

A Tabela 1.3 apresenta, de forma condensada, os algoritmos apresentados nesse capítulo e resume as principais características de cada um dos algoritmos abordados nesse capítulo indicando, também, seus respectivos autores, ano em que foi divulgado e qual metodologia que seguem.

Tabela 1.3 – Algoritmos relevantes utilizados para identificação de padrões sequenciais. Ano em que foram propostos, seus autores, influências e principais características. * Algoritmos baseados na metodologia de geração e poda de candidatos, ** Algoritmos baseados na metodologia de crescimento de padrões.

Abordagem	Algoritmo	Ano	Autor(es)	Características
*Apriori	Apriori	1994	Agrawal e Srikant	• Pioneiro e mais popular • Mineração incremental • Antimonotonia de relações • Utiliza árvore-hash
	GSP	1996	Srikant e Agrawal	• Poda de candidatos refinada
	SPADE	2001	Zaki	• Projeção vertical da base de dados • Busca padrões inter-eventos (temporalidade)
	SPAM	2002	Ayres et al.	• Mapeamento vertical de bits
	SPaRSe	2004	Antunes e Oliveira	• *base de dados de suporte*
	LAPIN-SPAN	2005	Yang e Kitsuregawa	• *ITEM_IS_EXIST_TABLE*
	MFS+	2005	Kao et al.	• Refinamento sucessivo de padrões • Gera candidatos de tamanhos variáveis
**Crescimento de Padrões	FreeSpan	2000	Han et al.	• Dividir-e-conquistar • Projeção da base de dados • Escalabilidade linear
	PrefixSpan	2001	Pei et al.	• Projeção da base de dados baseada em prefixos
	MEMISP	2002	Lin e Lee	• Copia BD para a memória RAM • Realiza uma única varredura na base • Faz indexação da base de dados • Procedimento *Partition-and-Validation*
	BIDE	2004	Wang e Han	• Extensão Bi-Direcional • Mineração de sequências fechadas
	ARMADA	2007	Winarko e Roddick	• intervalos temporais • *maximum-gap* • Lógica intervalar de Allen • Estratégia *find-then-index*

Capítulo 2

O ALGORITMO APRIORI

Este capítulo descreve, em detalhe, um dos trabalhos mais utilizados como referência para novas propostas na área de mineração de padrões. O Algoritmo Apriori destaca-se por sua simplicidade e eficiência, ao empregar técnicas pioneiras para o processo de identificação de padrões e geração de regras de associação.

2.1 Considerações Iniciais

O algoritmo Apriori, proposto por Agrawal e Srikant (1994), foi um dos primeiros algoritmos com foco em mineração de padrões sequenciais. Sua popularidade até hoje se deve, principalmente, à sua simplicidade e seu fácil entendimento. Quando implementado de maneira cuidadosa, levando em conta estruturas de dados que sejam apropriadas, tem se mostrado bastante promissor em competições voltadas à avaliação de implementações de algoritmos que fazem mineração de dados frequentes (ver, por exemplo, resultados em (GOETHALS; ZAKI, 2003)).

O algoritmo popularizou-se, também, por ter sido utilizado no desenvolvimento do IBM *Intelligent Data Miner* (IBM-DM), um software voltado à mineração de dados a partir de bases de dados que armazenam grandes volumes de informação. O algoritmo foi essencialmente idealizado e desenvolvido para operar com bases de dados contendo *transações*, em que uma transação é caracterizada como um conjunto de itens. Via de regra um item representa um determinado objeto de um determinado domínio; é o caso, por exemplo, da

transação t = (leite, café, pão, açúcar), que representa um conjunto de quatro itens que participaram de uma transação de venda a consumidor, feita por um supermercado.

No trabalho em que o Apriori é proposto (AGRAWAL; SRIKANT, 1994) o algoritmo gera como saída um conjunto de *itemsets* frequentes. O problema da geração de regras a partir dos *itemsets* frequentes é tratado superficialmente, por meio de um algoritmo trivial, brevemente descrito em linguagem natural. Na Seção 2.4, que trata da geração de regras de associação, o pseudocódigo associado ao procedimento de construção de regras de associação foi baseado na descrição encontrada em (AGRAWAL; SRIKANT, 1994).

2.2 Descrição do Algoritmo Apriori

Com vistas a organizar sua apresentação e padronizar a sua descrição, o algoritmo Apriori discutido nesta seção, tem por *input* (a) uma base de dados (BD) contendo um conjunto de transações (cada uma delas tratada como um *itemset*; no que segue ambos os termos são usados como referência a um mesmo conceito); (b) um conjunto I contendo todos os possíveis itens de dados disponíveis no domínio de aplicação considerado e (c) um valor numérico (*freq*) a ser usado para a caracterização de um item frequente. É importante lembrar que o Apriori não faz uso do conceito de sequência, que é empregado, por exemplo, pelo MEMISP.

Basicamente, o foco do algoritmo Apriori é a busca, em uma base de dados que lhe é informada, de conjuntos de *itemsets* que sejam frequentes para, posteriormente, gerar as chamadas *regras de associação*. Regras de associação nada mais são que regras que associam a presença de determinados itens (em uma transação), à presença de outros.

Um fluxograma do algoritmo Apriori em alto nível é apresentado na Figura 2.1 e nele podem ser identificados três procedimentos principais, a saber: *itemsets_frequentes_uni*, *gera_itemsets_candidatos* e *valida_candidatos_frequentes*.

O procedimento *itemsets_frequentes_uni* é invocado apenas uma vez logo no início do processamento (passo k = 1); inicialmente determina o suporte de cada item de I, com base nas transações armazenadas na BD e, então, constrói L_1, que

conterá apenas aqueles itens com suporte maior que o valor *freq*, fornecido como *input* ao algoritmo. Apesar de não ter sido tratado no documento em que foi proposto, neste trabalho a geração de L_1 é feita de maneira tal que, ao final da geração, os itens participantes de L_1 estão em ordem lexicográfica.

O algoritmo Apriori entra então em um ciclo de repetição em que, primeiro incrementa o contador do passo e, então, faz chamadas aos dois procedimentos i.e., o *gera_itemsets_candidatos e, então, o valida_candidatos_frequentes,* permanecendo no ciclo até que o procedimento *valida_candidatos_frequentes* gere um conjunto vazio de tais candidatos.

O procedimento *gera_itemsets_candidatos* expande o conjunto de candidatos (*itemsets*) frequentes gerado no passo anterior, com tamanho k−1, em 1 item, gerando assim um conjunto de *itemsets* candidatos (a serem frequentes), cada um deles com k itens. Tais *itemsets* são então promovidos (ou não) a *itemsets* frequentes pelo procedimento *valida_candidatos_frequentes*, dependendo ou não do valor do suporte de cada um deles. Aqueles que forem promovidos formam então o conjunto L_k que, se não for vazio, provoca o reinício do ciclo de chamadas.

O algoritmo Apriori (AGRAWAL; SRIKANT, 1994), apresentado em Algoritmo 2.1, é descrito na notação adotada nesta dissertação. É importante observar que o procedimento *valida_candidatos_frequentes*, parte do ciclo repetitivo interno ao algoritmo, faz uma varredura na BD.

Uma a uma, todas as transações que compõem a BD são selecionadas e o valor de suporte dos candidatos que são encontrados nelas, é incrementado. Dessa forma, após a análise de todos os candidatos gerados, é possível identificar aqueles frequentes a partir de seus valores de suporte.

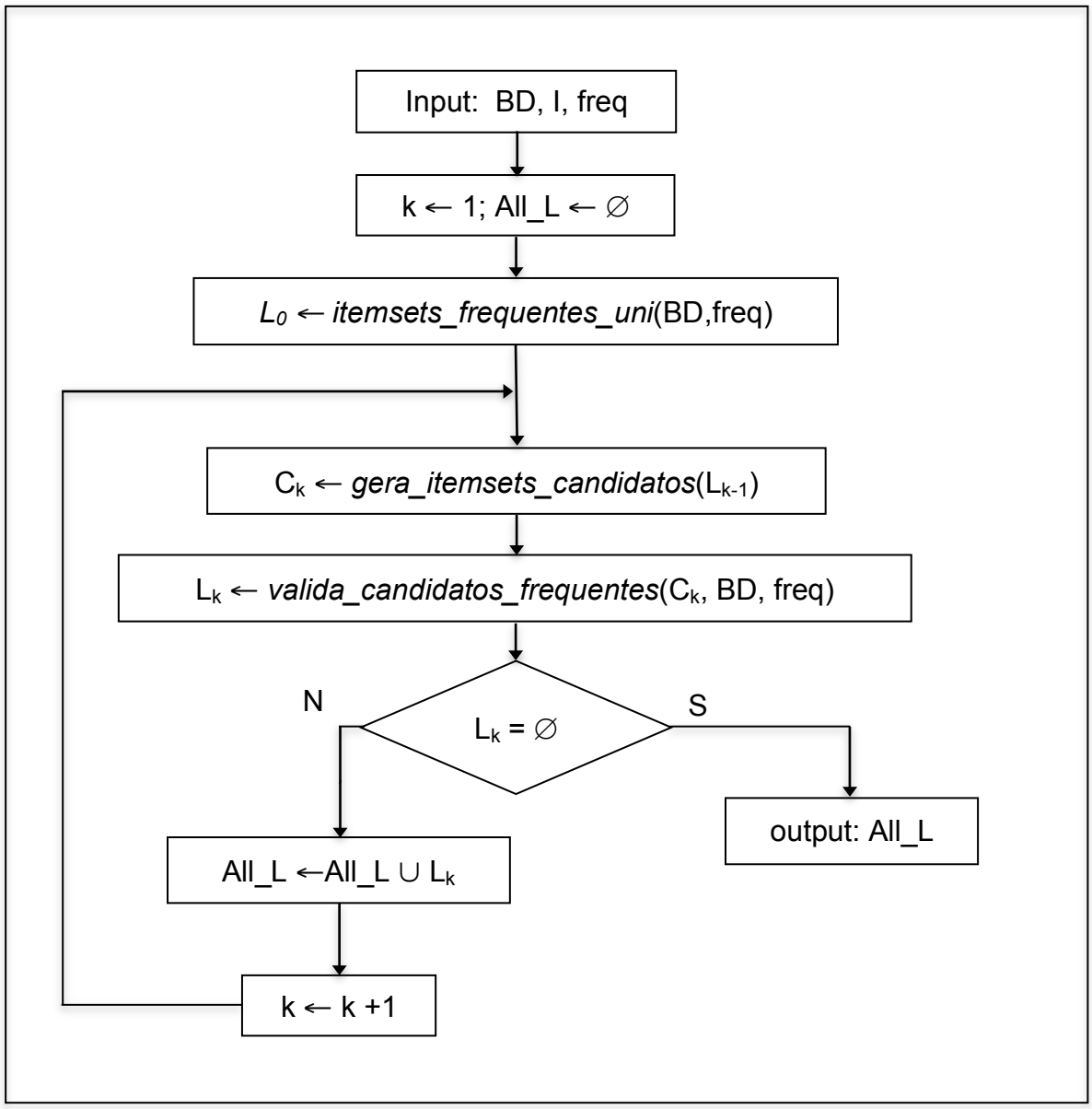


Figura 2.1 –Fluxograma alto nível do algoritmo Apriori.

```
Procedure Apriori(BD, freq)
Entrada: BD = { t1,t2,...t|BD|}
         freq: {limite mínimo de frequência}
Saída: All_L = {L1,L2,...Lk} {conjunto com todos itemsets frequentes de BD}
       R {conjunto de regras de associação}

1. begin
2.   All_L ← ∅
3.   L1 ← itemsets_frequentes_uni(BD,freq)
4.   k ← 2
5.   while (Lk-1 ≠ ∅) do
6.   begin
7.     {is1,is2,...,isM} ← gera_itemsets_candidatos(Lk-1) {1a. poda é feita pelo procedimento}
8.     {a seguir, o suporte de cada um dos M itemsets candidatos é determinado}
9.     for i ← 1 to |BD| do    {para cada transação da BD}
10.    begin
11.      for j ← 1 to M do     { para cada itemset candidato}
12.      begin
13.        if isj ⊆ ti then incrementa_counter(isj)
14.      end
15.    end
16.    {a seguir é construído Lk, com itemsets cujo counter for maior que min_sup}
17.    Lk ← ∅
18.    for j ← 1 to M do     {para cada itemset candidato}
19.      begin
20.        if counter(isj) > freq then Lk ← Lk ∪ {isj} {2a. poda é feita
21.                                                    {procedimento valida_candidatos_frequentes }
22.        All_L ← All_L ∪ Lk
23.        k ← k +1
24.      end
25.  end
26.  R ← gera_regras(All_L,R)  {ver Algoritmo 2.4}
27.  return (All_L, R)
28. end procedure
```

Algoritmo 2.1 – Pseudocódigo do algoritmo Apriori com geração de regras.

O Algoritmo 2.2 mostra o procedimento *gera_itemsets_candidatos* descrito em pseudocódigo. Levando em conta o conjunto de *itemsets* frequentes construído no passo anterior, o procedimento busca exaustivamente por pares de *itemsets* que possam ser aglutinados em um único *itemset*, cujo tamanho será exatamente 1 a mais do que o tamanho dos pares considerados.

```
Procedure gera_itemsets_candidatos(L_{k-1})
 Entrada: L_{k-1} {Conjunto de itemsets frequentes da iteração k–1 do ciclo interno}
 Saída:  C_k {Conjunto com todos os candidatos a  itemsets frequentes}

1. begin
2.   C_k ← ∅
3.   for i ← 1 to |L_{k-1}| – 1 do
4.     for j ← 1 to |L_{k-1}| do
5.     begin
6.       if can_join(is_i,is_j,k–1) then new_is ← append(is_i,last_item(is_j))
7.       C_k ← C_k ∪ new_is
8.     end
9.   return C_k
10. end


Procedure can_join(is_1,is_2,k)
 Entrada: is_1, is_2 {itemsets para serem comparados}
          k = |is_1| = |is_2|
Saída: true ou false

1. begin
2.   for i ← 1 to k – 1 do
3.   begin
4.     if is_1[i] ≠ is_2[i] then return false
5.   end
6.   if is_1[k] ≥ is_2[k] then return false
7.   else return true
8. end
```

Algoritmo 2.2 – Pseudocódigo do procedimento *gera_itemsets_candidatos.*

A condição para que dois *itemsets* possam ser aglutinados é a de que ambos sejam exatamente iguais até o penúltimo de seus itens e cujos últimos itens são tais que o último do primeiro itemset é lexicograficamente menor que o último do segundo *itemset* sendo considerado. Tal condição é formalizada a seguir, na Definição 2.1.

Definição 2.1 Considere dois *itemsets* $is_1 = (i_1, i_2, ..., i_{k-1})$ e $is_2 = (j_1, j_2, ..., j_{k-1})$ tal que $|is_1| = |is_2| = k–1$ (i.e., ambos possuem k–1 itens). Um *itemset* is_3, $|is_3| = k$, pode ser gerado a partir da extensão de is_1, com o último item de is_2 se satisfizerem as duas condições a seguir: (1) ambos possuem seus k–2 primeiros itens iguais e (2) o último item de is_1 é lexicograficamente menor que o último item de is_2, ou seja, $i_1 = j_1$, $i_2 = j_2$, ..., $i_{k-2} = j_{k-2}$, $i_{k-1} < j_{k-1}$. O *itemset* gerado será pois $is_3 = (i_1 = j_1, i_2 = j_2, ..., i_{k-2} = j_{k-2}, i_{k-1}, j_{k-1})$.

O procedimento *gera_itemsets_candidatos* (Algoritmo 2.2) é a descrição, em pseudocódigo, da operação *join*, seguida pela operação *prune*, ambas descritas na referência (AGRAWAL; SRIKANT, 1994), Como comentado em tal referência, a satisfação da condição is1[k] < is2[k] (que permite o procedimento *can_join* retornar com valor true) garante que o procedimento *gera_itemsets_candidatos* não gerará *itemsets* duplicados.

Uma vez gerado o conjunto dos candidatos a *itemsets* frequentes, cada um dos candidatos é validado, para confirmar se é, efetivamente, um *itemset* frequente. O procedimento *valida_candidatos_frequentes*, descrito no Algoritmo 2.3, implementa a tarefa de validação. Antes, entretanto, para tornar o processo mais inteligível, é detalhado o exemplo apresentado em (AGRAWAL; SRIKANT, 1994).

Considerando o conjunto de itens I = {1,2,3,4,5}, suponha que L_3 = {(1,2,3), (1,2,4), (1,3,4), (1,3,5), (2,3,4)}. A execução do *gera_itensets_candidatos*(L_3) para a geração do conjunto de candidatos contendo 4 itens, como descrito na Definição 2.1, produz o conjunto C_4 = {(1,2,3,4),(1,3,4,5)}. Na sequência o procedimento *valida_candidatos_frequentes* avalia cada *itemset* de C_4, com vistas a gerar o conjunto daqueles que forem efetivamente frequentes, isto é, elementos do conjunto L_4.

No exemplo em questão o *valida_candidatos_frequentes*: (1) irá manter o *itemset* (1,2,3,4), uma vez que os (k−1)-*itemsets* (i.e., com 3 itens) dele derivados, a saber, (1,2,3), (1,2,4), (1,3,4) e (2,3,4), estão todos em L_3 (ou seja, todos são frequentes) e (2) irá remover o *itemset* (1,3,4,5), uma vez que tal *itemset,* que tem como 3-*itemsets*: (1,3,4), (1,3,5), (1,4,5) e (3,4,5), tem dois deles, o (1,4,5) e o (3,4,5), que não estão em L_3.

Algoritmos baseados no Apriori, como por exemplo o GSP, possuem uma abordagem muito próxima daquela usada pelo Apriori, quando da geração de candidatos a *itemsets* frequentes. A partir de itens (unitários) tais algoritmos geram *itemsets* candidatos que, por sua vez, servem de base para a geração de novos candidatos, a cada iteração do algoritmo. Tais algoritmos geralmente diferem na forma como os candidatos a *itemsets* frequentes são avaliados, com o objetivo de confirmar se são efetivamente frequentes ou, então, diferem no número de varreduras feitas da BD, para a execução do algoritmo.

```
Procedure valida_candidatos_frequentes(Ck,Lk-1)
Entrada: Ck {Conjunto de candidatos (de tamanho k) gerados}
         Lk-1 {Conjunto com todos itemsets frequentes da passagem k-1}
 Saída: Lk {Conjunto com todos itemsets frequentes no passo k}

1.  begin
2.    Lk ← Ck
3.    for i ← 1 to |Ck| do
4.    begin
5.      Z ← todos_itemsets_tamanho_k-1(isi)
6.      j ← 1
7.      continua ← true
8.      while j ≤ |Z| and continua do
9.        if isj ∉ Lk-1 then
10.       begin
11.         Lk ← Lk – isi
12.         continua ← false
13.       end
14.       else j ← j +1
15.     end
16.   return Lk
17. end procedure
```

Algoritmo 2.3 – Pseudocódigo do procedimento *valida_candidatos_frequentes*.

Os *itemsets* pertencentes a C_k são armazenados em uma estrutura de dados denominada árvore-hash, discutida na próxima seção.

2.3 Considerações sobre a Estrutura *Árvore hash*

O processo de validação dos candidatos gerados (ver Algoritmo 2.3), tanto para o armazenamento do conjunto de candidatos (C_k), quanto para a contagem do suporte dos *itemsets* contidos em C_k, pode ser adaptado para a utilização de uma estrutura de dados denominada *árvore hash*. Cada um dos nós da árvore *hash*, pode conter uma lista de *itemsets* (nós folhas), ou simular uma tabela *hash* (nós internos).

Os *itemsets* de C_k são inseridos, um a um, no passo em que são gerados, sempre nos nós folhas. Quando a quantidade de *itemsets* em um nó folha excede um limite máximo pré-definido, o nó folha é convertido em um nó interno e os *itemsets* armazenados no nó são realocados em novos nós folhas. A escolha de

qual ramo seguir, tanto para armazenamento, quanto busca do *itemset*, é feita por meio da aplicação de uma função *hash* ao item na posição referente a profundidade do nó interno alcançado (e.g., durante um percurso, ao atingir um nó interno de profundidade d, uma função *hash* é aplicada ao item na posição d do *itemset* e o resultado indica qual ramo do nó seguir). A raiz da árvore *hash* é definida com profundidade 1.

O exemplo a seguir, adaptado de (TAN et al., 2005), descreve o processo de armazenamento, dos *itemsets* candidatos gerados, em uma *árvore hash*. Em seguida, exemplifica a busca pelos *itemsets* de uma transação da BD na árvore gerada.

Dados (a) um conjunto de candidatos C_3 = {(1,2,4) (1,2,5) (1,3,6) (1,4,5) (1,5,9) (2,3,4) (3,4,5) (3,5,6) (3,5,7) (3,6,7) (3,6,8) (4,5,7) (4,5,8) (5,6,7) (6,8,9)}, (b) um limitante máximo de três *itemsets* por nó folha e (c) uma função *hash de dois argumentos, itemset* e a profundidade (p) em que o nó da árvore sendo visitado está; o valor da função aplicada aos dois argumentos será um valor inteiro, que indicará qual o ramo seguinte a ser percorrido.

Para o exemplo em questão, considerando um novo itemset is_{new} a ser armazenado na árvore e um nó interno a uma profundidade 2, a função *hash* escolhida foi hash(is_{new},2) → is_{new}[2] mod 3 ou seja, a função identifica o segundo item do *itemset* is_{new} e, a ele, aplicada a função mod 3, que, por exemplo, poderia resultar no valor 0 (zero) indicando, com isso, que a 'descida' na árvore *hash* na busca da posição onde inserir is_{new} é via ramo de label 0 associado ao nó considerado.

Todos os *itemsets* de C_3 são armazenados na estrutura árvore *hash*, cujo percurso a ser seguido até a posição correta para armazenamento do *itemset* é obtido pela aplicação da função *hash* aos itens que o compõem.

A construção da árvore é inicializada com a criação de um único nó que, ao mesmo tempo, desempenha as funções de raiz e folha. Inicialmente os três primeiros *itemsets* de C_3, (1,2,4), (1,2,5) e (1,3,6), são armazenados no nó raiz criado até então, uma vez que o limitante máximo por nó folha foi estabelecido como 3.

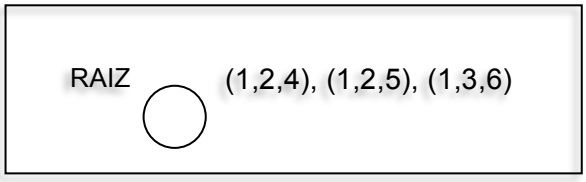


Figura 2.2 – Inserção inicial dos três primeiros *itemsets* na raiz da árvore *hash*.

Quando o próximo *itemset* (1,4,5) for adicionado à árvore, como a raiz (profundidade 1) já armazena o número máximo estabelecido de *itemsets*, um procedimento utilizando a função *hash* é acionado. Considerando a aplicação da função *hash* ao primeiro *itemset* (1,2,4), tem-se: *hash*((1,2,4),1) → (1,2,4)[1] mod 3 = 1 mod 3 = 1.

O processo tem continuidade com a criação de um nó folha, filho do nó corrente e ligado a ele por um ramo rotulado 1; o *itemset* (1,2,4) é então transferido para esse nó folha. Quando a função *hash* é aplicada ao *itemset* (1,2,5), tem-se: *hash*((1,2,5),1) → (1,2,5)[1] mod 3 = 1 mod 3 = 1 e, portanto, o *itemset* (1,2,5) é transferido para o mesmo nó folha criado anteriormente.

Considerando agora o *itemset* (1,3,6), o mesmo processo é repetido e tem-se: *hash*((1,3,6),1) → (1,3,6)[1] mod 3 = 1 mod 3 = 1, o que faz com que tal *itemset* se junte aos dois anteriores, no nó folha recém criado, como mostra a Figura 2.3.

O processo tem continuidade considerando agora o *itemset* (1,4,5). Quando a função *hash* é aplicada à (1,4,5), tem-se: *hash*((1,4,5),1) → (1,4,5)[1] mod 3 = 1 mod 3 = 1 e, portanto, o *itemset* (1,4,5) deveria ser transferido para o mesmo nó folha criado anteriormente. No entanto, como tal nó está saturado (tem já 3 *itemsets* armazenados), todo o processo é repetido novamente, agora considerando a profundidade do nó corrente i.e., a profundidade do nó folha recentemente criado, que é 2.

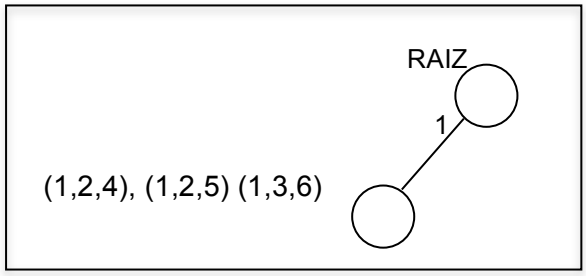


Figura 2.3 – Criação de um novo nó folha (ligado a seu pai via ramo rotulado 1) e transferência dos *itemsets* da raiz para o nó folha criado.

O procedimento *hash* é novamente acionado, agora considerando o nó folha recém criado, com profundidade 2. Considerando a aplicação da função *hash* ao primeiro *itemset* (1,2,4), tem-se: *hash*((1,2,4),2) → (1,2,4)[2] mod 3 = 2 mod 3 = 2. É então criado um nó folha, com profundidade 3, filho do nó corrente e ligado a ele por um ramo rotulado 2, como mostra a Figura 2.4.

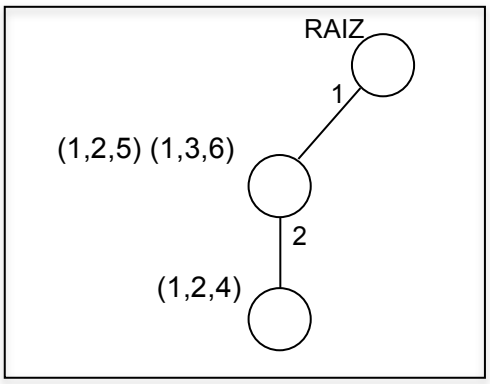


Figura 2.4 – Criação de um novo nó folha (com profundidade 3) e transferência do primeiro itemset (1,2,4), armazenado em seu pai, para ele.

Quando a função *hash* é aplicada ao *itemset* (1,2,5), tem-se: *hash*((1,2,5),2) → (1,2,5)[2] mod 3 = 2 mod 3 = 2 e, portanto, o *itemset* (1,2,5) é transferido para o último nó folha criado. Considerando agora o *itemset* (1,3,6), o mesmo processo é repetido e tem-se: *hash*((1,3,6),2) → (1,3,6)[2] mod 3 = 3 mod 3 = 0, o que provoca a criação de um novo nó folha, irmão do último criado, ligado ao nó pai por um ramo rotulado 0. O *itemset* (1,3,6) é então transferido ao novo nó folha criado, como mostra a Figura 2.5.

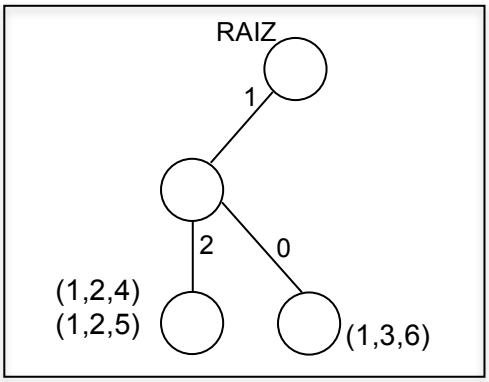


Figura 2.5 – Criação de um novo nó folha (com profundidade 3) e transferência do itemset (1,3,6,) armazenado em seu pai, para ele.

O processo tem continuidade considerando agora o *itemset* (1,4,5). Quando a função *hash* é aplicada à (1,4,5), tem-se: *hash*((1,4,5),2) → (1,4,5)[2] mod 3 = 4 mod 3 = 1 e, portanto, a 'descida' na árvore *hash* para armazenar o *itemset* (1,4,5) é via ramo rotulado 1. Como não existe, ainda, um ramo de rótulo 1 ele é criado e, então,

é criado um nó folha, com profundidade 3, filho do nó corrente, como mostra a Figura 2.6. Portanto, o *itemset* (1,4,5) é transferido para o novo nó criado.

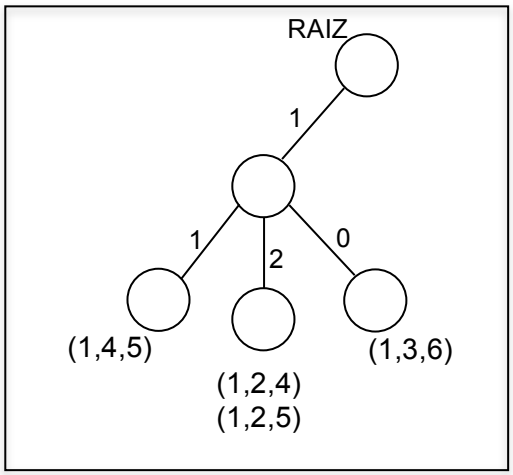


Figura 2.6 – Criação de um novo nó folha (com profundidade 3) e armazenamento do *itemset* (1,4,5) no nó folha criado.

O processo descrito continua até que todos os candidatos contidos em C_3 tenham sido armazenados na estrutura de árvore *hash*. A Figura 2.7 mostra uma representação pictórica da árvore *hash* criada após o término do processo de geração e armazenagem do conjunto C_3.

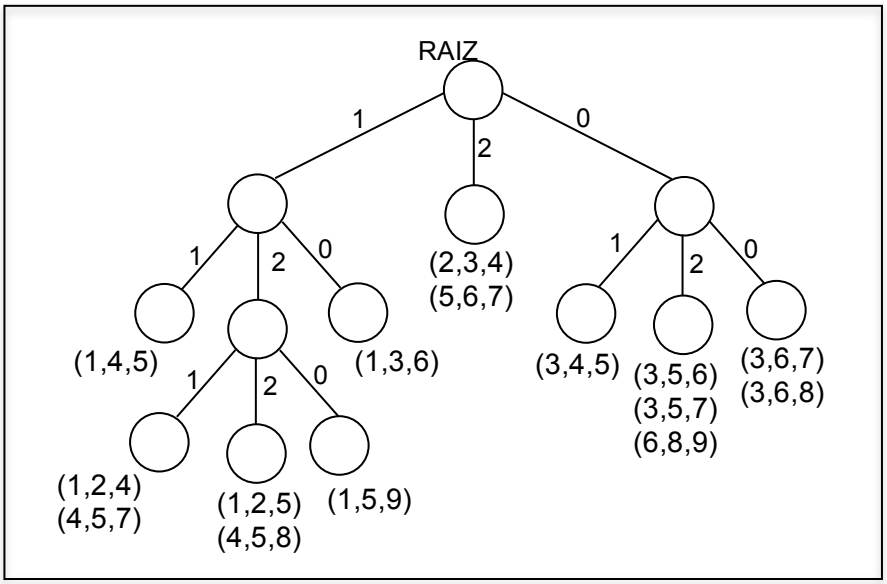


Figura 2.7 – Árvore *hash* que armazena o conjunto C_3 = {(1,2,4) (1,2,5) (1,3,6) (1,4,5) (1,5,9) (2,3,4) (3,4,5) (3,5,6) (3,5,7) (3,6,7) (3,6,8) (4,5,7) (4,5,8) (5,6,7) (6,8,9)}. Adaptada de (TAN et al., 2005).

Após armazenados na *árvore hash*, é implementado um processo de validação dos candidatos, utilizando as transações da BD. Cada vez que um *itemset* candidato é encontrado em uma transação da BD, sua contagem de suporte é incrementada.

Considere que a primeira transação t ∈ BD, seja t = {1,2,3,5,6}. O algoritmo considera todos os possíveis *itemsets*, de tamanho k (nesse exemplo, k = 3), que podem ser obtidos a partir de t. Para o exemplo, os possíveis *itemsets* são: (1,2,3), (1,2,5), (1,2,6), (1,3,5), (1,3,6), (1,5,6), (2,3,5), (2,3,6), (2,5,6) e (3,5,6). Cada um dos *itemsets* é então selecionado e uma busca na árvore *hash* é executada para verificar se a árvore armazena um *itemset* igual ao *itemset* selecionado. A mesma função *hash* utilizada na geração da árvore é aplicada ao *itemset* de t em cada nível da árvore para determinar o percurso que a busca na árvore deve seguir.

O procedimento de validação dos candidatos de C_3 é iniciado selecionando o primeiro *itemset* que pode ser gerado a partir de t ou seja, o (1,2,3). A busca pelo *itemset* é iniciada pela raiz da árvore *hash* e como a raiz não é um nó folha a função *hash* é aplicada ao *itemset* (1,2,3). Tem-se: *hash*((1,2,3),1) → (1,2,3)[1] mod 3 = 1 mod 3 = 1 e, portanto, o percurso a ser seguido para a busca do *itemset* na *árvore has*h é pelo ramo de rótulo 1. O nó ligado à raiz por esse ramo também não é um nó folha e, portanto, o processo continua com a aplicação da função *hash* a (1,2,3) porém agora, no nível 2 da árvore. Sendo assim, *hash*((1,2,3),2) → (1,2,3)[2] mod 3 = 2 mod 3 = 2 e, portanto, o percurso a ser seguido na busca é pelo ramo de rótulo 2.

Como o nó alcançado também é um nó interno, a função *hash* deve ser aplicada novamente a (1,2,3), considerando agora a profundidade do nó corrente, i.e., 3. Aplicando a função *hash* à (1,2,3), tem-se: *hash*((1,2,3),3) → (1,2,3)[3] mod 3 = 3 mod 3 = 0 e, portanto, a 'descida' na árvore é via ramo de rótulo 0.

O nó corrente, atingido pela busca, é um nó folha de profundidade 4 e, portanto, armazena um conjunto de *itemsets* candidatos que, no caso, contem apenas um *itemset*, i.e., (1,5,9). Como o *itemset* (1,2,3) não pertence ao conjunto unitário armazenado no nó folha, a contagem de suporte do *itemset* candidato não é incrementada.

A Figura 2.8 mostra o processo de busca do *itemset* seguinte, i.e., (1,2,5), na *árvore hash*. Note que (1,2,5) pertence ao conjunto de *itemsets* armazenado no nó folha alcançado pela busca, i.e., o *itemset* candidato a ser frequente faz parte da transação t e, portanto, a contagem de suporte do candidato (1,2,5) é incrementada em 1.

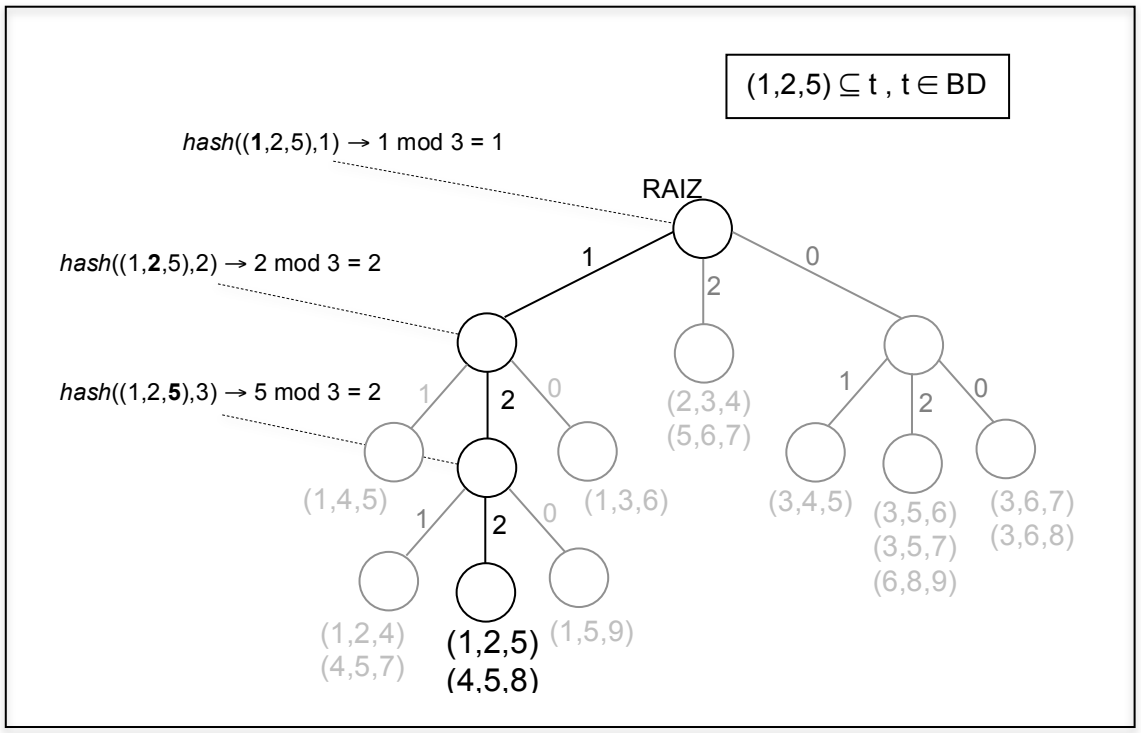


Figura 2.8 – Busca pelo *itemset* (1,2,5) na *árvore hash* que armazena o conjunto C_3 = {(1,2,4) (1,2,5) (1,3,6) (1,4,5) (1,5,9) (2,3,4) (3,4,5) (3,5,6) (3,5,7) (3,6,7) (3,6,8) (4,5,7) (4,5,8) (5,6,7) (6,8,9)}.

2.4 Geração de Regras de Associação

A essência da mineração de dados, em geral, pode ser caracterizada como extrair o maior número possível de informações úteis de uma base de dados. Encontrar regras de associação representativas é um dos objetivos do algoritmo Apriori (AGRAWAL; SRIKANT, 1994). Inferir tais regras significa encontrar uma relação existente entre os itens que definem um *itemset* (transação) de uma base de dados. Por exemplo, afirmar que existe um padrão que estabelece que clientes de uma papelaria que compram lápis, quase sempre também compram borracha. Em outro contexto, o recurso *auto-completar*, utilizado em muitos formulários virtuais utiliza-se desse tipo de regras para sugerir quais palavras o usuário quer digitar, baseando-se apenas nos primeiros caracteres sendo digitados pelo usuário.

Para inferir regras eficientes, é inserido no contexto um novo conceito, o *valor de confiança*, descrito pela Eq (2.1), que estabelece a contribuição que existe em uma possível regra de associação inferida, para um padrão na base de dados.

$$\text{Conf}(x \Rightarrow y) = \text{frequência de x e y juntos / frequência de x} \qquad (2.1)$$

O valor de confiança representa a medida da frequência com que a regra associada aos itens de um *itemset* é encontrada na BD. Ou seja, analisando as transações, em quantas delas é possível validar a regra inferida.

Considerando uma regra genérica R: $x \Rightarrow y$, se c = Conf(R), o valor de confiança, c, de R significa que y ocorre nas relações em que x ocorre com confiança c. Considerando uma BD, que contém informação sobre os itens comprados em um supermercado, sabe-se que os itens de um *itemset* frequente aparecem em muitas transações, as regras de associação refletem a relação existente entre tais itens do *itemset* frequente, ou seja, quais produtos (itens) da compra têm maior influência para que o cliente compre os demais produtos da transação.

Segundo Agrawal e Srikant (1994), o problema da descoberta de todas as regras de associação, pode ser decomposto em dois subproblemas: (1) encontrar *itemsets*, Y, na BD que ocorrem com alta frequência e (2) usar os *itemsets* encontrados em (1) para gerar regras de associação entre os itens que os compõem. Ainda segundo Agrawal e Srikant (1994), regras de associação são geradas a partir da identificação de todos os subconjuntos X, não vazios, de cada *itemset* frequente, Y, de tamanho k. Para cada subconjunto X de Y, é formada uma regra do tipo R: $X \Rightarrow (Y - X)$, em que seu antecedente é constituído por um subconjunto X de Y e o consequente da regra será composto pelos itens restantes, Y – X.

No processo de geração de regras, efetuado pelo algoritmo Apriori, não é preciso revisitar a BD devido ao fato de que os valores de suporte de todos os *itemsets* frequentes identificados já estão calculados. Inicialmente o Apriori assume que os subconjuntos X tem tamanho k – 1 e, portanto, tais regras são constituídas por consequentes com um único item ($|Y - X| = 1$); a regra é gerada quando possuir valor de confiança superior a um limitante pré-estabelecido (*min_conf*). Em seguida, o algoritmo decrementa o tamanho de X e gera novas regras de associação. O procedimento segue, recursivamente, até que $|X| = 1$, condição de parada do procedimento. O processo de geração de regras de associação entre *itemsets* frequentes, *gera_regras* (ver Algoritmo 2.4), foi elaborado a baseando-se no processo implementado pelo Apriori e da descrição apresentada em (GOETHALS, 2003).

```
Procedure gera_regras(All_L)
 Entrada: All_L = Conjunto com todos os itemsets frequentes, identificados pelo Apriori;
          min_conf = parâmetro de valor mínimo para considerar a regra forte.
 Saída: R = Conjunto com todas as regras de associação geradas;
1. begin
2. R ← ∅
3.   for i ← 1 to |All_L| do
4.   begin
5.     e_i ← get_itemset(All_L,i)          {seleciona o itemset na posição i em All_L}
6.     verifica_regras(R, e_i, |e_i|, min_conf)     {avalia as regras geradas do itemset}
7.   end
8. return R

Procedure verifica_regra(R, e_i, k, min_conf)
 Entrada: R = Conjunto com todas as regras de associação já geradas;
          e_i = itemset frequente utilizado para gerar novas regras;
          k = tamanho do itemset analisado; min_conf = parâmetro de valor
          mínimo para considerer a regra forte.
1. begin
2. sup_e_i ← calcula_suporte(e_i)
3. E_k-1 ← todos_itemsets(e_i, k-1)   {Identifica todos os itemsets de tamanho k-1
4.     for j ← 1 to |E_k-1| do          que podem ser gerados a partir de e_i}
5.     begin
6.       it_j ← get_itemset(E_k-1,j)      {seleciona o itemset na posição j em E_k-1}
7.       conf ← sup_e_i / calcula_suporte(it_j)
8.       if(conf ≥ min_conf) then
9.       begin
10.        R ← R ∪ regra(it_j, e_i, conf)
11.        if(|e_i| > 1) then
12.          verifica_regra(R, it_j, k-1, min_conf)
13.      end
14.    end
15. end
```

Algoritmo 2.4 – Pseudocódigo do procedimento para geração de regras de associação baseadas nos *itemsets* frequentes identificados pelo algoritmo Apriori.

2.5 Um Exemplo de Uso do Apriori

No que segue está descrito um exemplo do uso do algoritmo Apriori na extração de regras de associação. Considere um supermercado hipotético que disponibiliza para venda apenas sete produtos: açúcar (a), café (c), leite (l), manteiga (m), pão (p), queijo (q) e presunto (r). Considere ainda a pequena base de dados BD contendo dez transações, apresentada na Tabela 2.1. Nela cada transação representa um conjunto de produtos de supermercado adquiridos por um determinado freguês.

O conjunto dos produtos disponibilizados à venda pode ser considerado o conjunto de itens I = {a, c, l, m, p, q, r} e uma transação T na BD, um subconjunto de I ou seja, um *itemset* (Definição 1.1). Considere ainda que o limite inferior (*min_sup*) de frequência definido para o exemplo é 40%, ou seja, 2/5 = 0,4.

Tabela 2.1 – Base de dados BD de transações (vendas de um supermercado).

Transações	Produtos	Transações	Produtos
T_1	a, c, l, m, p, q	T_6	a, c, l, p
T_2	a, c, l, m, p, r	T_7	q, r
T_3	a, m, p	T_8	a, c, l, m
T_4	l, m, p	T_9	c, l, p
T_5	a, c, l, m, p	T_{10}	a, c, l, m, p, q, r

Seguindo a notação e o algoritmo, o primeiro passo do Apriori é identificar os itens frequentes nas transações da BD para, então, gerar o conjunto inicial de itens frequentes, L_1. A Tabela 2.2 mostra a contagem de suporte (frequência de ocorrência) de cada um dos itens que comparecem nos *itemsets* que definem a BD. Como pode ser observado na Tabela 2.2, não são todos os itens que têm um valor de suporte que os caracterizem como frequentes e o conjunto L_1 é criado como L_1 = {a, c, l, m, p}.

Tabela 2.2 – Contagem de suporte dos itens da BD para formar o conjunto L_1. Todos os itens que comparecem em 40% ou mais das transações da BD são considerados frequentes.

Item	Suporte	Item	Suporte
a	7/10 = 0,7	p	8/10 = 0,8
c	7/10 = 0,7	q	3/10 = 0,3
l	8/10 = 0,8	r	3/10 = 0,3
m	7/10 = 0,7	–	–

Na execução do algoritmo o contador da iteração, k, é inicializado com 2 e o primeiro conjunto de candidatos $C_{k=2}$ é então gerado. Como mostra a Tabela 2.3, o resultado do processo de junção entre o conjunto de itens frequentes L_1 com ele próprio gera *itemsets* de tamanho 2, chamados pré-candidatos (passo 6 do Algoritmo 2.2).

Tabela 2.3 – Pré-candidatos gerados a partir da junção de L_1 com ele mesmo (k = 1).

Itemsets de L_1	*Itemsets* de L_1	Candidatos gerados (C_2)
a	c	(a, c)
a	l	(a, l)
a	m	(a, m)
a	p	(a, p)
c	l	(c, l)
c	m	(c, m)
c	p	(c, p)
l	m	(l, m)
l	p	(l, p)
m	p	(m, p)

Na Tabela 2.3 a terceira coluna apresenta todas as combinações geradas que respeitam a condição do processo de geração de candidatos, descrita no Algoritmo 2.2. São descartadas as combinações que não contemplam a Definição 2.1, tais como: (a) e (a), (c) e (a), (c) e (c), (l) e (a), (l) e (c), (l) e (l), (m) e (a), (m) e (c), (m) e (l), (m) e (m), (p) e (a), (p) e (c), (p) e (l), (p) e (m) e, finalmente, (p) e (p).

Após a geração dos pré-candidatos a *itemsets* frequentes, o procedimento *gera_itemsets_candidatos* realiza mais uma poda, a fim de considerar apenas as combinações em que satisfazem a propriedade Apriori i.e.: dado um pré-candidato, todos subconjuntos que podem ser gerados com seus itens devem, também, ser pertencentes ao conjunto de *itemsets* frequentes da passagem anterior (L_1), utilizado para gerar os atuais candidatos. Como mostra a Tabela 2.3, todos os pré-candidatos gerados possuem dois itens que os compõem (2-*itemsets*, k = 2) e, em todos eles, os itens que os compõem também fazem parte do conjunto L_1, validando a condição do *gera_itemsets_candidatos*. Portanto, todas as combinações apresentadas na Tabela 2.3, são efetivamente consideradas candidatas a *itemsets* frequentes.

Os *itemsets* candidatos que não foram eliminados na etapa de poda executada pelo procedimento *gera_itemsets_candidatos*, compõem o conjunto de candidatos C_2 = {(a, c),(a, l),(a, m),(a, p),(c, l),(c, m),(c, p),(l, m),(l, p),(m, p)}. A Tabela 2.4 mostra cada candidato a *itemset* de C_2, seu respectivo valor de suporte e se participará (sim) ou não (não) de L_2.

Tabela 2.4 – Passagem k = 2 do Apriori. Conjunto de candidatos C_2 gerados a partir de L_1 e respectivos valores de suporte. Os frequentes compõem o conjunto L_2.

Itemsets $\in C_2$	Suporte	*Itemset* $\in L_2$?
(a, c)	6/10 = 0,6	sim
(a, l)	6/10 = 0,6	sim
(a, m)	6/10 = 0,6	sim
(a, p)	6/10 = 0,6	sim
(c, l)	7/10 = 0,7	sim
(c, m)	5/10 = 0,5	sim
(c, p)	6/10 = 0,6	sim
(l, m)	6/10 = 0,6	sim
(l, p)	7/10 = 0,7	sim
(m, p)	6/10 = 0,6	sim

Na próxima iteração (k = 3) inicialmente os elementos do conjunto C_3, caracterizados como pré-candidatos, são gerados por meio do mesmo processo anterior, a partir do conjunto de *itemsets* frequentes da passagem anterior (k = 2) ou seja, a partir de L_2 = {(a, c), (a, l), (a, m), (a, p), (c, l), (c, m), (c, p), (l, m), (l, p), (m, p)} para, em seguida, os *itemsets* frequentes de C_3 serem identificados para compor o L_3.

Para gerar os pré-candidatos, o procedimento *gera_itemsets_candidatos*, (ver Algoritmo 2.2), seleciona cada um dos *itemsets* frequentes de L_2 e compara, um a um, com todos os demais *itemsets* de L_2, a fim de encontrar pares que possam ser combinados para gerar um *itemset* (tamanho k+1). Como já descrito, a junção mantém a ordem lexicográfica entre os itens dos *itemsets*. Nesse exemplo, inicialmente, o primeiro *itemset* de L_2, e_1 = (a, c), é selecionado e todos os demais *itemsets* que respeitam a condição são utilizados para estender e_1 em um item. Os itemsets que podem estender e_1 são: (a, l), (a, m) e (a, p) uma vez que todos possuem seus k-2 primeiros itens iguais, i.e., a = a e o (k-1)-item de e_1 é lexicograficamente menor que o (k-1)-item de cada um dos *itemsets* selecionados. Assim, o *itemset* e_1 pode ser estendido com o último elemento de cada um dos *itemsets* selecionados, gerando os candidatos (a, c, l), (a, c, m) e (a, c, p).

A Tabela 2.5 apresenta todas as combinações possíveis para os *itemsets* de L_2. Na primeira coluna são listados os *itemsets* de L_2 que serão combinados e na segunda os *itemsets* que possuem seus k-2 primeiros itens iguais ao itemset da primeira coluna. Na terceira é feita a comparação entre os últimos itens dos *itemsets* selecionados, de forma que respeitem a ordem lexicográfica. Na quarta coluna são apresentadas as combinações resultantes da extensão do *itemset* da

primeira coluna com o ultimo item do *itemset* na segunda coluna, gerando um pré-candidato de tamanho k = 3.

Tabela 2.5 – Passagem k = 3. Junção do conjunto de *itemsets* frequentes L_2, com ele mesmo, gerar os pré-candidatos que possivelmente irão compor C_3.

Itemsets $\in L_2$	***Itemsets*** **com (k - 2) primeiros itens iguais**	**Comparação do último item**	**Pré-candidatos gerados (C_3)**
(a, c)	(a, l)	c < l	(a, c, l)
(a, c)	(a, m)	c < m	(a, c, m)
(a, c)	(a, p)	c < p	(a, c, p)
(a, l)	(a, m)	l < m	(a, l, m)
(a, l)	(a, p)	l < p	(a, l, p)
(a, m)	(a, p)	m < p	(a, m, p)
(c, l)	(c, m)	l < m	(c, l, m)
(c, l)	(c, p)	l < p	(c, l, p)
(c, m)	(c, p)	m < p	(c, m, p)
(l, m)	(l, p)	m < p	(l, m, p)

Os *itemsets* (a, p), (c, p), (l, p) e (m, p) de L_2 não são listados na primeira coluna da Tabela 2.5 uma vez que não há nenhum outro *itemset* (x, y) $\in L_2$ que compartilhe com qualquer um deles os k-2 primeiros itens ou, então, respeitem a ordem lexicográfica em seus últimos (k-1) itens. Os únicos pré-candidatos gerados a partir de L_2 por meio da operação de junção utilizando o *itemset* (a, c) $\in L_2$, por exemplo, são: (a, c, l), (a, c, m) e (a, c, p).

Todos os pré-candidatos a *itemsets* frequentes voltam a se apresentadas na Tabela 2.6, junto com o respectivo suporte. Para cada um dos pré-candidatos gerados, todos os seus subconjuntos com k-1 itens são validados quanto à propriedade Apriori. Caso algum de seus subconjuntos não faça parte do conjunto L_2, o pré-candidato é eliminado, senão é considerado como um candidato.

Note na segunda coluna da Tabela 2.6 que cada um dos pré-candidatos gerados (i.e., cada elemento de C_3) contempla a condição de ter todos os seus subconjuntos de tamanho k-1 (2) presentes no conjunto L_2 e, portanto são considerados candidatos a terem o seu suporte calculado. A terceira coluna da tabela evidencia que todos os *itemsets* de C_3 têm o respectivo suporte ≥ min_sup previamente estabelecido e, portanto, todos eles definem o conjunto de *itemsets*

frequentes ou seja, L_3 = {(a, c, l), (a, c, m), (a, c, p), (a, l, m), (a, l, p), (a, m, p), (c, l, m), (c, l, p), (c, m, p), (l, m, p)}.

Tabela 2.6 – Passagem k = 3 do Apriori. Conjunto de candidatos C_3 gerados a partir de L_2 e respectivos valores de suporte. Os frequentes compõem o conjunto L_3.

Itemsets ∈ C_3	Subconjuntos ∈ L_2?	Suporte	*Itemset* ∈L_3?
(a, c, l)	sim	6/10 = 0,6	sim
(a, c, m)	sim	5/10 = 0,5	sim
(a, c, p)	sim	5/10 = 0,5	sim
(a, l, m)	sim	5/10 = 0,5	sim
(a, l, p)	sim	5/10 = 0,5	sim
(a, m, p)	sim	5/10 = 0,5	sim
(c, l, m)	sim	5/10 = 0,5	sim
(c, l, p)	sim	6/10 = 0,6	sim
(c, m, p)	sim	4/10 = 0,4	sim
(l, m, p)	sim	5/10 = 0,5	sim

O algoritmo continua a sua execução pois a condição de parada ($L_k = \varnothing$) não foi verificada e, então prossegue incrementando o valor de k (que passa a ser 4) e utiliza o conjunto L_3 para gerar o conjunto de *itemsets* candidatos com tamanho 4, i.e., C_4. Na sequência cada *itemset* de C_4 é verificado quanto a ter todos os seus subconjuntos de tamanho 3 em L_3 e, aqueles que tiverem (no exemplo, todos eles satisfazem a condição), têm seu valor de suporte calculado.

A Tabela 2.7 apresenta todos os *itemsets* de C_4 com seus respectivos valores de suporte. Todos os candidatos contidos em C_4 são gerados pelo mesmo processo de junção descrito nas passagens anteriores, que também realiza a poda desses candidatos (condição de todos os subconjuntos de cada *itemset* serem elementos do conjunto *de itemsets* frequentes anterior). Também mostra o conjunto L_4, resultado da quarta iteração do algoritmo Apriori.

Tabela 2.7 – Conjunto de candidatos C_4 gerados a partir de L_3 com seus respectivos valores de suporte. Os frequentes compõem o conjunto L_4.

Itemsets ∈ C_4	Subconjuntos ∈ L_3?	Suporte	*Itemsets* ∈ L_4?
(a, c, l, m)	sim	5/10 = 0,5	sim
(a, c, l, p)	sim	5/10 = 0,5	sim
(a, c, m, p)	sim	4/10 = 0,4	sim
(a, l, m, p)	sim	4/10 = 0,4	sim
(c, l, m, p)	sim	4/10 = 0,4	sim

O algoritmo continua e incrementa o o valor de k para 5 e, com base no L_4 gera o conjunto de candidatos C_5 e, então, o conjunto L_5, de *itemsets* frequentes de maneira similar às anteriores. A Tabela 2.8 mostra o único *itemset* candidato gerado

pelo procedimento *apriori_gen* e, também, apresenta o seu valor de suporte, validando-o como um *itemset* frequente da BD da passagem k = 5 do algoritmo Apriori.

Tabela 2.8 – Conjunto de candidatos C_5, contendo apenas um *itemset* candidato, gerado a partir de L_4. Também seu respectivo valor de suporte que o valida como o único *itemset* frequente do conjunto L_5.

Itemsets $\in C_5$	**Subconjuntos** $\in L_4$**?**	**Suporte**	***Itemsets*** $\in L_5$**?**
(a, c, l, m, p)	sim	4/10 = 0,4	sim

Devido ao fato do conjunto L_5 ser unitário, não é possível gerar novos candidatos e, portanto, $C_6 = \varnothing$ e, consequentemente, $L_6 = \varnothing$. A condição de parada do algoritmo é então satisfeita e, portanto, termina a busca por *itemsets* frequentes na BD.

A próxima etapa (geração de regras de associação) é considerada, por alguns autores que subsidiam este trabalho, um procedimento realizado após o Apriori ter terminado e identificado o conjunto de *itemsets* frequentes. Tal etapa é tratada a seguir. Considerando todos os conjuntos de *itemsets* frequentes, em cada passagem k do algoritmo Apriori, é possível criar regras de associação existentes entre os *itemsets* frequentes. Para esse exemplo, foi selecionado o último *itemset* frequente identificado, por ser o maior entre todos.

Assim como descrito na Seção 2.4, para cada *itemset* frequente Y, é construída uma regra de acordo com a seguinte sintaxe: R: $X \Rightarrow (Y - X)$, na qual o antecedente é um *itemset* $X \subseteq Y$ e o consequente o *itemset* $Y - X$.

Dessa forma, para o exemplo em questão, o processo de geração de regras de associação para o *itemset* Y = (a, c, l, m, p) se inicia com a regra (R_1) que tem como antecedente X = (a, c, l, m), de tamanho k-1 e como consequente Y – X = (p).

O valor de confiança de R_1 é então calculado (razão entre o valor do suporte de Y pelo valor do suporte de X), ou seja, 0,4 / 0,5 = 0,8. Tal valor é então comparado ao parâmetro pré-definido (*min_conf*) e caso o valor de confiança da regra supere *min_conf*, a regra é gerada e o tamanho de X é decrementado em um item para uma nova regra ser gerada e avaliada. O processo se repete enquanto o procedimento não encontrar uma regra com nenhum item na parte antecedente ou que possua valor de confiança abaixo de *min_conf*. As regras inferidas para o único

itemset do conjunto L_5 = {(a, c, l, m, p)}, podem ser vistas na Tabela 2.9, considerando *min_conf* = 0,6.

Tabela 2.9 – Conjunto de regras de associação geradas a partir do único *itemset* frequente pertencente a L_5 = (a, c, l, m, p). Também, seus respectivos valores de confiança e validação quando a regra de associação for considerada forte.

Regra	Confiança	Regra considerada?
a, c, l, m ⇒ p	0,4 / 0,5 = 0,8	Sim
a, c, l ⇒ m, p	0,4 / 0,6 = 0,66	Sim
a, c ⇒ l, m , p	0,4 / 0,6 = 0,66	Sim
a ⇒ c, l, m , p	0,4 / 0,7 = 0,57	Não

Como pode ser visto na Tabela 2.9, as regras de associação consideradas *fortes* são as que apresentam um valor de confiança superior ao limite mínimo pré-definido que, para esse exemplo, foi estabelecido como 0,6 (que representa o valor 6/10 ou seja, a cada 10 transações analisadas da BD que apresentam o *itemset* da regra, pelo menos seis delas têm que validar a regra). É importante, entretanto, informar que as combinações apresentadas na Tabela 2.9 foram geradas a partir de L_5, porém o processo de geração de regras de associação, implementado pelo algoritmo, utiliza-se de todos os conjuntos gerados, i.e., L_2, L_3 e L_4, também.

Capítulo 3

A PROPOSTA MEMISP (MEMORY INDEXING FOR SEQUENTIAL PATTERN MINING)

O conceito de sequência bem como os conceitos subjacentes a ele são fundamentais para a representação e mineração de padrões, particularmente em situações que envolvem temporalidade. Uma sequência pode ser informalmente abordada como uma lista de elementos em uma determinada ordem. A sequência dos números pares menores (ou igual) a 10, por exemplo, é uma sequência finita que, na notação empregada nesse trabalho é representada por <(2)(4)(6)(8)(10)>.

3.1 Notação e Conceitos Básicos

No que segue, inicialmente são estabelecidas definições e correspondente notações dos conceitos iniciais, descritos por Lin e Lee (2002) e Han et al. (2000), que subsidiam um novo algoritmo, o MEMISP (MEMory Indexing for Sequential Pattern mining), que implementa uma maneira rápida para a mineração de padrões sequenciais.

Se BD representa uma base de dados de sequências, cada sequência é representada como uma sequência de dados (nesse formalismo uma BD de sequências é considerada um conjunto no qual todo elemento é uma sequência de dados).

A seguir, a definição 1.1 é revisitada (reescrita como Definição 3.1) com vistas a facilitar a compreensão das demais definições estabelecidas neste capítulo.

Definição 3.1

(1) Seja um conjunto de *itens* notado por $I=\{i_1, i_2, ..., i_N\}$.

(2) Um *itemset e* é um subconjunto do conjunto de itens, i.e., $e \subseteq I$.

(3) Uma *sequência* s, escrita da forma $s=<e_1e_2...e_n>$, é uma lista ordenada na qual e_j é um *itemset*, i.e., $e_j \subseteq I$, j = 1, ..., n.

(4) Cada *itemset* e_j (j = 1,..., n) da sequência é escrito como $e_j = (i_{j_1}, i_{j_2}, ..., i_{j_m})$, em que $i_{j_k} \in I$ (k=1, ..., m). A notação $|e_j|$ representa o número de itens que definem o *itemset* e_j e, portanto, para o *itemset* $e_j = (i_{j_1}, i_{j_2}, ..., i_{j_m})$, $|e_j| = m$.

(5) Um item $i_k \in I$ (k = 1, ..., N) pode ocorrer, no máximo, uma vez em um particular *itemset* de uma determinada sequência. Um mesmo item i_k pode, entretanto, ocorrer múltiplas vezes em uma sequência, uma vez que pode ocorrer em vários dos *itemsets* que definem a sequência.

(6) Os *itemsets* e_i, $e_j \subseteq I$ (i, j = 1, ..., n) em uma sequência não necessariamente compartilham os mesmos itens e não necessariamente têm a mesma cardinalidade. Pode pois acontecer que: $e_i \cap e_j = \varnothing$, $i \neq j$ ou $|e_i| \neq |e_j|$, $i \neq j$.

(7) Sem perda de generalidade, assume-se que os itens que definem um *itemset* estão em ordem lexicográfica;

(8) Note que, apesar de ser um conjunto, a notação adotada para representar um *itemset* e_j é $e_j = (i_{j_1}, i_{j_2}, ..., i_{j_n})$, em virtude de (7).

Definição 3.2 O *tamanho de uma sequência* $s=<e_1e_2...e_n>$, notado por |s|, é dado pelo número total de itens que comparecem nos *itemsets* e_i (i = 1, ..., n) da sequência. Portanto $|s| = \sum_{i=1}^{n} |e_i|$. Referências a sequências usualmente empregam o seu tamanho, ou seja, uma sequência com tamanho k é referida como uma k-sequência.

Exemplo 3.1 Considere I = {a, b, c, d, e, f, g, h} um conjunto de itens e considere a sequência de *itemsets* $s=<e_1e_2e_3e_4>$, em que e_1=(a,c), e_2=(d), e_3=(a,c,e) e e_4=(b,f) com os itens ordenados lexicograficamente em cada *itemset*. Os itens a e

c, que participam dos *itemsets* e_1 e e_3, se repetem na sequência, mas não em um mesmo *itemset.*

A sequência s expandida é pois s=<(a,c)(d)(a,c,e)(b,f)>, que pode ser referenciada como uma 8-sequência. Nela, $e_1 \cap e_2 = e_1 \cap e_4 = \varnothing$ enquanto que $e_1 \cap e_3$ = (a,c) e $e_2 \cap e_4$ = (f). Também, $|e_1| = |e_4| = 2$, $|e_2| = 1$ e $|e_3| = 3$. Tem-se pois que $|s| = \sum_{i=1}^{4} |e_i|$=2+1+3+2=8.

Definição 3.3 Considere duas sequências $s_1 = <a_1a_2...a_n>$ e $s_2 = <b_1b_2...b_m>$ tal que $|s_1| \leq |s_2|$ (implicando $n \leq m$). A sequência s_1 é uma *subsequência* de s_2, notada por $s_1 \sqsubseteq s_2$ se os itens que compõem cada um dos elementos de s_1 estiverem presentes também em s_2, i.e., se existirem n números inteiros i_1, i_2, ..., i_n tal que $1 \leq i_1 < i_2 <. .. < i_n < m$ e $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, ..., $a_n \subseteq b_{i_n}$. Neste caso a sequência s_2 é dita ser *supersequência* de s_1.

Exemplo 3.2 Considere duas sequências: (I) sequência $s_1 = <a_1a_2a_3a_4> =$ <(e)(b,c)(d)(c)>, ou seja, a_1 = (e), a_2 = (b,c), a_3 = (d) e a_4 = (c) e (II) a sequência $s_2 = <b_1b_2b_3b_4b_5> =$ <(e,f)(a,b,c)(d,f)(d)(c)> ou seja, b_1 = (e,f), b_2 = (a,b,c), b_3 = (d,f), b_4 = (d) e b_5 = (c). Com base na Definição 3.2 sabe-se que $|s_1| = 5$ e $|s_2| = 9$, fazendo com que a sequência s_1 seja caracterizada como uma 5-sequência e a s_2 como uma 9–sequência.

Nota-se ainda que s_1 é uma subsequência de s_2 ($s_1 \sqsubseteq s_2$) pois cada um dos *itemsets* de s_1 participa de algum *itemset* de s_2, de maneira ordenada, como estabelece a Definição 3.3.

Assim, pode se observar que o item (i.e., e) do *itemset* $a_1 \in s_1$ é encontrado no *itemset* $b_1 \in s_2$, ou seja, $a_1 \subseteq b_1$. Da mesma forma é possível observar que $a_2 \subseteq b_2$, $a_3 \subseteq b_3$ e $a_4 \subseteq b_5$. Assim, sendo s_1 é uma *subsequência* de s_2, o que torna s_2 uma *supersequência* de s_1. A Tabela 3.1 apresenta diversas situações que exemplificam os conceitos de subsequência e de *supersequência*.

O procedimento descrito em Algoritmo 3.1 faz uso da Definição 3.3 para verificar se, dadas duas sequências s_1 e s_2, $s_1 \sqsubseteq s_2$. Em linhas gerais, caso s_1 seja subsequência de s_2, o algoritmo retorna, para cada *itemset e* de s_1 o correspondente

índice do *itemset* em s_2 no qual *e* comparece. O algoritmo retorna, pois, um vetor de números inteiros com dimensão igual ao número de *itemsets* de s_1.

Tabela 3.1 – Exemplos de uso do Algoritmo 3.1 para a verificação de $s_1 \sqsubseteq s_2$ considerando $s_1=<a_1, a_2, ..., a_n>$ e $s_2=<b_1, b_2, ..., b_m>$, n ≤ m.

s_1	s_2	m	n	i_1	i_2	i_3	i_4	$s_1 \sqsubseteq s_2$?
<(b)(a,e)>	<(**b**,c)(c)(**a**,c,**e**)>	3	2	1	3	–	–	S
<(b)(a,e)>	<(**b**,c,d,e)(**a**,b,c,d,**e**)>	2	2	1	2	–	–	S
<(a)(b)(c,d)(e,f,g)>	<(**a**,f)(h,m,n)(a,**b**)(e,f,g) (a,b,**c**,**d**,e)(a,h,l)(**e**,**f**,**g**,h,i)>	7	4	1	3	5	7	S
<(a)(b)(c,d)(e,f,g)>	<(**a**,f)(h,m,n)(e,f,g,h,i,j)(a,**b**) (e,f,g)(a,b,**c**,**d**,e)(a,h,l) (**e**,**f**,**g**,h,i)>	8	4	1	4	6	8	S
<(a,b)(c)(d,e,f)>	<(a)(**a**,**b**)(**c**,d)(**d**,**e**,**f**,g)>	4	3	2	3	4	–	S
<(a)(b)(a,b,c)>	<(**a**)(a,c)(a,**b**,e)>	3	3	1	3	–	–	N
<(a)(b)(a,b,c)>	<(**a,**b,c)(a,c)(a,**b**,e)>	3	3	1	3	–	–	N
<(a)(b)(c)(d,m)>	<(c)(d,f,k)(**a**,c,d)(j,l)(a,**b**,p) (h,p,r)(a,b,**c**,v)(e,f,k)(c,**d**,**m**,n)>	9	4	3	5	7	9	S
<(a,b,c,d)>	<(**a**)(b)(c)(d)>	4	1	–	–	–	–	N
<(a)(b)(c,d)>	<(**a**,b)(**b**)(**c**,**d**)>	3	3	1	2	3	–	S
<(a)(b)(c,d)>	<(**a**,b)(**b**,c,d)(e)>	3	3	1	2	–	–	N
<(r,t,u)(b,h)(a,c,d) (x,y,z)>	<(a,b)(b,c,d)(a,g,h,**r**,s,**t**,**u**,v) (**b**,e,f,g,**h**)(**a**,b,**c**,**d**,h,i,j) (a,b,g,v,**x**,**y**)>	6	4	3	4	5	–	N

Definição 3.4 Seja DB uma base de dados de sequências e s uma sequência qualquer. O *suporte da sequência* s, notado por s.sup, é dado pela razão entre o número total de sequências de dados de DB que contém s pelo total de sequências de dados no DB, como descrito pela Eq. (1)

$$\mathrm{s.sup} = \frac{|\{\mathrm{s}' \in \mathrm{DB} \,|\mathrm{s} \subset \mathrm{s}'\}|}{|\mathrm{DB}|} \qquad (1)$$

```
procedure verificaSubsequencia(s1,s2,l)
 Entrada: {s1=<a1, a2, ..., an> e s2=<b1, b2, ..., bm>} {o procedimento verifica se s1 ⊑ s2}
 Saída: l=[i1,i2,...in] {vetor dos índices de itemsets de s2, que contém itemsets ai ( 1 ≤ i ≤ n) de s1}

1  begin
2    n ← tamanho(s1)
3    m ← tamanho(s2)
4    ultimo_em_s2 ← 1
5    t  ← 1
6    while (t ≤ n && ultimo_em_s2 ≤ m) do {enquanto não percorrer toda sequência s1 e s2}
7      begin
8          while( tamanho(bultimo_em_s2) < tamanho(at) && ultimo_em_s2 < m) do
9             ultimo_em_s2 ← ultimo_em_s2 +1
10         it ← buscaIndice_em_s2(at, s2, ultimo_em_s2,m) {posição em que at está em s2}
11         ultimo_em_s2 ← it + 1
12         t ← t + 1
13     end                                          { end while }
14   if (t ≠ n+1 && ultimo_em_s2 = m + 1)  then message("s1 nao e subsequência de s")
15   else return(l)
16 end_procedure



procedure buscaIndice_em_s2(A,s2,k,m)
Entrada: A {itemset at de s1 buscado em s2}; s2 {sequência candidata a supersequência};
          k {ultimo índice de s2 que contém elemento de s1}; m {tamanho de s2}
Saída:   k {índice do itemset em s2 no qual foi encontrado o itemset A de s1.
           Na eventualidade de A não ter sido encontrado em s2, k retorna com o valor m}

1  begin
2   v ← tamanho(A)
3   while(k ≤ m) do          {enquanto todos os itemsets de s2 não foram considerados}
4      begin
5        r = tamanho(bk)
6        p ← 1
7        q ← 1
8        while( (q ≤ r) && (p ≤ v) ) do
9        begin        {Percorre todos itens de bk comparando com os itens de A}
10           if (ap = bkq)   then  {Se o item de A corresponde ao item de B, caminha
11             p ← p + 1           para o próximo item de A}
12             q ← q + 1
13       end
14       if (p = v + 1) then return(k)     {se p caminhou assim como q seu  tamanho é
15       k ← k + 1                                maior que o tamanho do itemset A, logo A
                                            foi percorrido completamente e seus ítens encontrados}
16  end  {end while}
17  return(m)
18 end_procedure
```

Algoritmo 3.1 – Pseudocódigo do algoritmo para verificação se, dadas duas sequências s_1 e s_2, $s_1 \sqsubseteq s_2$.

Exemplo 3.3 Considere as sequências s_1 = <(a,b)(c)(d,e,f)> e s_2 = <(a)(a,c,d)(a,b)(c,d)(d,e,f,g)>, com tamanhos $|s_1| = 6$ e $|s_2| = 12$, respectivamente, e a verificação se $s_1 \sqsubseteq s_2$. O algoritmo *verificaSubsequencia* busca sequencialmente em s_2, *itemsets* que contenham cada um dos *itemsets* de s_1. É importante notar que o número de *itemsets* de s_1 é 3 (n = 3) e de s_2 é 5 (m = 5).

A Tabela 3.2 mostra, na primeira iteração (t = 1), a busca do primeiro *itemset* de s_1, ou seja, a_1 = (a,b), na sequência s_2. Na primeira verificação, nota-se que o primeiro *itemset* de s_2, b_1 = (a), tem tamanho menor que aquele do *itemset* a_1 e, portanto, o algoritmo passa a considerar o seu sucessor, i.e., b_2 = (a,b). O algoritmo itera n vezes, no máximo – a cada iteração é considerado um novo *itemset* da sequência s_1, assumindo que as buscas dos *itemsets* anteriores de s_1 tenham sido bem sucedidas.

Como $|a_1| \leq |b_2|$ (ou seja, b_2 é um candidato plausível de conter a_1), o procedimento *buscaindice_em_s2* é invocado, com os seguintes parâmetros: a_1, s_2, índice do *itemset* em s_2, a partir do qual será iniciada a busca (no caso, 2 (b_2)) e o número total de *itemsets* em s_2, i.e., m.

Tabela 3.2 – Primeira iteração da execução do algoritmo para validar se $s_1 \sqsubseteq s_2$, em que s_1 = <(a,b)(c)(d,e,f)> e s_2 = <(a)(a,c,d)(a,b)(c,d)(d,e,f,g)>. A variável t representa a iteração atual.

t = 1										
a_t	**k(ultimo_em_s2)**	**b_k**			**$\|b_k\| < \|a_t\|$?**		**Procedimento**			
	1	(a)			Sim		Descarta b_k			
	2	(a,c,d)			Não		Busca índice i_t a partir de b_k			
		k	**b_k**	**r**	**p**	**q**	**q≤r && p≤v ?**	**a_p**	**b_{k_q}**	**$e_{t_p} = e_{k_q}$?**
	buscaindice_em_s2	2	(a,c,d)	3	1	1	Sim	a	a	Sim
	((a,b),s_2, 2, 5)	2	(a,c,d)	3	2	2	Sim	b	c	Não
(a,b)	**v = |(a,b)| = 2**	2	(a,c,d)	3	2	3	Sim	b	d	Não
	r = $\|b_k\|$	2	(a,c,d)	3	2	4	**Não**	–	–	–
	p: índice que varre a	**3**	(a,b)	2	1	1	Sim	a	a	Sim
	q: índice que varre b	3	(a,b)	2	2	2	Sim	b	b	Sim
		3	(a,b)	2	3	3	**Não**	–	–	–

No procedimento secundário invocado, a sequência candidata à supersequência é percorrida, a partir do índice k (último índice que assinala um *itemset* em s_2 que contém um *itemset* de s_1), até que seja encontrado um *itemset*

que contenha o *itemset* buscado ou, então, que após ter considerado b_m, último *itemset* de s_2, tal busca falhe, não encontrando o *itemset* buscado.

Ainda considerando as informações da Tabela 3.2, pode ser notado que o *itemset* a_t = (a,b) não é encontrado em b_k = (a,c,d) (t = 1 e k = 2). O algoritmo prossegue uma vez que a condição de término dada por (p = v + 1) não foi verificada. O índice k é incrementado (k = 3) e o processo se repete. Como a_t é encontrado em b_3, a busca de a_1 em s_2 é bem sucedida e o vetor saída I, recebe, na sua posição i_1 o valor 3.

Como nenhuma condição de parada do algoritmo foi satisfeita, próximo *itemset* de s_2 é, então, atualizado para 4 (i.e., aponta o *itemset* em s_2 a partir do qual o próximo *itemset* de s_1 será buscado) e uma nova iteração tem início (t = 2), que buscará o *itemset* a_2 (i.e., (c)) de s_1 em s_2, a partir de seu *itemset* b_4 (i.e. (c,d)) como mostra a Tabela 3.3.

Tabela 3.3 – Segunda iteração da execução do Algoritmo 3.1 para validar s_1 = <(a,b)(c)(d,e,f)> como subsequência da sequência s_2 = <(a)(a,c,d)(a,b)(c,d)(d,e,f,g)>.

t = 2										
a_t	**k(ultimo_em_s2)**		**b_k**			**$\|b_k\| < \|a_t\|$?**		**Procedimento**		
(c)	4		(c,d)			Não		Busca índice i_t a partir de b_k		
	buscaindice_em_s2	**k**	**b_k**	**r**	**p**	**q**	**q≤r && p≤v ?**	**a_p**	**b_{k_q}**	**$e_{t_p} = e_{k_q}$?**
	((c), s_2, 4,5)	4	(c,d)	2	1	1	Sim	c	c	Sim
	v = |(c)| = 1	4	(c,d)	2	2	2	**Não**	–	–	–
	r = $\|b_k\|$									
	p: índice que varre a									
	q: índice que varre b									

O conjunto I de índices é atualizado com a inclusão do valor 4, i.e., onde o *itemset* a_2 = (c) foi encontrado. O iterador t é incrementado para o valor 5, bem como o valor da variável ultimo_em_s_2.

Da mesma forma, como a condição de parada, linha 3 do algoritmo, não foi satisfeita o algoritmo prossegue buscando o próximo *itemset* de s_1 em s_2.

Tabela 3.4 – Terceira iteração da execução do Algoritmo 3.1 para validar s_1 = <(a,b)(c)(d,e,f)> como subsequência da sequência s_2 = <(a)(a,c,d)(a,b)(c,d)(d,e,f,g)>.

t = 3										
a_t	k(ultimo_em_s2)	b_k			$\|b_k\| < \|a_t\|$?		Procedimento			
(d,e,f)	5	(d,e,f,g)			Não		Busca índice i_t a partir de b_k			
	buscaindice_em_s2	k	b_k	r	p	q	q≤r && p≤v ?	a_p	b_{k_q}	$e_{t_p} = e_{k_q}$?
	((d,e,f), s_2, 5,5)	5	(d,e,f,g)	4	1	1	Sim	d	d	Sim
	v = |(d,e,f)| = 3	5	(d,e,f,g)	4	2	2	Sim	e	e	Sim
	r = $\|b_k\|$	5	(d,e,f,g)	4	3	3	Sim	f	f	Sim
	p: índice que varre a	5	(d,e,f,g)	4	4	4	**Não**	–	–	–
	q: índice que varre b									

Ao final da execução do algoritmo, a condição de parada é satisfeita com t maior que o tamanho da sequência s_1. O conjunto de índices I que representa as posições em s_2, as quais os *itemsets* de s_1 foram encontrados é composto pelos valores 3, 4 e 5. Indicando que o *itemset* a_1 de s_1 foi encontrado no *itemset* b_3 de s_2, bem como a_2 e a_3 em b_4 e b_5, respectivamente.

Exemplo 3.4 Considere ainda a sequência s_1 = <(a)(b)(c,d)> e s_2 = <(a,b)(b,c,d)(e)> com tamanhos $|s_1|$ = 4 e $|s_2|$ = 6, respectivamente. Da mesma forma como apresentado no Exemplo 3.3, o algoritmo *verificaSubsequencia* busca em s_2 todos os *itemsets* de s_1 para validar s_2 como uma supersequência de s_1, i.e., $s_1 \sqsubseteq s_2$. O número de *itemsets* de s_1 e de s_2 é 3, (n = m = 3).

Tabela 3.5 – Primeira iteração da execução do algoritmo para validar s_1 = <(a)(b)(c,d)> como subsequência da sequência s_2 = <(a,b)(b,c,d)(e)>.

t = 1										
a_t	k(ultimo_em_s2)	b_k			$\|b_k\| < \|a_t\|$?		Procedimento			
(a)	1	(a,b)			Não		Busca índice i_t a partir de b_k			
	buscaindice_em_s2	k	b_k	r	p	q	q≤r && p≤v ?	a_p	b_{k_q}	$e_{t_p} = e_{k_q}$?
	((a), s_2, 1, 3)	1	(a,b)	2	1	1	Sim	a	a	Sim
	v = |(a)| = 1	1	(a,b)	2	2	2	**Não**	–	–	–
	r = $\|b_k\|$									
	p: índice que varre a									
	q: índice que varre b									

Tanto na primeira iteração, Tabela 3.5, quanto na segunda iteração, Tabela 3.6, a condição de parada do algoritmo não é satisfeita e o algoritmo consegue encontrar correspondentes para os *itemsets* a_1 = (a) e a_2 = (b) de s_1, porém, como

pode ser visto na Tabela 3.7, na terceira iteração, t = 3, o algoritmo verifica que o *itemset* b_3 = (e) de s_2, candidata a *supersequência*, é menor que o *itemset* a_3 = (c,d) de s_1 buscado e, assim, o descarta.

Pela lógica do algoritmo, o sucessor de b_3 é o próximo a ser analisado, entretanto, como a variável ultimo_em_s_2 tem valor 4, a nova busca deve ser iniciada a partir do índice 4 (b_4), porém m = 3, dessa forma a condição de parada da linha 6 do algoritmo é satisfeita e sua execução para, não construindo por completo o conjunto de índices I.

Tabela 3.6 – Segunda iteração da execução do algoritmo para validar s_1 = <(a)(b)(c,d)> como subsequência da sequência s_2 = <(a,b)(b,c,d)(e)>.

t = 2									
a_t	**k(ultimo_em_s2)**	**b_k**			**$\|b_k\| < \|a_t\|$?**		**Procedimento**		
(b)	2	(b,c,d)			Não		Busca índice i_t a partir de b_k		
buscaindice_em_s_2	**k**	**b_k**	**r**	**p**	**q**	**q≤r && p≤v ?**	**a_p**	**b_{k_q}**	**$e_{t_p} = e_{k_q}$?**
((b), s_2, 2, 3)	2	(b,c,d)	3	1	1	Sim	b	b	Sim
v = |(b)| = 1	2	(b,c,d)	3	2	2	**Não**	–	–	–
r = $\|b_k\|$									
p: índice que varre a									
q: índice que varre b									

Tabela 3.7 – Terceira iteração da execução do algoritmo para validar s_1 = <(a)(b)(c,d)> como subsequência da sequência s_2 = <(a,b)(b,c,d)(e)>.

t = 3				
a_t	**k(ultimo_em_s2)**	**b_k**	**$\|b_k\| < \|a_t\|$?**	**Procedimento**
(c,d)	3	(e)	Sim	Descarta b_k

Definição 3.5 Seja DB uma base de dados de sequências. Seja *min_sup* um limite inferior, estabelecido pelo usuário, para o suporte de qualquer sequência. Uma sequência de dados s é uma *sequência frequente* (ou *padrão sequencial*) em DB se s.sup ≥ *min_sup*.

Dados uma base de sequências de dados, BD, e um valor para *min_sup*, o problema da mineração de sequências frequentes pode ser equacionado como uma busca pelo conjunto de todas as sequências frequentes em BD.

Exemplo 3.5 Considere uma base de dados de sequências D_1, contendo 3 sequências de dados, representadas pelo conjunto $D_1 = \{C_1, C_2, C_3\}$, como mostra a Tabela 3.8.

Tabela 3.8 – Base de Dados (D_1) de Sequências.

ID_seq	D_1
C_1	<(e,f)(a,b)(d,f)(c)(b)>
C_2	<(a)(b,c)(d)(c)>
C_3	<(a)(a,b,c,d)(f)(a,c)>

Considere a sequência s_i = <(a,b)(c)>. O cálculo de s_i.sup é dado por $\frac{|\{s_j \in D_1 \mid s_i \subset s_j\}|}{|D_1|}$. O conjunto de sequências $s_j \in D_1 \mid s_i \subset s_j$ = {<(e,f)(a,b)(d,f)(c)(b)>, <(a)(a,b,c,d)(f)(a,c)>} e portanto, suporte de s_i = 2/3 = 0,66. Considerando um valor de *min_sup* = 0,5, a sequência s_i pode ser considerada uma sequência frequente (i.e., um padrão sequencial) em D_1.

Exemplo 3.6 Considere uma base de dados de sequências D_2 contendo 6 sequências de dados, agrupadas no conjunto D_2 = {C_1,C_2,C_3,C_4,C_5,C_6}, como mostra a Tabela 3.9, conforme apresentado em (LIN; LEE, 2002).

Tabela 3.9 – D_2: Base de Dados de Sequências.

ID_seq	D_2
C_1	<(a,d)(b,c)(a,e)>
C_2	<(d,g)(c,f)(b,d)>
C_3	<(a,c)(d)(f)(b)>
C_4	<(a,b,c,d)(a)(b)>
C_5	<(b,c,d)(a,c,e)(a)>
C_6	<(b,c)(c)(a,c,e)>

Considere a sequência s_i = <(b)(a)>; o cálculo do suporte de s_i é dado por s_i.sup = $\frac{|\{s_j \in D_2 \mid s_i \subset s_j\}|}{|D_2|}$. O conjunto de sequências $s_j \in D_2 \mid s_i \subset s_j$, i.e., sequências de D_2 que contém s_i = <(b)(a)>, está discriminado na Tabela 3.10.

Tabela 3.10 – Sequências de D_2 que contém s_i = <(b)(a)>.

ID_seq	$s_i \in D_2 \mid s_i \subset s_j$
C_1	<(a,d)(b,c)(a,e)>
C_4	<(a,b,c,d)(a)(b)>
C_5	<(b,c,d)(a,c,e)(a)>
C_6	<(b,c)(c)(a,c,e)>

Considere ainda D_2 e uma nova sequência s_k = <(a,d)(a)>. O conjunto de sequências $s_j \in D_2 \mid s_k \subset s_j$ = {<(a,d)(b,c)(a,e)>, <(a,b,c,d)(a)(b)>} e, portanto, s_k.sup = 2/6 = 1/3 = 0,33. Considerando um valor de *min_sup* = 0,4; a sequência s_k = <(a,d)(a)> não se qualifica como um padrão, i.e., não frequente.

Considerando a base de sequências D_2 do Exemplo 3.6 e *min_sup* = 0,5, as únicas sequências que se qualificam como padrão sequencial em D_2 estão discriminadas na Tabela 3.11, com o respectivo valor de suporte identificado.

Tabela 3.11 – Padrões sequenciais com relação à base de sequências D_2.

Padrão	Suporte	Padrão	Suporte
<(a)>	0,83	<(b,c)(e)>	0,5
<(a)(a)>	0,5	<(b,d)>	0,5
<(a)(b)>	0,5	<(c)>	1
<(a,c)>	0,67	<(c)(a)>	0,67
<(a,c)(a)>	0,33	<(c)(a,e)>	0,5
<(a,e)>	0,5	<(c)(b)>	0,5
<(b)>	1	<(c)(e)>	0,5
<(b)(a)>	0,67	<(d)>	0,83
<(b)(a,e)>	0,5	<(d)(a)>	0,5
<(b)(e)>	0,5	<(d)(b)>	0,67
<(b,c)>	0,67	<(d)(c)>	0,5
<(b,c)(a)>	0,67	<(e)>	0,5
<(b,c)(a,e)>	0,5	–	–

Definição 3.6 Seja ρ um padrão sequencial e seja x um item frequente na base de dados BD.

(1) Um padrão ρ' é um padrão tipo-1 se pode ser formado a partir de ρ, estendendo-o com um novo *itemset* (x).

(2) Um padrão ρ' é um padrão tipo-2 se for formado a partir de ρ, estendendo o seu ultimo *itemset* com o item x.

(3) Tanto em (1) quanto em (2), x é chamado de *item-stem* ou *stem* de ρ' e ρ de *P-pat* de ρ'.

Exemplo 3.7 Considere um padrão sequencial ρ = <(a)> e um item frequente b. Um padrão do tipo-1 ρ' pode ser formado estendendo ρ com a inclusão do item b, gerando ρ' = <(a)(b)>. Um padrão do tipo-2 também pode ser formado estendendo o

ultimo itemset do padrão ρ com o item b, gerando ρ'' = <(a,b)>. Note que a sequência vazia < > é P-pat de todos os padrões.

A Definição 3.7, a seguir, foi formalizada neste trabalho uma vez que ela se encontra apenas informalmente apresentada como um exemplo, em (LIN; LEE, 2002).

Definição 3.7 A relação R, associada ao padrão sequencial ρ é formada por pares <x,y> tal que x $\in$ I, y $\in$ BD e x marca a posição (em y), do término de ρ em sua primeira ocorrência em y.

O conjunto de índices ρ-idx é uma relação no domínio de R $\subseteq$ I x BD. É gerado um par (pos,ptr_ds) para cada sequência C_i que apresentar o padrão ρ, onde ptr_ds é um ponteiro para a sequência de dados que contém o stem e pos é a posição dentro da sequência onde ocorre a primeira aparição do stem.

Exemplo 3.8 Seja o padrão ρ=<(a,b)(c)> e a base de dados de sequências (BD) descrita no Exemplo 3.6 com as sequências s_1 = <(e,f)(a,b)(d,f)(c)(b)> , s_2 = <(a)(b,c)(d)(c)> e s_3 = <(a)(a,b,c,d)(f)(a,c)>. A relação R associada ao padrão ρ é formada por uma lista de índices, notada por <(a,b)(c)>-idx e composta pelos pares (pos, ptr_ds) que identificam as ocorrências do padrão em BD. Sendo assim, <(a,b)(c)>-idx é formada pelos pares (7,s_1) e (8,s_3) identificando que o padrão ρ = <(a,b)(c)> ocorre em s_1 e sua primeira ocorrência é finalizada no item 7 de s_1 = <(e,f)(**a**,**b**)(d,f)(**c**)(b)>. Da mesma forma, (8,s_3) indica que o padrão ρ ocorre em s_3 e sua primeira ocorrência é finalizada no item 8 de s_3 = <(a)(**a**,**b**,c,d)(f)(a,**c**)>.

3.2 MEMISP (Memory Indexing for Sequential Pattern mining)

O algoritmo MEMISP, proposto por Lin e Lee (2002), pode ser melhor compreendido como uma variação dos algoritmos GSP (SRIKANT; AGRAWAL, 1996) e PrefixSpan (PEI et al, 2001). À semelhança como o GSP e o PrefixSpan, o MEMISP também varre uma base de dados em busca de padrões sequenciais, porém não gera candidatos (como faz o GSP) nem projeta pequenas bases de dados (PrefixSpan). O algoritmo se utiliza de uma técnica recursiva chamada *find-then-index*, para encontrar os padrões a partir dos itens mais frequentes da BD. A

cada iteração o conjunto de padrões potenciais encontrados é avaliado, dentre eles são identificados os padrões efetivos e o algoritmo prossegue recursivamente, até que novos padrões não sejam mais identificados, condição de parada do algoritmo.

Bases de dados com tamanho muito grande, que ultrapassam a capacidade de armazenamento da memória, são também possíveis de serem mineradas. Isso é feito por meio da partição da base em grupos cujo volume de dados pode ser inteiramente alocado em memória. Dessa forma o algoritmo varre os dados contidos em cada uma das partições para encontrar os padrões possíveis e em seguida efetua uma nova varredura para validar os reais padrões. Essa técnica é chamada *partition-and-validation* e está descrita com mais detalhes na Seção 3.2.1. Essa ocasião é a única em que o algoritmo efetua mais de uma varredura na base de dados, uma para cópia e uma para validação na base completa dos padrões locais encontrados em cada partição.

Em ambos os casos o algoritmo inicia contando a frequência (suporte) das sequências menores, de tamanho unitário (1-sequência), cria um conjunto de índices para cada padrão identificado e, recursivamente, descobre todas as sequências de tamanhos maiores buscando no conjunto de sequências de dados, as que possuem subsequências em comum com relação ao padrão analisado.

3.2.1 A técnica Partition-and-Validation

Para tornar a mineração de bases de dados mais eficiente, alguns algoritmos fazem uma cópia da base para a memória interna (RAM), construindo a chamada MDB (M*emory DataBase*). Muitas vezes, a memória não é grande suficiente para armazenar todo o volume de registros que as bases possuem, portanto a cópia é feita de forma particionada. A técnica *partition-and-validation*, utilizada pelo algoritmo MEMISP, propõe a divisão da base de dados em subgrupos, que cabem na memória, e assim, podem ser minerados individualmente.

A técnica é executada em duas etapas: (1) partição da base de dados e (2) validação de sequências frequentes. Na primeira etapa, representada pelo fluxograma da Figura 3.1, a base de dados, BD, é analisada em subgrupos (G_i) que são, um a um, copiados para a memória interna disponível, MG_i. Cada um dos grupos, MG_i, é analisado pela técnica *find-then-index,* responsável por encontrar padrões locais e os armazenar em um conjunto de padrões locais, cp_i. Ao final da

primeira etapa da técnica *partition-and-validation,* o conjunto CP = { cp_1, cp_2, ..., cp_n} contém todos os padrões locais encontrados em cada subgrupo, MG_i.

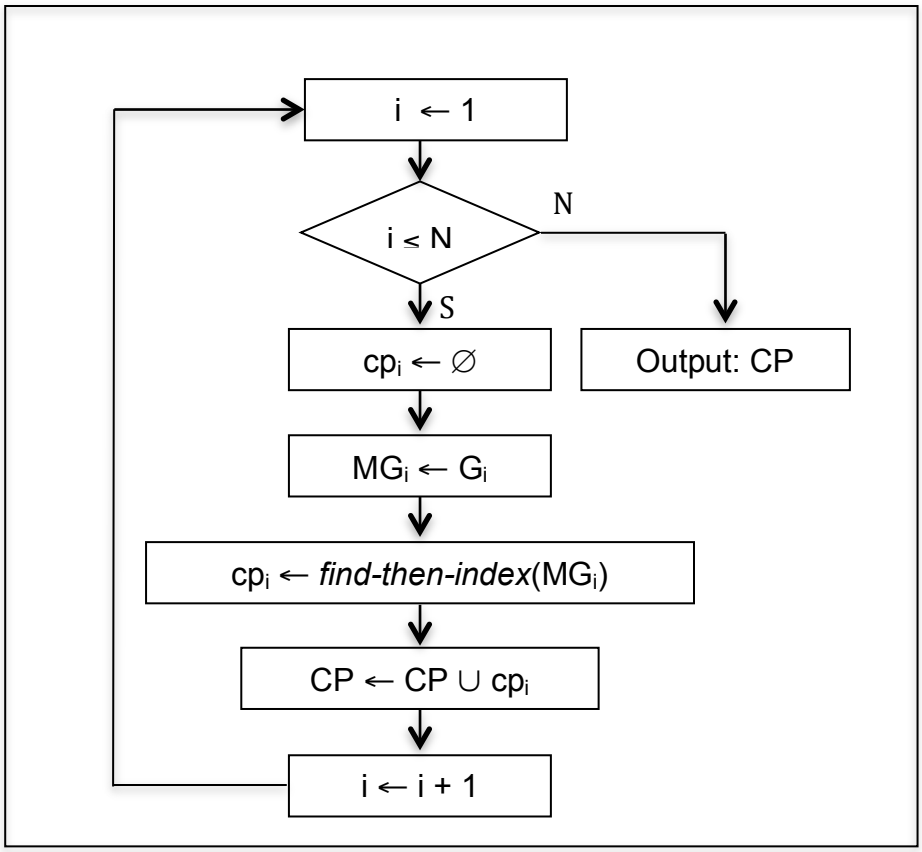


Figura 3.1 – Primeira etapa do procedimento *partition-and-validation* para mineração de bases de dados maiores que a memória interna disponível.

A segunda etapa da técnica *partition-and-validation* é responsável pela validação de todos os padrões locais contidos em CP, como padrões efetivos. Conforme pode ser visto no fluxograma da Figura 3.2, cada partição (cp_i) encontrada na primeira etapa é analisada e os suportes de cada um dos *itemsets* contidos nela são calculados com relação a base de dados completa (BD). Todos os *itemsets* com suporte maior que o limite min_sup, pré-definido, são considerados padrões sequenciais da base de dados e compõem o conjunto de padrões FCP.

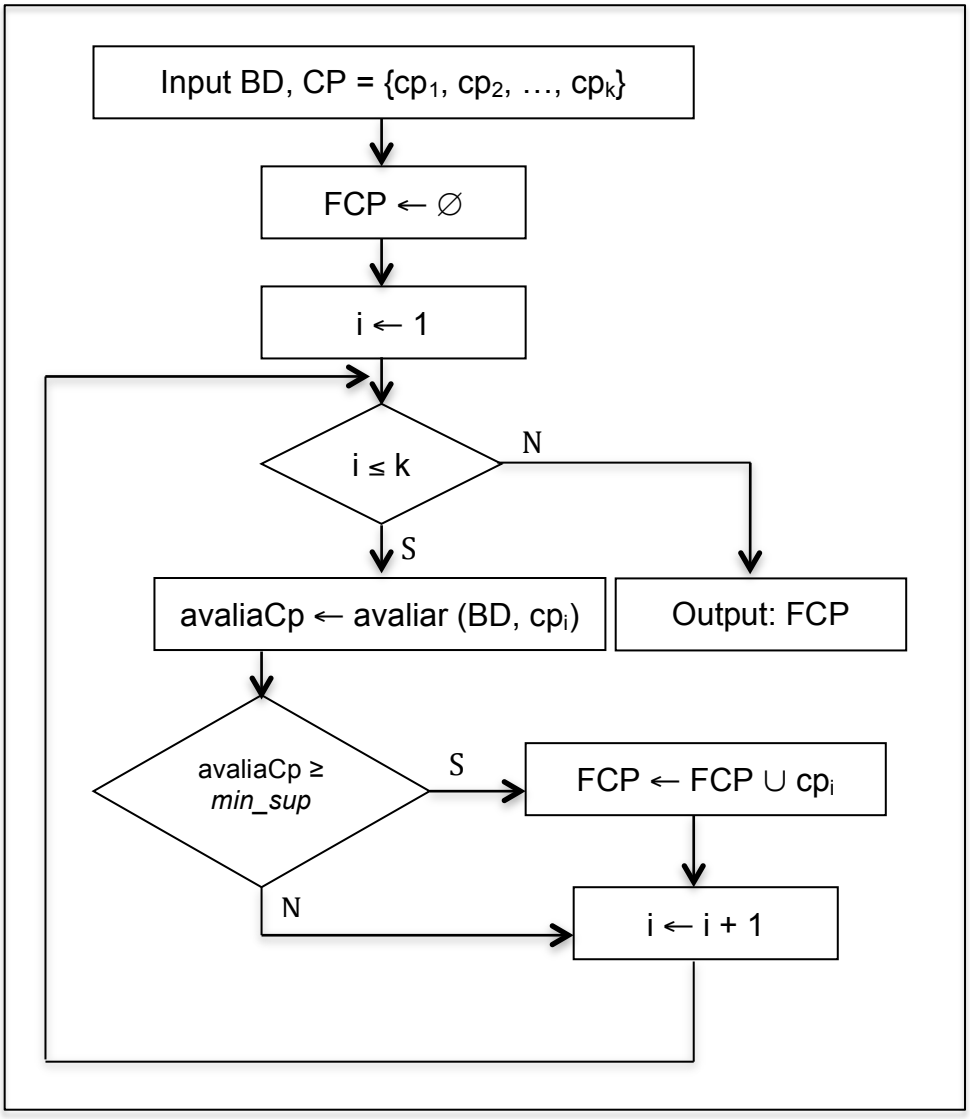


Figura 3.2 – Segunda etapa do procedimento *partition-and-validation* para mineração de bases de dados maiores que a memória interna disponível.

3.2.2 Passos do Algoritmo

O algoritmo MEMISP proposto para a identificação e mineração de sequências em uma base de dados é executado seguindo 3 etapas, seu pseudocódigo é apresentado em Algoritmo 3.2, extraído de (LIN; LEE, 2002), e reescrito com mais detalhes.

Passo1 Considerando os dados e sequências já relacionados na Tabela 3.9 e que o *min_sup* = 0,5 (50%) o algoritmo parte do princípio da sequência possível mais trivial na base de dados, com apenas 1 item.

Na primeira etapa da execução do algoritmo, uma única leitura na base de dados é feita de forma que as sequências sejam copiadas para a memoria interna, a maior quantidade que puder ser copiada, construindo a base de dados em memória (MDB - *Memory DataBase*).

Os itens inicialmente considerados pelo algoritmo são unitários, dessa forma enquanto as sequências são copiadas, o suporte de cada item unitário é calculado e armazenado em uma estrutura de dados.

Revisitando a Tabela 3.11, é possível ver que, entre os elementos listados, estão relacionados os itens frequentes, como x_1 = a com contagem de frequência igual a 5, consequentemente x_1.sup = 0,83 pois está presente nos *itemsets* das sequências C_1, C_3, C_4, C_5 e C_6, assim como x_2 = b, x_3 = c, x_4 = d e x_5 = e com contagens de frequências iguais a 6, 6, 5 e 3, respectivamente. Todos os itens x_i são stem do padrão ρ com respeito ao P-pat = <>. Os passos 2 e 3, descritos a seguir, são repetidos, iterativamente, para cada um dos stems unitários encontrados nessa primeira etapa do processo.

Passo 2 Tomando como stem de exemplo o item x_1 = a, o segundo passo é gerar um padrão ρ_i utilizando o P-pat atual, inicialmente vazio (< >) e o stem selecionado, neste exemplo, ρ_1 = <(a)>. A partir desse pressuposto é possível gerar o conjunto de índices para o padrão atual.

Nesta etapa da execução do algoritmo, é criada uma lista de índices denominada <(a)>-idx contendo um par de ponteiros (pos,ptr_dr) para cada ocorrência do padrão ρ_1 = <(a)> nas sequências copiadas para a memória interna.

Para o stem tomado inicialmente e o padrão ρ_1 = <(a)>, é gerado o conjunto <(a)>-idx com o mapeamento para as sequências C_1, C_2, C_3, C_4, C_5 e C_6, pois são as que apresentam o padrão ρ_1.

Passo 3 Encontrar todo padrão que possua ρ_1 como P-pat, isso é, o padrão atual torna-se o prefixo e novos stems são procurados para que sequências maiores sejam então descobertas. Qualquer item que venha após a primeira ocorrência (pos) de ρ_1 na sequência pode ser um potencial stem para a composição de um novo padrão.

No caso analisado para o padrão ρ_1=<(a)> e a sequência de dados C_1, o par que mapeia a ocorrência de ρ_1 em C_1, possui pos = 1, portanto só são analisados os

itens após a posição 1. Nota-se que o padrão ρ_1' = <(a,d)>, tipo-2, pode ser gerado pelo stem *d* se for considerado o *min_sup*, assim como e em ρ_1'' = <(a,e)>. Outros padrões do tipo-1 também podem ser gerados com stem's a,b e c. Esse processo é efetuado para todos os pares da lista de índices.

Os novos padrões encontrados formam novos conjuntos de índices e tornam-se P-pat para que novos stem's sejam procurados. O processo é recursivamente refeito até que não se encontre mais possíveis padrões. Os padrões e conjuntos de índices podem ser melhor visualizados na Figura 3.3, onde observa-se em (1) o conjunto <(a)>-idx é base para (2), (3), (4) e (6), em (4) ainda observa-se que o padrão do conjunto (5) pode ser gerado então também é considerado.

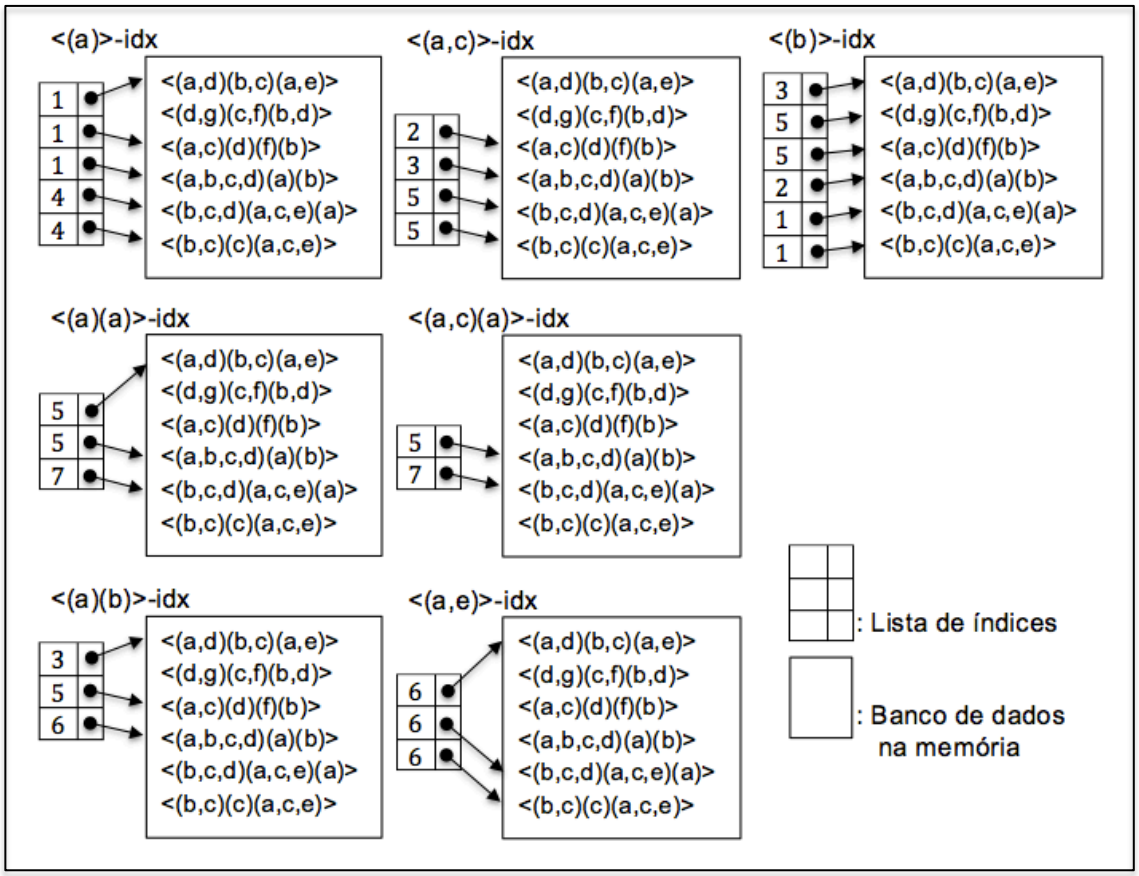


Figura 3.3 – Listas de índices dos padrões.

```
Procedure MEMISP
Entrada: {DB = Base de dados de sequências; min_sup = suporte mínimo}
Saída: { pat =  conjunto de todos padrões sequenciais encontrados em DB
1 begin
2   td ← tamanho(DB)
3   j ← 1
4   for i ← 1 to td do
5     begin
6       if( suporte(xi) > min_sup )then { se o ítem analisado for um ítem frequente}
7         begin
8           lj ← xi                {atualiza o conjunto de itens frequentes}
9           j ← j+1
10        end
11    end
12    ti ← tamanho(l)
13    for  j ← 1 to ti do
14      begin
15        ρ ← geraPadrao("<>", lj)          {gera o padrão sequencial ρ = <(x)> }
16        ρ-idx ← indexSet(lj, "<>", MDB)   {gera o índice  ρ-idx; MDB = base de dados na memória}
17        pat ← pat U ρ-idx
18        mine(ρ, ρ-idx,min_sup)            {busca todos os padrões ρ com índice ρ-idx}
19      end
20    return pat
21 end procedure

Procedure indexSet(x, ρ, range-set)
Entrada: { x = stem-item; ρ = P-pat; range-set = {c1, c2, ..., ck}  {conjunto de sequências de dados para
           indexar}
Saída: {conjunto de índices ρ'-idx = {p1, p2, ..., pq}, onde ρ' denota o padrão formado pelo stem-item e P-
         pat; cada item p é um par que representa a relação do padrão ρ com range-set}
1 begin
2   t_rs ← tamanho(range_set)
3   t_MDB ← tamanho(MDB)
4   for j ← 1 to t_rs do
5     begin
6       ds ← cj     { sequência de dados de range_set}
7       if( t_rs = t_MDB ) then start_pos ← 1
8       else start_pos ← pos + 1
9       for i ← start_pos to tamanho(ds) do
10        begin                                  {se o stem-item x é 1o. encontrado na posição pos em
11          if( primeiro_em_ds(x, pos,ds)         ds, inserir um par (pos,ptr_ds) no conujnto de índices
12            ρ'-idx ← ρ'-idx + par(pos,ptr_ds)   ρ'-idx, em que ptr_ds aponta para ds}
13        end
14    end
15  return ρ'-idx
16end

Procedure Mine(ρ,ρ-idx,min_sup)
Entrada: { : ρ=um padrão/ρ-idx=Conjunto de índices/min_sup=Minimo valor de suporte considerado}
Saída: {conjunto de padrões ρ' encontrados na base de dados}
1 begin
2   tam_p ← tamanho(ρ-idx)
3   tam_ds ← tamando(ds)
4   for i ←1 to tam_p do
5     for j ← 1 to tam_ds do
6      begin
7       itemj.sup ← itemj.sup + 1
8       if( itemj.sup ≥ min_sup) then
9         l ← l U itemj
10    end
11    tam_l ← tamanho(l)
12    for k ← 1 to tam_l do
13    begin
14      ρ' ← geraPadrao(ρ, lk)                {gera o padrão sequencial ρ' }
15      ρ'-idx ← indexSet(lk, ρ, ρ-idx)       {gera a lista de índices  ρ'-idx}
16      mine(ρ', ρ'-idx,min_sup)              {busca todos os padrões ρ' com índice ρ'-idx}
17    end
18   return ρ'
19 end procedure
```

Algoritmo 3.2 – Pseudocódigo do algoritmo MEMISP.

Capítulo 4

ALGORITMO ARMADA - INCORPORAÇÃO DO ASPECTO TEMPORAL

Ao considerar o sequenciamento das informações contidas em uma base de dados, é possível observar a evolução dos dados e, consequentemente, associar ao tempo em que tais mudanças ocorreram – O tempo é um fator de grande importância para a área de banco de dados. Este capítulo apresenta o algoritmo ARMADA, referência no processo de mineração de dados temporais e suas técnicas empregadas. O capítulo também uma forma de interpretar as relações entre intervalos temporais associados aos dados – descrita por meio do formalismo da Algebra Intervalar de Allen.

4.1 Considerações Iniciais

A grande maioria de trabalhos relacionados ao tema de mineração de dados temporais, que é um dos focos dessa investigação, é subsidiada pelo formalismo de representação temporal proposto por Allen (1981), conhecido como Álgebra Intervalar de Allen (AIA). Como será mostrado, a AIA é um dos formalismos mais simples e mais utilizados para representação e raciocínio sobre conhecimento temporal. O algoritmo ARMADA (*An algorithm for discovering Richer relative teMporal Association rules from temporal DAta)* (WINARKO; RODDICK, 2007),

principal referência utilizada nesta pesquisa para identificação de padrões temporais e geração de regras de associação entre eles, faz uso do formalismo proposto por Allen, para relacionar os intervalos temporais associados aos eventos armazenados em uma base de dados. No que segue, a Seção 4.2 apresenta os principais conceitos relacionados à AIA e a Seção 4.3 investe na apresentação/discussão do ARMADA.

4.2 A Álgebra Intervalar de Allen

Como comentado em Nicoletti et al. (2013) "... a representação de conhecimento temporal com vistas à implementação de raciocínio temporal é um problema recorrente em muitas áreas de conhecimento, particularmente na área de Inteligência Artificial (IA). O aspecto temporal de situações e eventos é extremamente relevante para a implementação de sistemas inteligentes; as inúmeras propostas para representar eficientemente aspectos temporais da informação, bem como as várias revisões encontradas na literatura são evidência da importância desse aspecto - ver, por exemplo, (LONG, 1989), (KNIGHT; MA, 1994), (MA; KNIGHT, 1996, 2001), (BETTINI, 1997), (BELLINI et al., 2000), (CHITARO; MONTANARI, 2000), (SCHOCKAERT et al., 2008), (FURIA et al., 2010) e (LADKIN, 1986)." No que segue as informações sobre a AIA foram compiladas de muitas fontes que discutem o formalismo, particularmente de Allen e Ferguson (1994).

A AIA é um modelo linear de tempo razoavelmente simples que tem um objeto primitivo identificado como *período de tempo* e uma relação primitiva chamada *meets/2.* Um *período de tempo* pode ser abordado como um espaço de tempo associado com algum evento ou alguma propriedade sendo verificada, no mundo real. Como apontado por Nicoletti et al. (2013), embora a noção intuitiva associada com a relação *meets/2* seja bastante clara, sua descrição como um conceito formal (com vistas à sua implementação) não é trivial, uma que envolve uma definição prévia da granularidade de tempo adotada, bem como da representação adotada para intervalo de tempo. Allen (1983) discute a conveniência no uso de pontos de tempo em oposição à intervalo de tempo, como base para a representação e raciocínio temporais. Como Mani et al. (2004) comentam,

"Dependendo da representação e da escolha da primitiva (intervalos, instâncias ou ambos), uma variedade de diferentes relações entre tempos pode ser definida. "

Apesar da AIA considerar intervalos de tempo como o conceito primitivo, o formalismo contempla a possibilidade de, assumindo um modelo consistindo de um conjunto de pontos de dados completamente ordenado, representar um intervalo T como um conjunto ordenados de pares de pontos de tempo [t−, t+], em que cada par satisfaz a pré-condição do primeiro ponto ser menor que o segundo i.e., t− < t+.

A Tabela 4.1 apresenta na primeira coluna as cinco relações binárias básicas entre intervalos temporais da AIA, a saber, *meets*, *before*, *overlaps*, *during*, e *equal*. Um intervalo é tratado como um período de tempo definido por dois pontos temporais. Nas cinco relações, o intervalo A = [a−,a+] (a− < a+) e B= [b−, b+] (b− < b+), sendo que {a−,a+, b−,b+} ⊆ ℜ. A segunda coluna exibe as condições *default* associadas, derivadas do fato que para um intervalo de tempo qualquer T = [t−,t+], a desigualdade t− < t+ é sempre verdade.

Tabela 4.1 - As cinco relações temporais (binárias) básicas de Allen. Intervalo A = [a−,a+] (a− < a+) e B= [b−, b+] (b− < b+), sendo que {a−,a+, b−,b+} ⊆ ℜ. A segunda coluna exibe as correspondentes condições *default* associadas a cada uma das cinco relações e a terceira coluna mostra uma representação pictórica geral associada a cada uma delas.

Relação temporal entre intervalos de tempo	Condições que os pontos que definem cada um dos intervalos envolvidos devem satisfazer	Correspondente representação pictórica geral
meets(A,B)	a− < b−, a− < b+, **a+ = b−**, a+ < b+	AAAA BBBBBBB
before(A,B)	a− < b−, a− < b+, **a+ < b−**, a+ < b+	AAAA BBBBBBB
overlaps(A,B)	**a− < b−**, a− < b+, **a+ > b−**, **a+ < b+**	AAAA BBBBBBB
during(A,B)	(**a− > b−**, **a+ =< b+**) ; (**a− >= b−**, **a+ < b+**)	Ver Tabela 4.2
equal(A,B)	**a− = b−**, a− < b+, a+ > b−, **a+ = b+**	AAAAA BBBBB

As condições em negrito são aquelas que caracterizam efetivamente a relação, dado que as outras caracterizam intervalos de tempo. Nas expressões mostradas na coluna 2 a vírgula representa o operador lógico ‘and’ e o ponto e vírgula, o operador lógico inclusivo ‘or’.

De acordo com Allen a subdivisão da relação *during* em três outras a saber, *starts*, *finishes* e uma nova *during* fornece um melhor modelo computacional; as três relações temporais que refinam e particularizam a relação *during* estão mostradas na Tabela 4.2. Considerando a subdivisão da relação *during*, o número de relações temporais básicas passa a ser 7.

Lembrando que se R for uma relação binária, a relação reversa de R, notada por R^{-1}, é uma relação tal que $yR^{-1}x$ se e somente se xRy. As sete relações básicas mostradas nas Tabela 4.1 e Tabela 4.2 podem ser expandidas em 13, se suas relações reversas forem consideradas (a reversa da relação *equal* é ela própria). A Tabela 4.3 nomeia as relações reversas associadas às 6 relações temporais básicas (*equal* excluída).

Tabela 4.2 - Subdivisão da relação *during* em três novas relações: *starts*, *finishes* e uma nova relação *during*. Notação e convenção seguem aquelas estabelecidas para a Tabela 4.1.

Relação temporal entre intervalos de tempo	Condições que os pontos que definem cada um dos intervalos envolvidos devem satisfazer	Correspondente representação pictórica geral
starts(A,B)	**a– = b–**, a– < b+, a+ > b–, **a+ < b+**	AAAA BBBBBBB
finishes(A,B)	**a– > b–**, a– < b+, a+ > b–, **a+ = b+**	AAAA BBBBBBB
during(A,B)	**a– > b–**, a– < b+, a+ > b–, **a+ < b+**	AAAA BBBBBBB

Tabela 4.3 - Relações temporais básicas, suas correspondentes relações reversas e a relação de equivalência entre cada um dos pares.

Relação Temporal	Relação Reversa	Equivalências
meets	met_by	meets(A,B) ≡ met_by(B,A)
before	after	before(A,B) ≡ after(B,A)
overlaps	overlapped_by	overlaps(A,B) ≡ overlapped_by(B,A)
starts	started_by	starts(A,B) ≡ started_by(B,A)
finishes	finished_by	finishes(A,B) ≡ finished_by(B,A)
during	contains	during(A,B) ≡ contains(B,A)

Como apresentado e discutido em (ALLEN; HAYES, 1985), levando em consideração as possíveis relações entre dois períodos de tempo, tais relações podem ser construtivamente redefinidas exclusivamente em termos da relação *meets*, como mostra a Tabela 4.4 (relações *meets* e *equal* excluídas).

A redefinição (como exibida na Tabela 4.4) subsidiou a evolução da AIA na proposta de uma teoria de temporalidade (TT), inicialmente descrita por Allen e Hayes (1985) e posteriormente revisada em (ALLEN; HAYES, 1989), (ALLEN, 1991)

e (ALLEN; FERGUSON, 1994). A TT é baseada em uma classe não vazia, I, de períodos (intervalos) de tempo (em que período de tempo é assumido como conceito primitivo). A TT é axiomatizada por meio do uso de uma única relação binária temporal, *meets*, assumida como primitiva, que pode ser estabelecida entre dois intervalos temporais. Intuitivamente dois intervalos de tempo x e y (nesta ordem) satisfazem à relação *meets* se e apenas se x precede y e, no entanto, (1) não existe tempo entre x e y e (2) x e y não se sobrepõem.

Tabela 4.4 - Relações temporais básicas entre períodos temporais i_1 e i_2 e suas respectivas representações em termos da relação primitiva *meets*.

Relação Temporal	Definição equivalente em termos apenas da relação primitiva *meets*
before(i_1,i_2)	$\exists$ i $\in$ I tal que (meets(i_1,i) $\wedge$ (meets(i,i_2))
overlaps(i_1,i_2)	$\exists$ i,j,k,l,m $\in$ I tal que (meets(i,i_1) $\wedge$ (meets(i_1,l) $\wedge$ meets(l,m) $\wedge$ (meets(i,j) $\wedge$ meets(j,i_2) $\wedge$ meets(i_2,m) $\wedge$ (meets(j,k) $\wedge$ meets(k,l))
starts(i_1,i_2)	$\exists$ i,j,k $\in$ I tal que (meets(i,i_1) $\wedge$ (meets(i_1,j) $\wedge$ meets(j,k) $\wedge$ (meets(i,i_2) $\wedge$ meets(i_2,k))
finishes(i_1,i_2)	$\exists$ i,j,k $\in$ I tal que (meets(i,j) $\wedge$ (meets(j,i_1) $\wedge$ meets(i_1,k) $\wedge$ (meets(i,i_2) $\wedge$ meets(i_2,k))
during(i_1,i_2)	$\exists$ i,j,k,l $\in$ I tal que (meets(i,j) $\wedge$ (meets(j,i_1) $\wedge$ meets(i_1,k) $\wedge$ (meets(k,l) $\wedge$ meets(i,i_2) $\wedge$ meets(i_2,l))

A axiomatização proposta subsoma (ver subsumir, subsunção) a AIA e seus seis axiomas são descritos a seguir; neles i, j, k, l, m, n são variáveis lógicas restritas ao domínio I.

(A1): Unicidade dos pontos de encontro – estabelece que o 'ponto' em que dois intervalos se encontram (i.e., dois intervalos que satisfazem a relação *meets*) é único e está associado com os intervalos considerados. Se os dois intervalos estão em uma relação *meets* com um terceiro, então qualquer intervalo que está em uma relação *meets* com um, estará com o outro também, situação formalmente definida por:

$\forall$ i; j; k, l $\in$I (meets(i, j) $\wedge$ meets(i, k) $\wedge$ meets(l, j) $\rightarrow$ meets(l, k)).

(A2): Pontos de encontro são totalmente ordenados – se intervalo I está em uma relação *meets* com intervalo j e intervalo k está em uma relação com intervalo l, então exatamente uma das situações é verdade: (a) i está em uma relação *meets* com l; (b) $\exists$ intervalo m tal que i está em uma relação *meets* com m e m está em uma relação *meets* com l e (c) $\exists$ intervalo n tal que k está em uma relação *meets* com n e n está em uma relação *meets* com j, representadas simbolicamente por:

$\forall$ i, j, k, l $\in$ I ((meets(i,j) $\wedge$ meets(k, l)) $\rightarrow$ meets(i, l) $\oplus$ $\exists$ m tal que meets(i,m) $\wedge$ meets(m,l) $\oplus$ $\exists$ n tal que meets(k, n) $\wedge$ meets(n,j)).

(A3): Continuidade do tempo – não existe começo ou fim do tempo; cada intervalo tem pelo menos um intervalo vizinho que o precede e outro que o sucede, formalmente definido por:

$\forall i \in I, \exists j, k \in I$ (meets(j,i) $\wedge$ meets(i,k)).

(A4): Intervalos iguais – se dois intervalos, ambos, estão em uma relação *meets* com um mesmo intervalo e um terceiro intervalo está em uma relação *meets* com ambos, então os dois intervalos iniciais são o mesmo intervalo (ou seja, são iguais), formalmente definido por:

$\forall i, j \in I, \exists k,l \in I$ (meets(k,i) $\wedge$ meets(k,j) $\wedge$ meets(i,l) $\wedge$ meets(j,l) $\rightarrow$ i = j).

(A5):Composição de intervalos – intervalos podem ser compostos produzindo um intervalo maior. Para quaisquer dois intervalos que estão em uma relação *meets,* existe um terceiro intervalo que é a concatenação de ambos, formalmente descrito por:

$\forall$ i, j, k, l $\in I$ (meets(i,j) $\wedge$ meets(j,k) $\wedge$ meets(k,l)) $\rightarrow \exists$ m (meets(i,m) $\wedge$ meets(m,l)).

4.3 O Algoritmo ARMADA

Como descrito anteriormente e enfatizado em (WINARKO; RODDICK, 2007), a mineração de dados temporais pode ser caracterizada como a busca por correlações interessantes ou padrões, em grandes conjuntos de dados temporais. A mineração de dados temporais busca descobrir padrões ou regras que podem ser desconsideradas quando o componente temporal for ignorado ou, então, quando for tratado como um simples atributo numérico.

Muitas pesquisas na área de mineração de dados temporais buscam induzir regras temporais tais como padrões sequenciais (AGRAWAL; SRIKANT, 1995), regras de associação temporal (NING et al, 2001), (OZDEN et al, 1998) e regras de associação inter-transação (LU et al, 2000). Como lembram Winarko e Roddick em (WINARKO;RODDICK, 2007), a maioria dessas pesquisas focalizam dados temporais que foram armazenados e interpretados como pontos temporais, e não como intervalos temporais. Como apontado em (BÖHLEN et al., 1998), em muitos

casos, certos acontecimentos não são instantâneos; eles ocorrem em um intervalo de tempo, portanto são melhores de serem tratadas como tal.

O algoritmo ARMADA (*An algorithm for discovering Richer relative teMporal Association rules from temporal DAta)* (WINARKO; RODDICK, 2007) se propõe a induzir regras de associação temporais a partir de dados. Em tais dados, a temporalidade é representada como um intervalo associado ao dado. Tais autores lembram que a mineração de regras temporais em que o tempo tem uma representação intervalar é, sem sombra de dúvida, bem mais complexa e requer uma abordagem diferente daquela usada por métodos que se propõe à mesma tarefa usando, entretanto, dados em que a temporalidade é representada pontualmente. Como um intervalo tem uma duração, os padrões gerados têm uma semântica diferente daquela dada simplesmente pelas relações antes e depois, por exemplo. As possibilidades de relacionamentos descritas pela AIA (apresentada na Seção 4.2) são comumente utilizadas para descrever relações entre intervalos temporais.

A proposta ARMADA pode ser abordada como o algoritmo MEMISP (LIN; LEE, 2002) (ver Capítulo 3) estendido, de maneira a fazer a mineração de padrões temporais frequentes. A escolha do MEMISP, segundo os seus autores, se deve ao fato de tal algoritmo ser mais eficiente que ambos, GSP (SRIKANT; AGRAWAL, 1996) e PrefixSpan (PEI et al, 2001), na busca de padrões sequencias a partir de uma base de dados de transações (BD). De maneira similar ao MEMISP, o ARMADA requer apenas uma 'varredura' da BD, etapa em que a BD é copiada para a memória interna do computador e referenciada como MDB (Memory DataBase). Ainda, o ARMADA, não requer a geração de candidatos ou projeção da BD.

Quando a BD for muito volumosa para ser armazenada na memória, o algoritmo a divide em partes, via procedimento *partition-and-validation* (ver Seção 3.2.1) e realiza a mineração em cada uma delas. Uma segunda 'varredura' da BD é então feita para validar os padrões encontrados. Em sequência, as regras temporais, denominadas pelos autores como 'regras de associação ricas' são geradas a partir dos padrões frequentes identificados.

Adicionalmente, uma restrição de tempo identificada pelo parâmetro *maximum_gap* é introduzida, que pode ser usada para remover padrões insignificantes o que, por sua vez, pode reduzir o número de padrões frequentes e, consequentemente, o número de regras de associação temporais geradas pelo

algoritmo.

O ARMADA considera o conceito de *itemset* de maneira diferenciada daquela considerada pelo MEMISP, devido ao tratamento de temporalidade. Para tanto, adota a definição de sequência de estado, proposta por Höppner (2001), como segue. Para a notação empregada pelo ARMADA no tratamento do aspecto temporal, o algoritmo faz uso da conceituação relacionada à temporalidade proposta em (HÖPPNER, 2001).

No processo de identificação de padrões, assim como descrito anteriormente, o ARMADA realiza uma cópia da BD para a memória interna enquanto realiza uma única varredura na base de dados. Enquanto a construção da MDB é feita, os itens (unitários) frequentes são identificados e, para cada um dos itens frequentes identificados, o ARMADA gera uma lista de índices, pela qual cada item frequente identificado é, então, relacionado com a localização de suas ocorrências nas transações da MDB. O ARMADA identifica padrões pela combinação de tais itens frequentes com os itens que os sucedem na MBD, utilizando para tal tarefa a lista de índices gerada. Ao final da execução do algoritmo, uma segunda varredura da BD é necessária para validar os padrões identificados.

Seja S um conjunto de objetos, propriedades, tendências ou estado que representam uma determinada situação. Um *estado* $s \in S$ se mantém durante um período de tempo [b, f), em que os elementos b e f representam, respectivamente, o instante inicial (em tempo) em que o estado s começa e o momento final em que o estado s deixou existir. Um *estado* s, de acordo com a conceituação e notação propostas por Höppner (2001), é representado pela terna (b, s, f), nomeada de *intervalo de estado*. Uma *sequência de estado* com base em S, por sua vez, é uma série intervalos de estado ordenados quanto a seus pontos temporais, (b_1, s_1, f_1), (b_2, s_2, f_2), ..., (b_n, s_n, f_n), ... em que $b_i \leq b_{i+1}$ e $b_i < f_i$. O formalismo proposto por Höppner exige que todo estado s, representado pelo intervalo de estado (b_i, s, f_i), seja maximal no sentido que não possa existir, na sequência de estado, qualquer outro intervalo de estado relacionado a s, (b_j, s, f_j), em que $[b_i, f_i)$ e $[b_j, f_j)$ se sobrepõem ou se encontram, como formalizado na Eq. (4.1).

$$\forall (b_i, s_i, f_i), (b_j, s_j, f_j), i < j \mid f_i \leq b_j \rightarrow s_i \neq s_j \qquad (4.1)$$

Na eventualidade da Eq. (4.1) não ser satisfeita para dois intervalos de estado, (b_i, s_i, f_i) e (b_j, s_j, f_j), ambos são substituídos por um único intervalo de estado, resultante da união de ambos, i.e., (min(b_i, b_j), s, max(f_i, f_j)).

Como descrito anteriormente, a formalização adotada por Höppner (2001) define um intervalo de estado pela terna (b_s, s, f_s), em que b_s representa o ponto no tempo que o estado s se inicia e f_s o ponto em que s termina, dessa forma o espaço temporal entre b_s e f_s é o espaço em que o estado s ocorre, i.e., o intervalo de duração de s. A Figura 4.1 apresenta uma representação pictórica do intervalo de estado (b_s, s, f_s) no espaço temporal.

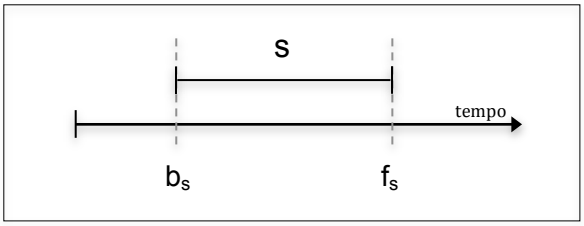
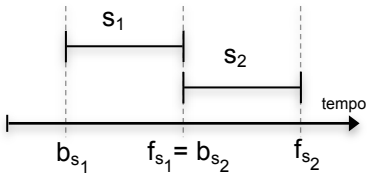
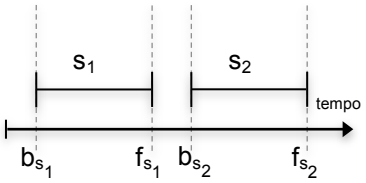
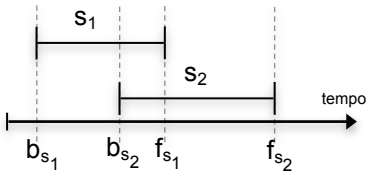
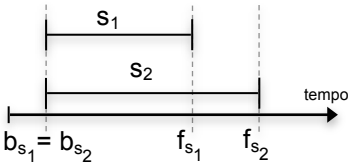
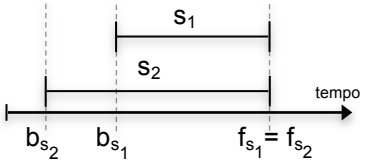
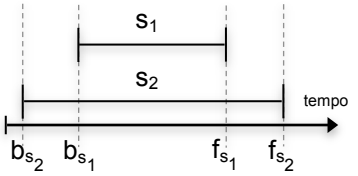
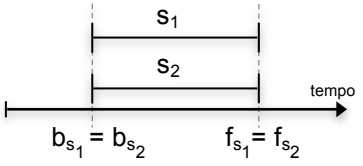


Figura 4.1 – Representação pictórica do intervalo de estado (b_s, s, f_s) no espaço temporal.

Para representar as relações entre os intervalos de estado, Höppner utiliza a AIA (descrita na Seção 4.2). Dessa forma, para cada par de intervalos de estado, há um total de 13 possíveis relações (ver Tabela 4.3). A relação *meets*, por exemplo, pode ser utilizada para representar a relação existente entre os intervalos de estado (b_i, s_i, f_i) e (b_j, s_j, f_j) se $f_i = b_j$, i.e., s_i termina no mesmo ponto no tempo em que s_j se inicia. Na Tabela 4.5 as 7 relações binárias básicas da lógica temporal de Allen (1983) são reescritas considerando a representação de Höppner (2001) para intervalos de estado (b_s, s, f_s). Considere RT o conjunto de relações temporais básicas da AIA, apresentadas na Tabela 4.3 e considere que sejam dados n intervalos de estado (b_i, s_i, f_i), $1 \leq i \leq n$. As posições relativas desses intervalos podem ser representadas por uma matriz $R_{n\times n}$, em que cada elemento da matriz i.e., R[i,j], descreve a relação entre o intervalo de estado i e o intervalo de estado j.

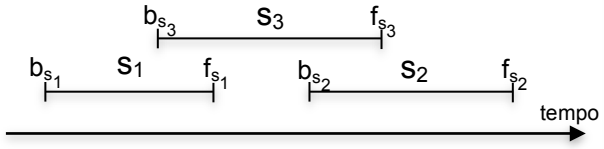
Formalmente, um *padrão temporal* ρ de tamanho n é definido pelo par (s, R), em que s: {1,..,n} $\rightarrow$ S é uma função injetora que associa um número inteiro i ($1 \leq i \leq n$) com o correspondente estado s_i e R[i,j] $\in$ RT informa a relação existente entre os intervalos [b_i,f_i) e [b_j,f_j), $1 \leq i, j \leq n$. A dimensão (quantidade de intervalos de estado) de um padrão ρ é denotada por dim(ρ). Se dim(ρ) = k, o padrão ρ é chamado de k-padrão.

Tabela 4.5 – Relações binárias básicas da lógica temporal de Allen (1983), expressas na representação de Höppner (2001), para intervalos de estado (b_s, s, f_s).

Relação temporal entre intervalos de estado	Condições que os pontos que definem cada um dos intervalos de estado devem satisfazer	Representação pictórica da relação temporal entre intervalos de estado
meets(s_1, s_2)	$b_{s_1} < b_{s_2}$, $b_{s_1} < f_{s_2}$, $f_{s_1} = b_{s_2}$, $f_{s_1} < f_{s_2}$	
before(s_1, s_2)	$b_{s_1} < b_{s_2}$, $b_{s_1} < f_{s_2}$, $f_{s_1} < b_{s_2}$, $f_{s_1} < f_{s_2}$	
overlaps(s_1,s_2)	$b_{s_1} < b_{s_2}$, $b_{s_1} < f_{s_2}$, $f_{s_1} > b_{s_2}$, $f_{s_1} < f_{s_2}$	
starts(s_1, s_2)	$b_{s_1} = b_{s_2}$, $b_{s_1} < f_{s_2}$, $f_{s_1} > b_{s_2}$, $f_{s_1} < f_{s_2}$	
finishes(s_1, s_2)	$b_{s_1} > b_{s_2}$, $b_{s_1} < f_{s_2}$, $f_{s_1} > b_{s_2}$, $f_{s_1} = f_{s_2}$	
during(s_1,s_2)	$b_{s_1} > b_{s_2}$, $b_{s_1} < f_{s_2}$, $f_{s_1} > b_{s_2}$, $f_{s_1} < f_{s_2}$	
equal(s_1, s_2)	$b_{s_1} = b_{s_2}$, $b_{s_1} < f_{s_2}$, $f_{s_1} > b_{s_2}$, $f_{s_1} = f_{s_2}$	

A Figura 4.2 apresenta um exemplo de uma sequência de intervalos de estado contendo 3 intervalos de estado, com suas respectivas relações (de acordo com a AIA) representadas na matriz R_{3x3} associada. Assim como definido anteriormente, uma *sequência de estado* é uma sucessão de intervalos de estado $(b_1, s_1, f_1), (b_2, s_2, f_2), ..., (b_n, s_n, f_n), ...$ em que $b_i \leq b_{i+1}$ e $b_i < f_i$. Portanto a sequência descrita na Figura 4.2 pode ser expressa da forma $S = <(b_{s_1}, s_1, f_{s_1})(b_{s_3}, s_3, f_{s_3})(b_{s_2}, s_2, f_{s_2})>$ dado que $b_{s_1} < f_{s_1}$, $b_{s_2} < f_{s_2}$, $b_{s_3} < f_{s_3}$ e $b_{s_1} \leq b_{s_3}$ e $b_{s_3} \leq b_{s_2}$.

Sequência de intervalos de estado:



Matriz das relações temporais ($R_{3\times3}$):

$eq(s_1,s_1)$	$b(s_1,s_2)$	$o(s_1,s_3)$
$a(s_2,s_1)$	$eq(s_2,s_2)$	$io(s_2,s_3)$
$io(s_3,s_1)$	$o(s_3,s_2)$	$eq(s_3,s_3)$

Legenda (eq: equal, a: after, b: before, o: overlaps, io: is_overlaped_by)

Figura 4.2 – Exemplo de uma sequência de três intervalos de estado e a matriz $R_{3\times3}$ associada.

O exemplo mostrado na Figura 4.2, pode ser utilizado para identificar o padrão temporal ρ de tamanho 3, que pode ser definido pelo par (s, R), em que s: {1, 2, 3} $\rightarrow$ {s_1, s_2, s_3} e R é a matriz $R = \begin{pmatrix} eq & b & o \\ a & eq & io \\ io & o & eq \end{pmatrix}$.

A leitura da matriz R é feita selecionado uma de suas linhas (i) e uma de suas colunas (j). O elemento da matriz R que se encontra na interseção da linha i e coluna j (i.e., R[i,j]) representa a relação existente entre o intervalo de estado si e o intervalo de estado s_j (i,j=1,...,3). Por exemplo, o elemento R[2,3] descreve a relação binária, em termos da AIA, entre os intervalos de estado s_2 e s_3, ou seja, (b_{s_2}, s_2, f_{s_2}) *is_overlaped_by* (b_{s_3}, s_3, f_{s_3}). Nota-se, ainda, que a relação expressa representada na posição R[i,j] é sempre a inversa daquela representada na posição R[j,i].

Naturalmente, muitos conjuntos de intervalos de estado são associados a um mesmo padrão temporal; quando há múltiplas ocorrências de um mesmo padrão,

admite-se que o conjunto de intervalos $\{(b_i, s_i, f_i) \mid 1 \leq i \leq n\}$ é uma instância do padrão temporal (s, R).

Um padrão temporal de tamanho n não tem um intervalo de duração próprio, porém seu espaço temporal pode ser representado pelo espaço temporal de seus intervalos de estado, i.e., o intervalo de tempo compreendido entre o ponto inicial do primeiro intervalo de estado (b_{s_1}) e o ponto final do ultimo intervalo de estado (f_{s_n}) do padrão. O espaço temporal dos padrões, TP(S), sobre o conjunto de todas as possibilidades de intervalos de estado, S, é informalmente definido como o espaço temporal de todos os padrões temporais válidos de dimensão (quantidade de intervalos de estado) arbitrária.

Informalmente, um padrão $\rho_\alpha = (s_\alpha, R_\alpha)$ é sub-padrão de $\rho_\beta = (s_\beta, R_\beta)$ se ρ_α pode ser obtido a partir da remoção de alguns estado (juntamente com seus respectivos relacionamentos) do padrão ρ_β. Considere novamente o exemplo da Figura 4.2 com o padrão ρ renomeado como ρ_β. Note que é possível identificar o sub-padrão $\rho_\alpha = (s_\alpha, R_\alpha)$, em que s_α:{1, 2} → {s_1, s_3} e R_α é a matriz $R = \begin{pmatrix} \text{eq} & \text{o} \\ \text{io} & \text{eq} \end{pmatrix}$ a partir da remoção do estado s_2, com todos seus respectivos relacionamentos, do padrão ρ_β.

A seguir é apresentada a definição de ordem parcial ($\sqsubseteq$), definida sobre o conjunto de padrões temporais (TPS), extraída de (HÖPPNER, 2001). Um padrão temporal (s_α, R_α) é dito sub-padrão de (s_β, R_β), i.e., $(s_\alpha, R_\alpha) \sqsubseteq (s_\beta, R_\beta)$ se $\dim(s_\alpha, R_\alpha) \leq \dim(s_\beta, R_\beta)$ e existe uma função injetora π: {1,.., $\dim(s_\alpha, R_\alpha)$} → {1,.., $\dim(s_\beta, R_\beta)$} que satisfaz a Eq. (4.2).

$$\forall i,j \in \{1, .., \dim(s_\alpha, R_\alpha)\}, R_\alpha[i,j] = R_\beta [\pi(i), \pi(j)] \qquad (4.2)$$

Uma relação definida sobre um conjunto é uma relação de ordem parcial se for reflexiva, transitiva e antissimétrica (a relação de continência entre dois conjuntos, por exemplo, é uma relação de ordem parcial). Pode ser observado que a relação $\sqsubseteq$ (sub-padrão) satisfaz as seguintes propriedades:

a) é reflexiva, pois $(s_\alpha, R_\alpha) \sqsubseteq (s_\alpha, R_\alpha)$;

b) é transitiva, visto que se é possível afirmar que se $(s_\alpha, R_\alpha) \sqsubseteq (s_\beta, R_\beta)$ e $(s_\beta, R_\beta) \sqsubseteq (s_\Upsilon, R_\Upsilon)$, naturalmente $(s_\alpha, R_\alpha) \sqsubseteq (s_\Upsilon, R_\Upsilon)$;

c) a relação $\sqsubseteq$, entretanto, não é antissimétrica, uma vez que pode ocorrer que o padrão $(s_\alpha, R_\alpha) \sqsubseteq (s_\beta, R_\beta)$ e $(s_\beta, R_\beta) \sqsubseteq (s_\alpha, R_\alpha)$ sem que $(s_\alpha, R_\alpha) = (s_\beta, R_\beta)$.

Um padrão temporal não necessariamente deve ser constituído por intervalos de estado dispostos de maneira ordenada. Portanto é possível afirmar que a permutação dos intervalos de estado contidos em um padrão temporal não altera a semântica do mesmo. Tal fato permite a possibilidade de considerar a relação de equivalência($\equiv$) entre padrões, ou seja, $(s_\alpha,R_\alpha) \equiv (s_\beta,R_\beta) \Leftrightarrow (s_\alpha,R_\alpha) \sqsubseteq (s_\beta,R_\beta) \wedge (s_\beta,R_\beta) \sqsubseteq (s_\alpha,R_\alpha)$. Como se sabe, uma relação de equivalência induz uma partição no conjunto sobre o qual está definida e cada bloco da partição caracteriza uma classe de equivalência em que padrões a ela pertencentes têm necessariamente a mesma dimensão (caso contrário não estariam relacionados pela relação $\equiv$).

Considere agora a fatoração do espaço temporal de todos os padrões válidos, dada por $\left(TP(S)/_{\equiv}, \sqsubseteq/_{\equiv}\right)$, em que o conjunto formado pelas classes de equivalência induzidas pela relação $\equiv$ definida sobre TP(S) é ordenado pela relação de ordem parcial dada por $\sqsubseteq$. Dessa forma é possível afirmar que $\sqsubseteq/_{\equiv}$ é, também, antissimétrica e, portanto, é possível considerar a ordem parcial em grupos de padrões temporais.

Para simplificar a notação de padrões temporais, Höppner usa a normalização de padrões temporais a partir do espaço temporal dos padrões, TP(S). O subconjunto $NTP(S) \subseteq TP(S)$ é o conjunto normalizado de padrões temporais, em que NTP(S) contém um elemento para cada uma das classes de equivalência descritas acima.

A técnica, basicamente, consiste em ordenar os intervalos de estado que definem o padrão, no tempo, por meio de índices crescentes. Dada uma ordenação de intervalos de estado, um padrão temporal $\rho_1 = (s,R)$ é dito estar em sua forma normalizada se para todo par de intervalos de estado $(b_i,s_i,f_i) \in \rho_1$, $(b_j,s_j,f_j) \in \rho_1$, tal que $1 \leq i \leq \dim(\rho_1)$ e $i < j \leq \dim(\rho_1)$, as seguintes condição são válidas:

a) $s_i = s_j$. Dois intervalos de estado associados ao mesmo estado implica a satisfação da Eq. (4.1) de intervalos de estado maximais e, portanto, existe um intervalo de tempo entre o final de s_i e o início de s_j, caso contrario ambos intervalos de estado teriam sido substituídos por um intervalo único

representando a concatenação de os ambos intervalos de estado. Consequentemente, as duas únicas relações da AIA que podem ser utilizadas para representar a relação binária entre s_i e s_j são R[i,j] ∈ {*before, after*}. Para preservar a ordem temporal, opta-se pela relação *before*.

b) $s_i \neq s_j$. Nos casos em que os intervalos de estado não representam o mesmo estado, duas possibilidades podem ser analisadas:, dependendo dos intervalos de estado possuir:

(1) tempos iniciais distintos ($b_{s_i} \neq b_{s_j}$) – uma ordenação por ordem crescente de tempo inicial é feita; nesse caso, R[i,j] ∈ {contains, is_finished_by, overlaps, meets, before}; ou

(2) tempos iniciais iguais ($b_{s_i} = b_{s_j}$) – dessa forma R[i,j] ∈ {equals, starts, is_started_by}. Se ambos intervalos de estado são idênticos ($b_{si} = b_{sj}$ e $f_{si} = f_{sj}$) a ordenação é feita pelo nome do estado, lexicograficamente. Se os tempos finais são distintos ($f_{si} \neq f_{sj}$) é utilizado R[i,j] = is_started_by, para garantir que o intervalo de estado (b_i,s_i,f_i) termine antes do intervalo de estado (b_j,s_j,f_j).

Exemplo Didático

Considere os padrões temporais: (1) (s_a, R_a), em que s_a:{1, 2, 3} → {s_1, s_2, s_3} e R_a é a matriz $R_a = \begin{pmatrix} eq & a & io \\ b & eq & io \\ o & o & eq \end{pmatrix}$ e (2) (s_b, R_b), em que s_b:{1, 2, 3} → {s_1, s_3, s_2} e R_b é a matriz $R_b = \begin{pmatrix} eq & io & a \\ o & eq & o \\ b & io & eq \end{pmatrix}$. Nota-se que ambos possuem o mesmo tamanho, i.e., 3 intervalos de estado cada um e que um é sub-padrão do outro. Assim como visto anteriormente, é possível permutar os elementos do padrão temporal sem que se altere sua semântica, portanto ao permutar os elementos s_3 e s_2 do padrão temporal (s_b, R_b) constata-se que existe a propriedade antissimétrica entre (s_a, R_a) e (s_b, R_b). Consequentemente, dessa forma, pela definição descrita por Höppner (2001) os padrões temporais são equivalentes e, portanto, representam instâncias de uma mesma classe de equivalência.

Considere ainda que (s_a, R_a) e (s_b, R_b) são parte do conjunto de padrões temporais, P = {(s_a, R_a), (s_b, R_b), (s_c, R_c), (s_d, R_d), (s_e, R_e), (s_f, R_f), (s_g, R_g), (s_h, R_h), (s_i, R_i), (s_j, R_j), (s_k, R_k).}, que tem seu espaço temporal definido por TP(S). Da mesma forma, como no passo descrito para os padrões (s_a,R_a) e (s_b,R_b), é suposto (para este exemplo) que (s_a,R_a) ≡ (s_b,R_b) ≡ (s_c,R_c), (s_d,R_d) ≡ (s_j,R_j), (s_e,R_e) ≡ (s_f,R_f) ≡ (s_g,R_g) ≡ (s_h,R_h) e (s_i,R_i) ≡ (s_k,R_k). De acordo com as equivalências identificadas, é possível, portanto, observar que o conjunto TP(S) pode ser particionado em 4 classes de equivalência, assim como apresentado na Tabela 4.6.

Tabela 4.6 – Classes de equivalência identificadas em TP(S).

Classes de equivalência em TP(S)	Padrões nas classes de equivalência
1	{(s_a, R_a), (s_b, R_b), (s_c, R_c)}
2	{(s_d, R_d), (s_j, R_j)}
3	{(s_e, R_e), (s_f, R_f), (s_g, R_g), (s_h, R_h)}
4	{(s_i, R_i), (s_k, R_k)}

Com a tarefa de partição do conjunto TP(S) em classes de equivalência, o próximo passo é selecionar um padrão de cada uma das classes de equivalência com a finalidade de formar o conjunto NTP(S), de padrões temporais normalizados; qualquer um dos padrões temporais da mesma classe de equivalência que for selecionado nesta etapa apresenta os mesmos intervalos de estado e os mesmos relacionamentos entre os intervalos de estado. O que pode divergir entre os padrões da mesma classe de equivalência é apenas a ordem em que os intervalos de estado aparecem no padrão.

Assim como descrito anteriormente, o processo de normalização de padrões temporais consiste em ordenar os intervalos de estado do padrão. Ao final da execução de tal processo, o conjunto NTP(S) de intervalos de estado normalizados é então formado.

Revendo a Figura 4.2, os intervalos de estados apresentados nela podem ser exemplos de um padrão temporal normalizado. Considere os intervalos de estado (b_{s_1}, s_1, f_{s_1}), (b_{s_2}, s_2, f_{s_2}) e (b_{s_3}, s_3, f_{s_3}), é possível notar que ambos estão ordenados quanto a seus pontos de início ($b_{s_1} \leq b_{s_3} \leq b_{s_2}$), ponto de término ($f_{s_1} \leq f_{s_3} \leq f_{s_2}$) ou nome do estado ($s_1 \leq s_3 \leq s_2$), i.e., normalizados. É possível ver ainda que, o 2-padrão ρ_1 = <((b_{s_1}, s_1, f_{s_1})(b_{s_2}, s_2,f_{s_2})> pode ser obtido a partir da remoção do estado

s_3 (e seus respectivos relacionamentos a (b_{s_1}, s_1, f_{s_1}) e (b_{s_2}, s_2, f_{s_2})) do 3-padrão ρ_2 = $<(b_{s_1}, s_1, f_{s_1})(b_{s_3}, s_3, f_{s_3})(b_{s_2}, s_2, f_{s_2})>$, portanto $\rho_1 \sqsubseteq \rho_2$, i.e., ρ_1 é um sub-padrão de ρ_2.

O algoritmo ARMADA utiliza-se da definição de Höppner (2001) para padrões temporais normalizados, dessa forma são necessárias apenas 7 das 13 relações binárias propostas por Allen (1983) para expressar o relacionamento entre os intervalos de estado do padrão temporal. São elas: before (b), meets (m), overlaps (o), is_finished_by (fb), contains (c), equals (eq) e starts (s).

Seja $\rho = <(b_1, s_1, f_1)(b_2, s_2, f_2)...(b_n, s_n, f_n)>$ o padrão temporal, onde $b_i \leq b_{i+1}$ e $b_i < f_i$. O espaço de tempo entre o ponto final do intervalo de estado s_i e o ponto de início do s_{i+1} é chamado *gap*(s_i, s_{i+1}). Considerando dois intervalos de estado i e j (para j = i + 1), o cálculo do *gap*(i, j) é feito pela diferença entre o ponto de início de j e o ponto final de i, ou seja, *gap*(i, j) = $b_j - f_i$.

O maior espaço possível entre os intervalos de estado de um padrão é definido como $\delta(\rho)$ = max{ *gap*(i, j) | i < j, i = 1,...,(n – 1) e j = 2, ..., n}. O parâmetro *maximum_gap* definido antes da execução do algoritmo ARMADA é uma constante que limita o maior espaço entre intervalos de estado que pode ser considerado em um padrão. Ao definir o parâmetro *maximum_gap*, o ARMADA somente considera que há relacionamento entre intervalos de estado que não tem um *gap* maior que o pré-estabelecido. Tal técnica é adotada para reduzir a quantidade de intervalos de estado no padrão, de forma a otimizar a identificação de padrões com aqueles que possuam relacionamentos mais relevantes entre seus intervalos de estado.

A Figura 4.3 apresenta a base de dados sintética (BD_3) temporal, adaptada de (WINARKO; RODDICK, 2007). A BD_3 é constituída por 17 registros de doenças referentes a 4 pacientes em uma clínica médica. Cada registro de doença contem também os tempos de inicio e termino, que representam o ponto em que a doença se iniciou e o ponto em que a doença terminou, respectivamente. A figura apresenta, ainda, um esquema pictórico representando visualmente a posição no espaço temporal de cada uma das doenças.

A base de dados BD_3 é percorrida pelo algoritmo ARMADA e cada um de seus registros é copiado para a memória interna (construindo a MDB) no formato de uma terna do tipo (b_i, s_i, f_i) (cada intervalo de estado representa um *itemset*). O ARMADA se utiliza de cada um dos *itemsets* gerados para formar sequencias de intervalos de estado referentes a cada um dos pacientes listados na BD_3. Por

exemplo, a sequência s_1 = <(2, s_1, 7)(5, s_5, 10)(5, s_2, 12)(16, s_4, 22)(18, s_3, 20)> é a sequência que contém todas as doenças observadas, com seus respectivos intervalos de duração, para o paciente 1 na BD_3. Durante o processo de criação da MDB o ARMADA também realiza o cálculo do valor de suporte para cada terna formada, identificando assim os *itemsets* unitários frequentes.

Paciente	Código Doença	Tempo Início	Tempo Final	Posição Relativa
1	s_1	2	7	
1	s_5	5	10	
1	s_2	5	12	
1	s_4	16	22	
1	s_3	18	20	
2	s_4	8	14	
2	s_3	10	13	
2	s_7	10	13	
2	s_6	15	22	
3	s_1	6	12	
3	s_2	7	13	
3	s_4	14	20	
3	s_3	17	19	
4	s_2	8	16	
4	s_1	18	21	
4	s_4	24	27	
4	s_5	25	28	

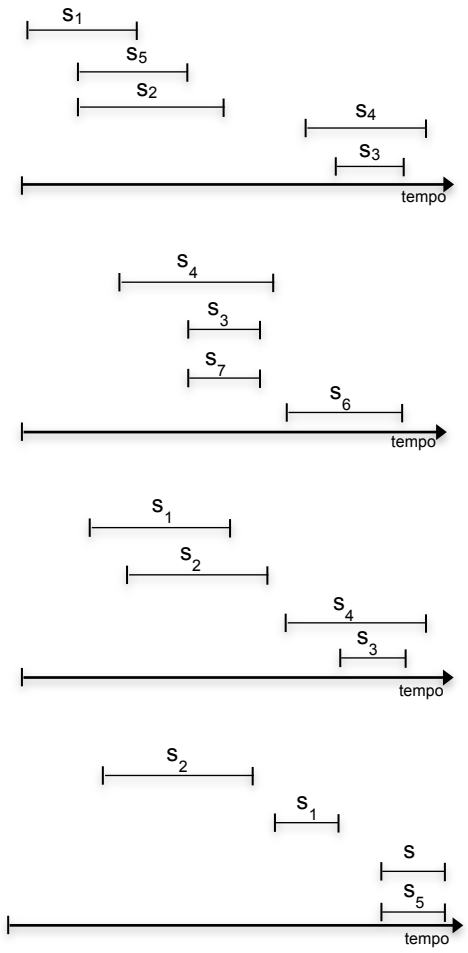


Figura 4.3 – Base de dados BD_3, utilizada para apresentação do algoritmo ARMADA. Extraída de (WINARKO; RODDICK, 2007).

Assim como o MEMISP, o algoritmo ARMADA identifica os padrões frequentes na MDB com o auxílio de uma lista de índices gerada para cada *itemset* frequente identificado em sua primeira etapa, no caso do ARMADA, para cada intervalo de estado. A Figura 4.4 apresenta duas listas de índices geradas pelo ARMADA, a primeira (<s_1>-idx) para o padrão, de tamanho 1, formado pelo estado frequente s_1 e a segunda (<s_1s_2>-idx) gerada a partir da combinação do estado s_1

com seu sucessor, s_2, na MDB. A combinação de s_1 com s_2 resulta no padrão do tipo-1 $\rho = <s_1 s_2>$.

Assim como no MEMISP, a lista de índices do padrão temporal relaciona o padrão identificado com a posição na MDB em que o padrão ocorre. Estendendo esse conceito, o ARMADA incorpora à lista de índices criada mais um atributo, o intervalo de duração do estado (nesse caso, da doença), utilizado para identificar as relações da AIA que podem ser aplicadas aos intervalos de estado do padrão identificado.

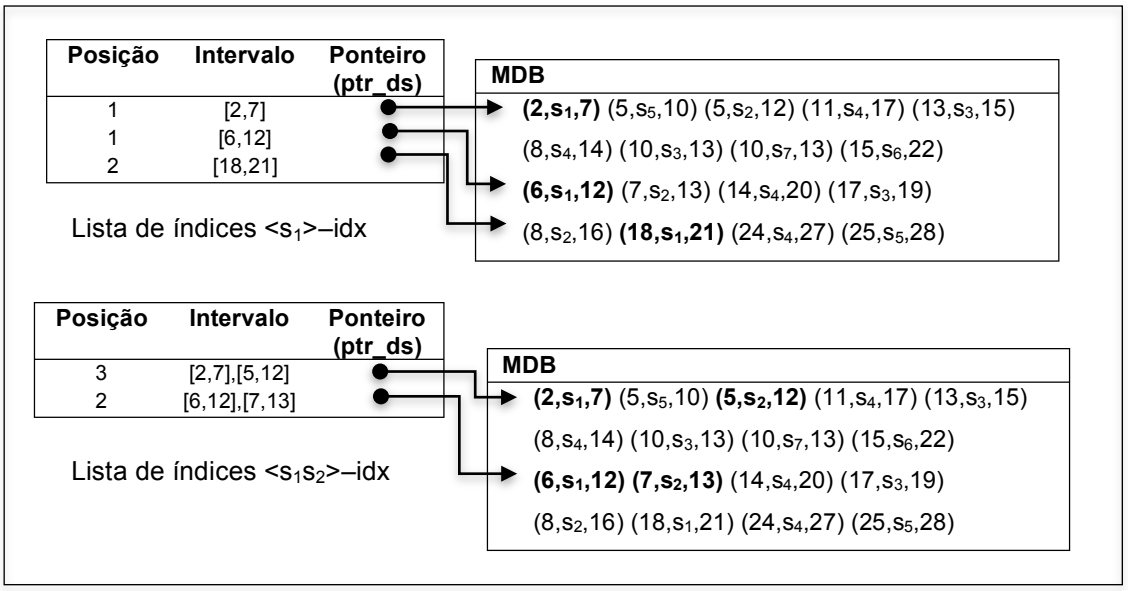


Figura 4.4 – Listas de índices <s_1>-idx e <$s_1 s_2$>-idx geradas a partir da execução do algoritmo ARMADA na base de dados BD_3.

Ao armazenar os pontos no tempo de cada *itemset* do padrão é possível identificar de maneira fácil as relações binárias da AIA que podem ser aplicadas para representar os relacionamentos entre os intervalos de estado do padrão. Por exemplo, o padrão temporal $\rho_2 = <s_1 s_2>$ apresentado pela Figura 4.4 possui em sua lista de índices, os pares de intervalos [2,7] e [5,12], o que significa que o estado s_1 ocorreu no intervalo de tempo durante os pontos 2 e 7 e o estado s_2 durante os pontos 5 e 12. Reescrevendo na forma de ternas é o mesmo que <(2, s_1, 7)(5, s_2, 12)>. Em termos da AIA, tal informação pode ser utilizada para identificar que o estado s_1 tem a relação binária de *overlaps* (de forma abreviada, o) com o estado s_2. De fato, é possível verificar que: (b_{s_1}= 2 < b_{s_2}= 5), (b_{s_1}= 2 < f_{s_2}= 12), (f_{s_1}= 7 > b_{s_2}= 5) e (f_{s_1}= 7< f_{s_2}= 12). O padrão $\rho_2 = <s_1 s_2>$ pode, também, ser representado pelo par (s, R), em que s:{1, 2} → {s_1, s_2} e R é a matriz R = $\begin{pmatrix} \text{eq} & \text{o} \\ \text{io} & \text{eq} \end{pmatrix}$.

Depois de descobertos todos os padrões temporais, o algoritmo ARMADA tem como sua próxima etapa de execução, gerar regras de associação entre os intervalos de estado que compõem os padrões temporais identificados na primeira etapa. Segundo define Winarko e Roddick (2007), as regras de associação são geradas para todos os pares (X, Y) de padrões temporais tal que X é um sub-padrão do k-padrão (tamanho k) normalizado Y, onde k > 1 e S = $\{s_1, s_2,...,s_k\}$ é a lista de estado que compõem o padrão Y.

Cada regra de associação é gerada de forma que o antecedente da regra é constituído por um sub-padrão X de Y tal que X é composto pela lista de estado S_i = $\{s_1,s_2,...,s_i\}$, i = 1,..,(k – 1) e o consequente da regra será composto pelos intervalos de estado restantes em Y (i.e., Y – X). Para um k-padrão Y é possível gerar no máximo k – 1 sub-padrões (X $\sqsubseteq$ Y); a regra é notada da forma R: X $\Rightarrow$ (Y – X).

O processo de geração de regras inicia pela criação do sub-padrão X com tamanho k – 1 (k – 1 intervalos de estado de Y), posteriormente decrementado até atingir o valor 1; caso a regra gerada não possua valor de confiança maior que o parâmetro pré-estabelecido *min_conf*, a regra não é considerada e os sub-padrões i < (k – 1) não são verificados, uma vez que as geras que seriam geradas possuiriam um baixo valor de confiança devido ao valor crescente do suporte de X (a medida em quem X contém menos intervalos de estado, naturalmente, torna-se mais frequente).

O algoritmo ARMADA, assim como descrito anteriormente, utiliza-se da mesma técnica empregada pelo algoritmo MEMISP para identificar padrões; a grande diferença entre ambos está, basicamente, na composição do *itemset*. Enquanto o MEMISP trata itemsets como listas de itens, o ARMADA estende o conceito do *itemset*, incorporando informações temporais. O processo de geração da lista de índices e mineração, entretanto, é executado da mesma forma. O processo de geração de regras de associação, empregado pelo ARMADA, pode ser visto no Algoritmo 4.1, procedimento, aqui chamado *generate_richer_temporal_assoc_rules,* executado de forma iterativa, percorrendo todos os padrões identificados, um a um para gerar um conjunto de regras de associação entre os intervalos de estado que compõem os padrões. O procedimento *generate_richer_temporal_assoc_rules* possui como entradas, (1) o conjunto F contendo todos os padrões identificados pela primeira etapa de execução do ARMADA e (2) o parâmetro pré-estabelecido *min_conf*, utilizado como limite inferior de confiança aplicado às combinações

possíveis para verificar se podem se tornar uma regra de associação, ou não. Como saída do procedimento, o conjunto All_R contem todas as regras de associação geradas a partir dos padrões temporais identificados.

Procedure *generate_richer_temporal_assoc_rules* (F,min_conf)
Entrada: F =conjunto que contém todos os padrões temporais identificados na primeira etapa do algoritmo ARMADA; min_conf = parâmetro que define se a regra de associação é considerada forte ou não;
Saída: All_R = conjunto que contém todas as regras de associação entre os padrões temporais indetificados.

```
1. begin
2.   All_R ← ∅
3.   tF ← |F|                          {tF contém a quantidade de padrões temporais em F }
4.   for i ← 1 to tF do
5.     begin
6.       p ← get_pattern(i,F)          {p recebe cada um dos padrões temporais contidos em F}
7.       for j ← 1 to nL do
8.         begin
9.           k ← |p|                   {k contém a quantidade de intervalos de estado do padrões p }
10.           while(k>1) do
11.           begin
12.             b ←get_subpadrao(p,k-1)        {b recebe cada um dos sub-padrões de p
13.                                             com k – 1  intervalos  de estado }
14.             conf ← suporte(p) / suporte(b)  {conf recebe o valor do cálculo de confiança da regra}
15.             if(conf ≥ min_conf) then        {se o valor de confiança supera min_conf
16.               begin                          a regra é considerada}
17.                 R ← gera_regra(p,b)         {a regra de associação do tipo R:  p ⇒ b é gerada}
18.                 All_R ← All_R ∪ R           e passa a compor o conjunto de todas regras temporais}
19.               end
20.             R ← ∅
21.             k ← k – 1
22.           end
23.         end
24.     end
25.   return All_R
26. end procedure
```

Algoritmo 4.1 – Pseudocódigo do procedimento *generate_richer_temporal_assoc_rules* utilizado para geração das regras de associação entre os padrões temporais identificados pelo algoritmo ARMADA.

A geração das regras de associação implementada no ARMADA tem vistas a identificar predições futuras, baseadas nos registros passados. Como por exemplo, ao identificar um padrão temporal $\rho = <(b_{s_1},s_1,f_{s_1})(b_{s_2},s_2,f_{s_2})(b_{s_3},s_3,f_{s_3})>$, é possível representar as relações existentes entre os intervalos de estados que o compõem, pela matriz $M = \begin{pmatrix} & s_1 & s_2 & s_3 \\ s_1 & eq & o & b \\ s_2 & io & eq & b \\ s_3 & a & a & eq \end{pmatrix}$.

Considere a matriz M_{ixj} que armazena as relações, segundo a AIA, referentes aos intervalos temporais do padrão identificado. É importante ressaltar duas características sobre a matriz de relações M gerada: (1) a diagonal principal sempre possui a relação *equal*, pois, de fato, representa uma relação de um intervalo temporal com ele mesmo e (2) a relação representada pelo elemento M[i][j] sempre será a relação inversa da representada pelo elemento M[j][i], portanto, é possível analisar as relações existentes entre os intervalos de estado que compõem um padrão temporal, apenas considerando os elementos acima, ou abaixo, da diagonal principal da matriz de relações M.

Portanto, é possível gerar uma regra de associação do tipo R: X ⇒ M, onde X é o sub-padrão de M, $X = \begin{pmatrix} & s_1 & s_2 \\ s_1 & eq & o \\ s_2 & io & eq \end{pmatrix}$. De fato, o padrão Y é frequente, pois é obtido a partir de um padrão frequente, portanto, se a regra R tem valor de confiança maior que *min_conf*, é possível interpretá-la como: *Se s_1 overlaps s_2 ocorrer então há uma grande possibilidade de s_1 before s_3 e s_2 before s_3 também ocorrerem*.

Capítulo 5

A EXTENSÃO MEMISP+AR: (*MEMORY INDEXING FOR SEQUENTIAL PATTERN MINING + ASSOCIATION RULES*)

Este capítulo descreve a implementação do sistema computacional S_MEMISP+AR, proposto neste trabalho, bem como as principais técnicas e abordagens que subsidiaram seu desenvolvimento. Também apresenta os primeiros experimentos e resultados obtidos em sua execução.

5.1 Considerações Iniciais

Como discutido no Capítulo 3, o algoritmo MEMISP, ao final do processamento, não contempla a geração de regras de associação, como faz o algoritmo Apriori, apresentado no Capítulo 2. Este capítulo apresenta e discute uma implementação do MEMISP, chamada algoritmo MEMISP+AR (Memory Indexing For Sequential Pattern Mining + Association Rules), que estende a proposta original, por meio da geração de regras de associação.

Como visto anteriormente, regras de associação são um importante recurso de sistemas de mineração de conhecimento, a partir de dados sequenciais. Podem ser abordadas como uma generalização de tais dados, com vistas a uso futuro.

O capítulo está organizado em cinco seções: inicialmente é apresentado o MEMISP+AR como uma extensão do algoritmo MEMISP, identificando as semelhanças e diferenças entre ambos e, também, discutindo um pseudo código em

alto nível de suas funcionalidades. Em seguida, são apresentadas as estruturas de dados utilizadas na implementação do algoritmo MEMISP+AR, como o sistema computacional S_MEMISP+AR. Na sequência, o exemplo de execução do MEMISP, apresentado na Seção 5.4, volta a ser revisitado, com vistas a mostrar o funcionamento do algoritmo MEMISP+AR, comparando seus resultados obtidos com os apresentados pelo estudo que descreve seu predecessor, ambos executados sob a mesma base de dados de sequências.

Em seguida é aprofundada a incorporação da técnica de geração de regras de inferência, foco principal desse capítulo, por meio de duas abordagens em duas subseções: (1) uma considerando sequências compostas por apenas um único *itemset* e (2) outra considerando sequências formadas por vários *itemsets*. Ambos exemplos são executados como testes observando as regras geradas e seus respectivos valores de confiança. A seção de geração de regras apresentada mostra a execução do algoritmo MEMISP+AR para ambas situações de forma que inicialmente identifica os padrões de uma base de dados de transações comerciais e em seguida gera regras a partir dos padrões identificados na primeira etapa.

Por fim, é descrito sucintamente algumas dificuldades identificadas e superadas no processo de implementação do S_MEMISP+AR.

5.2 Sobre o Algoritmo MEMISP+AR

O algoritmo MEMISP+AR é fundamentalmente subsidiado pelo MEMISP e contempla, ao término daquele, um procedimento para a extração de regras de associação, à semelhança daquela realizada pelo algoritmo Apriori (ver Capítulo 2). O Algoritmo 5.1 apresenta um pseudo código alto-nível do MEMISP+AR, considerando aquele descrito pelo Algoritmo 3.2 como fonte de identificação das sequências frequentes utilizadas neste.

Assim como seu antecessor, o MEMISP+AR faz uma única varredura na base de dados executando o cálculo do valor de suporte para todos os itens identificados enquanto efetua a cópia da base de dados para a memória do computador. Também contempla, se necessário, o uso da técnica *partition-and-validation*, quando a base sobrepor a capacidade da memória. Todos os *itemsets* frequentes são elaborados a

partir da indexação dos itens frequentes às sequências da base de dados e, assim por diante, as sequências frequentes são identificadas.

O MEMISP+AR incorpora o algoritmo MEMISP, permitindo a flexibilização na representação de *itemsets* que, no MEMISP+AR, não necessariamente precisam estar na ordem lexicográfica. É possível identificar padrões frequentes em sequencias que não possuem itens ordenados lexicograficamente, pois é mais importante a conservação da ordem de ocorrência dos itens dispostos nos *itemsets*. Tal flexibilização permite uma maior abrangência no número de bases de dados que podem ser utilizadas. O MEMISP+AR pode ser abordado como o algoritmo MEMISP acoplado a um processo de geração de regras, com a estrutura descrita na Figura 5.1. O processo de geração de regras, por sua vez, está apresentado em pseudocódigo em Algoritmo 5.1.

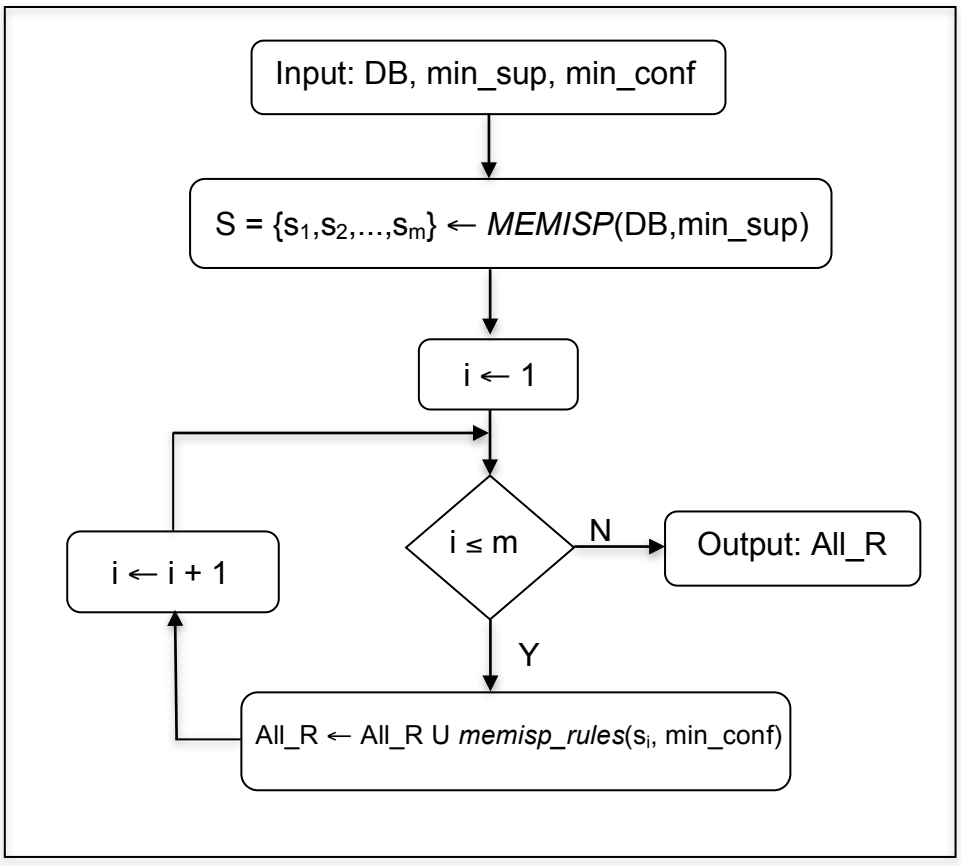


Figura 5.1 – Estrutura MEMISP+AR. Geração de regras a partir da execução do MEMISP.

Em linhas gerais, o procedimento *memisp_rules* (descrito pelo Algoritmo 5.1) gera, para cada uma das sequências frequentes identificadas pelo MEMISP, um conjunto associado de regras.

Cada *itemset* e_i ($e_i \in s$) dá origem a um conjunto de regras R_i. = { R_{i1}, R_{i2}, R_{ik}} por meio do procedimento *gera_regras_do_itemset*. Tal procedimento consiste em gerar todas as combinações possíveis dos itens que compõem o *itemset*, de forma que as regras geradas apresentem componentes formados por todas combinações de 1 a m – 1 itens (m, tamanho do *itemset* e_i). Tanto a parte antecedente quanto a consequente da regra sempre conservam a ordem de ocorrência dos itens no *itemset* analisado.

É importante ressaltar, entretanto, que o S_MEMISP+AR difere do Apriori (em sua primeira versão), pois foi idealizado e implementado de forma a admitir que a parte consequente das regras de associação geradas possua mais de um item, assim como visto em (HÖPPNER, 2001).

Procedure *memisp_rules(s, min_conf)*
Entrada: s = <$e_1e_2...e_N$> Sequência frequente; min_conf = limite mínimo de confiança;
Saída: All_R = {R_1,R_2,...R_k} *conjunto com todas as regras de associação consideradas* fortes, geradas a partir da sequência frequente identificada**.**

```
1  begin
2    All_R ← ∅
3    N ← qtd_itemsets(s)
4    for i ← 1 to N do
5    begin
6      R_i ← gera_regras_do_itemset(e_i)
7      qtd_regras ← num_regras_geradas(R_i)
8      for j ← 1 to qtd_regras do
9      begin
10       conf ← calcula_confianca_regra(R_ij)
11       if (conf ≤ min_conf) then
12         remove_regra(R_ij, R_i)
13     end
14     All_R ← All_R ∪ R_i
15   end
16   return(All_R)
17 end procedure
```

Algoritmo 5.1 – Procedimento *memisp_rules*. Geração de regras de associação a partir do conjunto de sequências frequentes obtida pelo MEMISP.

Uma vez finalizada a construção de cada um dos conjuntos de regras R_i, i=1,...,N, o valor de confiança de cada uma delas é calculado e, caso seja maior ou igual ao valor pré-estabelecido *min_conf*, a regra é considerada, caso contrário é descartada. Todas as regras que permanecerem no conjunto R_i, i=1,...,N, são adicionadas ao conjunto final de regras de associação consideradas i.e., o conjunto All_R que, ao final da execução do procedimento *memisp_rules*, irá conter todas as regras de associação geradas.

5.3 Implementação do MEMISP+AR - Estruturas de Dados e Procedimentos Utilizados

O algoritmo MEMISP+AR foi implementado em linguagem C++ como um sistema computacional interativo, chamado de sistema S_MEMISP+AR. No S_MEMISP+AR as estruturas de dados utilizadas para armazenar os itens, *itemsets* e sequências contam com recursos de alocação dinâmica de memória, disponibilizados pela linguagem, o que otimiza o uso de memória durante o processamento, evitando alocações desnecessárias.

A base de dados (MDB), armazenada na memória interna do computador (RAM), é criada usando um ponteiro que aponta para a cabeça de uma lista de sequências, alocadas dinamicamente, e copiadas para a memória, a partir da base de dados original fornecida (BD). O encadeamento da lista de sequências é feito via ponteiro.

Parafraseando a Definição 3.7, formalizada neste trabalho e inspirada por sua descrição informal apresentada por Lin e Lee (2002), " Dado um padrão sequencial ρ, o conjunto de índices ρ-idx é uma relação definida no domínio I x MDB, i.e., ρ-idx $\subseteq$ I x MDB. A relação ρ-idx, associada ao padrão sequencial ρ, é pois formada por pares <x,y> tal que x $\in$ I, y $\in$ MDB e x indica a posição (em y), do término de ρ em sua primeira ocorrência em y."

Considere que MDB = $\{C_1, C_2, ..., C_k\}$. Dado um padrão ρ, com o objetivo de definir a relação ρ-idx, é gerado um par <pos,ptr_ds> para cada sequência C_i ($1 \leq i \leq k$) que apresentar o padrão ρ; no par, ptr_ds é um ponteiro para a sequência que contém ρ e pos é a posição do término de ρ na sequência considerada.

Assim como a MDB é representada por um ponteiro, a lista de índices é também representada por um ponteiro, que aponta para a cabeça de uma lista de pares <pos,ptr_ds>. A Figura 5.2 mostra representação pictórica geral das estruturas de dados adotadas na implementação do S_MEMISP+AR para armazenar os itens, *itemsets* e sequências na MDB. Nela é possível observar a composição de cada estrutura, modelada especificamente para armazenar cada tipo de dado.

Os itens são armazenados pelo tipo de dado *tItem* que é composto pelas informações de valor de suporte do item (i_suporte), dado armazenado (it, do tipo string) e um ponteiro (prox), que referencia qual o próximo item em um *itemset*.

Itemsets são caracterizados pelo tipo de dado *tItemset*, que pode ser abordado como uma estrutura de dados que armazena: (1) tamanho do *itemset* (it_tamanho), i.e., quantidade de itens que o compõem; (2) suporte do *itemset* (it_suporte); (3) o ponteiro (prox) para o próximo *itemset* na sequência e finalmente, (4) o ponteiro (it_cabeca), que aponta o primeiro item de uma lista de itens que participam do *itemset*. O tipo tSequencia armazena as sequências e tem uma estrutura próxima àquela do tItemset: (1) o tamanho da sequência (s_tamanho); (2) o suporte da sequência (s_suporte); (3) o ponteiro (s_prox) que possibilita o encadeamento de sequências (útil quando representando a MDB) e, finalmente (4) o ponteiro (s_cabeca) que aponta para o primeiro *itemset* de uma lista de *itemsets* que compõem uma sequência.

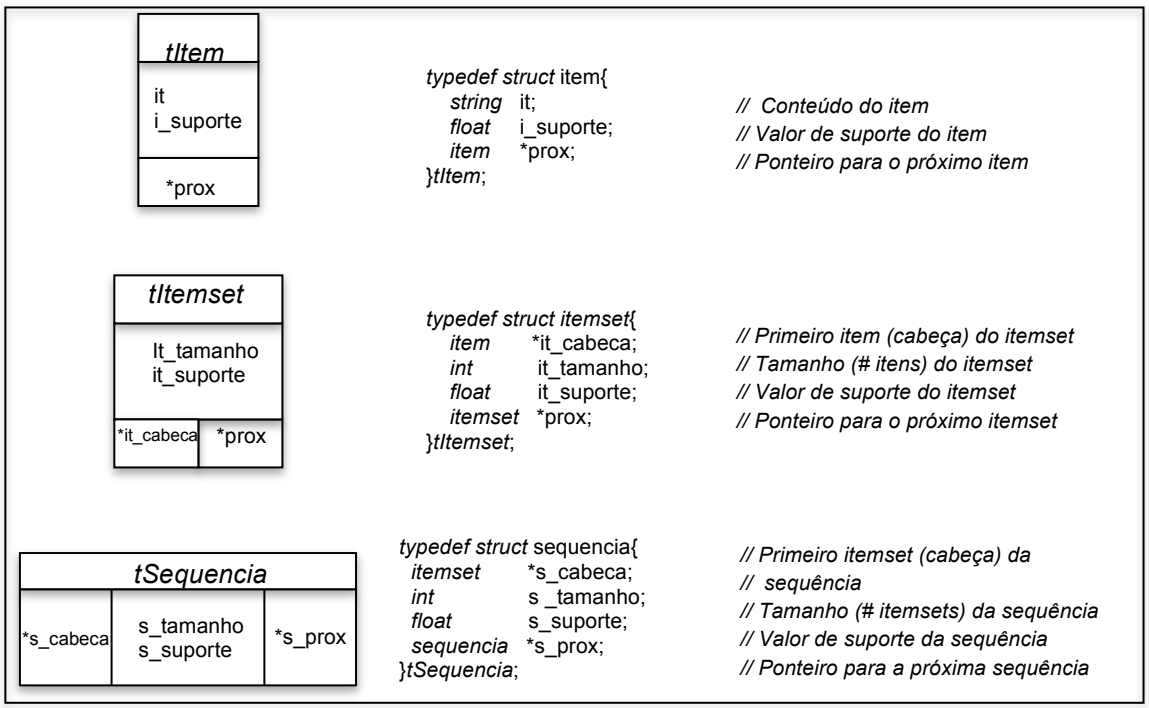


Figura 5.2 – Representação das estruturas de dados que armazenam os itens, *itemsets* e sequências.

A sequência s = <(a)(a,c,d)> pode ser vista na representação pictórica apresentada na Figura 5.3. Nota-se que a estrutura que armazena a sequência, além de armazenar o tamanho e o valor de suporte da sequência, não armazena a sequência propriamente dita. Em verdade, a estrutura da sequência s é um ponteiro

para uma lista de *itemsets*. De maneira similar, a estrutura que representa um *itemset* não armazena os itens mas sim um ponteiro para o primeiro item da lista que define o *itemset*. Os dados que definem a sequência (itens) estão armazenados, um a um, em estruturas isoladas para cada um dos itens que compõem um *itemset* da sequência.

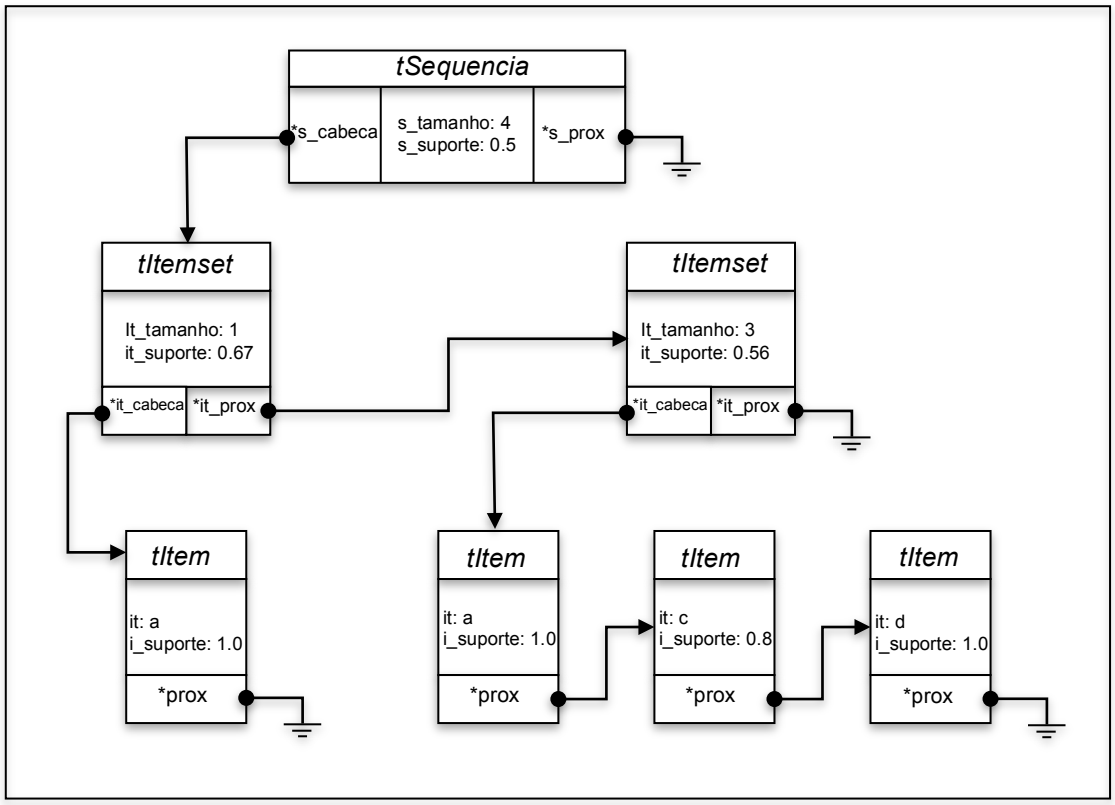


Figura 5.3 – Estruturas de dados que armazenam a sequência s = <(a)(a,c,d)>.

Como comentado, a estrutura que define uma sequência utilizada no S_MEMISP+AR armazena um ponteiro para outra sequência, que pode ser utilizada em um possível encadeamento. A MDB (base de dados na memória RAM do computador) é definida utilizando esse recurso; a estrutura que define a MDB contém: (1) um valor inteiro que armazena a quantidade de sequências que compõem a MDB (qtd_seq) e (2) o ponteiro (cabeca) que faz referência à primeira sequência de uma lista de sequências que, juntas, compõem a MDB. A Figura 5.4 ilustra a MDB de forma genérica. O ponteiro (cabeca) indica s_1 como sendo a primeira sequência da MDB e o ponteiro (s_prox) da sequência s_1 referencia a próxima sequência da MDB, s_2 e, assim sucessivamente; o encadeamento das sequências é responsável por definir o tamanho da MDB e a ordem das sequências.

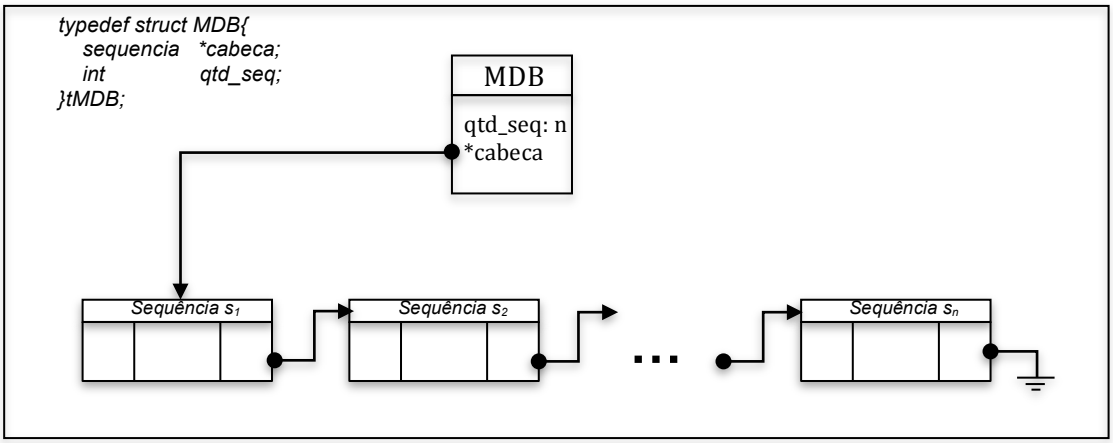


Figura 5.4 – Esquematização da estrutura de dados que armazena a MDB, base de dados na memória RAM.

5.4 MEMISP+AR × MEMISP

Para mostrar o funcionamento do algoritmo MEMISP+AR, à semelhança do que foi feito com o MEMISP, inicialmente foi executado com uma base de dados de sequências (BD_2) composta por 6 sequências, cada uma delas contendo apenas literais alfabéticos ordenados lexicograficamente, como pode ser observado na base de dados de sequências descrita pela Tabela 5.1. Assim como visto anteriormente, tal base de dados está descrita e foi usada em (LIN; LEE, 2002). Neste e nos próximos exemplos, o algoritmo foi programado para fazer o acesso à uma base de dados armazenada em arquivo texto (.txt ou .csv) e a transferir para memória. Tal decisão foi meramente uma opção de desenvolvimento e poderá ser facilmente readaptada para uma conexão direta com algum Sistema Gerenciador de Banco de Dados (SGBD).

Tabela 5.1 – BD_2: Base de Dados de Sequências, extraída de (LIN; LEE, 2002).

ID_seq	BD_2
C_1	<(a,d)(b,c)(a,e)>
C_2	<(d,g)(c,f)(b,d)>
C_3	<(a,c)(d)(f)(b)>
C_4	<(a,b,c,d)(a)(b)>
C_5	<(b,c,d)(a,c,e)(a)>
C_6	<(b,c)(c)(a,c,e)>

Como forma de validação da eficiência do algoritmo MEMISP+AR, seu resultado foi comparado com aquele apresentado em (LIN; LEE, 2002). A MDB da Figura 5.5 contém as 6 sequências de BD_2 transferidas para a memória, na representação discutida na Seção 5.2.

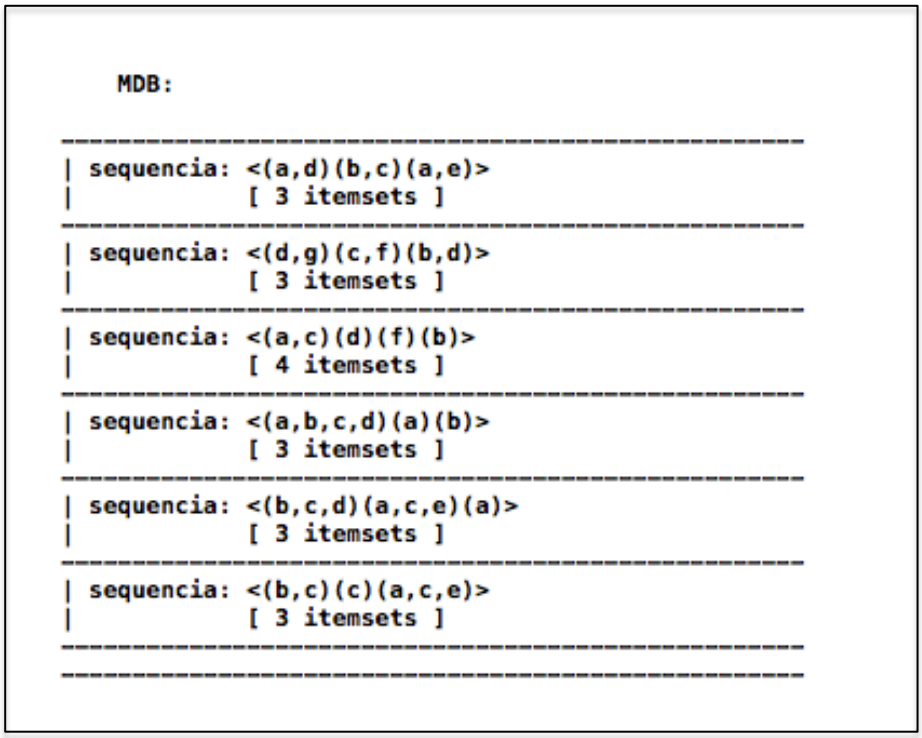


Figura 5.5 – MDB: Base de dados na memória interna do computador.

Considerando que o objetivo visa uma comparação entre resultados, assim como feito em (LIN; LEE, 2002), foi estabelecido o valor de *min_sup* = 0,5 (50%). Ao final da execução do MEMISP+AR as sequências frequentes encontradas, juntamente com os itens frequentes que as geraram, são apresentados na Figura 5.6; o resultado do MEMISP+AR coincide com aqueles apresentados na proposta original, como seria o esperado. Note que, nesse exemplo, o processo de geração de regras, que é o que diferencia o MEMISP+AR do MEMISP não foi contemplado, uma vez que é discutido na Seção 5.5.

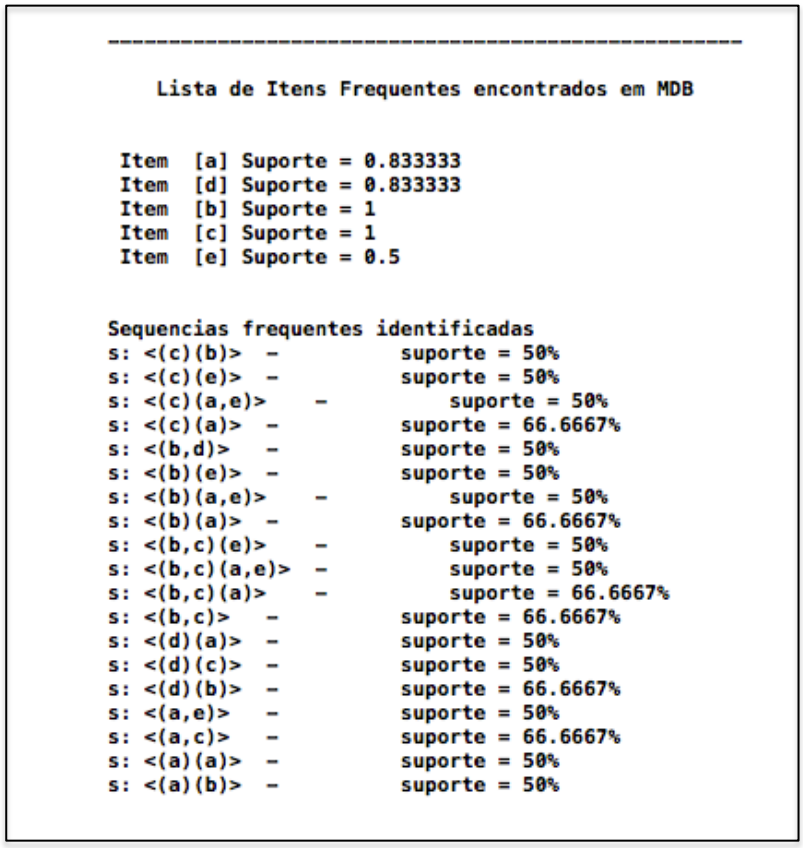


Figura 5.6 – Padrões frequentes identificados pelo MEMISP+AR (*min_sup* = 0,5).

5.5 MEMISP+AR - O Processo de Geração de Regras Associativas Considerando Duas Abordagens.

No que segue o processo de geração de regras associativas, implementado pelo MEMISP+AR, é discutido, considerando duas situações: (1) a primeira mais simplificada, em que sequências têm apenas um *itemset*, e (2) a segunda, em que admite-se cada sequência com vários *itemsets*. A justificativa de cada uma das escolhas é descrita com mais detalhes após a apresentação da base de dados utilizada. Como será visto, o processo de geração de regras é fortemente subsidiado pelo algoritmo Apriori. O algoritmo MEMISP+AR, até esse ponto do desenvolvimento (não contempla ainda o tratamento do aspecto temporal), pode ser abordado como uma evolução do MEMISP ao qual foi acoplado um processo de geração de regras de associação que, de certa forma, generaliza aquele proposto pelo Apriori.

5.5.1 Sobre o Processo de Identificação de Padrões do MEMISP+AR Considerando Sequências com um Único *Itemset* - Estudo de Caso I

No que segue, um exemplo que reflete uma situação real de geração de sequências de objetos é usada. A base de dados mostrada neste exemplo (BDOC) é apenas um recorte, constituído de 70 registros, da base de dados original (BDO), obtida junto à um supermercado em Osvaldo Cruz-SP; são registros relativos às vendas realizadas, durante o período de uma semana, pelo supermercado, via um determinado caixa. Para a obtenção da BDO, inicialmente foi selecionado, dentre os caixas disponibilizados no supermercado, um caixa específico (Ca). Todas as vendas registradas por Ca, durante o período de uma semana, constituíram o que, neste documento, está sendo referenciado por BDO.

A BDOC foi extraída da BDO segundo a seguinte metodologia: para cada dia da semana, uma amostra de 10 registros diários de venda naquele dia, realizadas via Ca, durante o período da manhã, foram selecionado manualmente. Ao final, a BDOC resultou em um conjunto com 70 registros de dados de venda de produtos, com 10 registros/dia da semana.

A BDOC original é constituída por dados brutos, sem que tenham sofrido qualquer procedimento de tratamento, com o objetivo de prepará-los para um processamento seguinte. O próximo passo do experimento foi pré-processar BDOC de maneira a adequa-lo para ser input do MEMISP+AR.

O pré-processamento consistiu em (1) renomear alguns produtos, de maneira a remover determinados tipos de caracteres, e.g., cedilha, brancos, acentos, etc...; (2) renomear determinados itens com um mesmo nome (e.g., vendas de produtos caracterizados como pão integral, pão sírio e pão francês, foram todas renomeadas como paes*;* processo semelhante sofreram muitos outros produtos, tais como biscoitos, salgados, etc.).

Após a renomeação dos itens, em cada registro de venda os itens foram ordenados alfabeticamente, como uma sequência com um único *itemset*, em que cada item da venda é um item do *itemset*. A opção pela adoção da abordagem de cada sequência ser constituída por um único *itemset* é devido ao fato de, nesse exemplo, buscar padrões frequentes em cada uma das vendas, independentemente dos dias ou horário em que tais vendas foram registradas. Dessa forma é mais intuitivo que a busca seja realizada em cada uma das vendas isoladamente. A

Tabela 5.2 apresenta um exemplo de cinco registros de vendas, escolhidos aleatoriamente da BDOC após ter sido submetida ao pré-processamento.

Tabela 5.2 – Trecho extraído da BDOC após pré-processamento.

Vendas	Sequências com um único *itemset* (produtos relativos a uma venda)
v_1	(condicionador,papelhigienico,pastadedentes,sabonete,xampu)
v_2	(carnebovina,carvao,cebola,cerveja,frango,paes,pepino,refrigerante,tomate)
v_3	(alface,bacon,banana,batatapalha,biscoitos,catchup,cereal,cerveja,fosforo,frango, iogurte,leite,leitecondensado,maionese,mamao,mortadela,ovos,presunto,queijo, sabonete,suco,xampu)
v_4	(adocante,banana,paes,presunto,queijo,refrigerante)
v_5	(achocolatado,acucar,arroz,biscoitos,catchup,cereal,chocolate,condicionador, desodorante,feijao,frango,leite,maionese,mostarda,paes,peitodeperu,queijo, requeijao,suco,xampu)

Para a execução do MEMISP+AR na BDOC foi estabelecido o valor *min_sup* = 0,5. Pela grande diversidade entre os itens das compras nenhum padrão frequente foi identificado, portanto o valor do *min_sup* foi decrementado, gradativamente, até que um primeiro padrão pudesse ser identificado, fato que aconteceu com *min_sup* = 0,25.

A Figura 5.7 apresenta a saída da execução do MEMIS+AR na BDOC, considerando *min_sup* = 0,25. Nela é possível observar os itens frequentes identificados, com seus respectivos suportes e a sequência frequente identificada (s= <(paes,refrigerante)>, com valor de suporte = 0,261). A subsequência frequente identificada foi elaborada a partir da combinação dos itens frequentes identificados com os demais itens que os sucedem na MDB. Assim como visto no Capítulo 3, que descreve o algoritmo MEMISP com seu processo de busca de padrões sequenciais e a indexação na base de dados, a combinação é feita observando a lista de índices (IndexSet) de cada item frequente identificado. O item é então combinado com todos os demais itens situados partir da posição (pos) do par (pos, ptr_ds) até o final da sequência. Para cada combinação é calculado o valor de suporte.

Ainda na Figura 5.7 é possível observar que vários outros itens foram identificados como frequentes; tais itens, entretanto, quando combinados a outros itens, não formaram *itemsets* com suporte superior a 0,25 (min_sup).

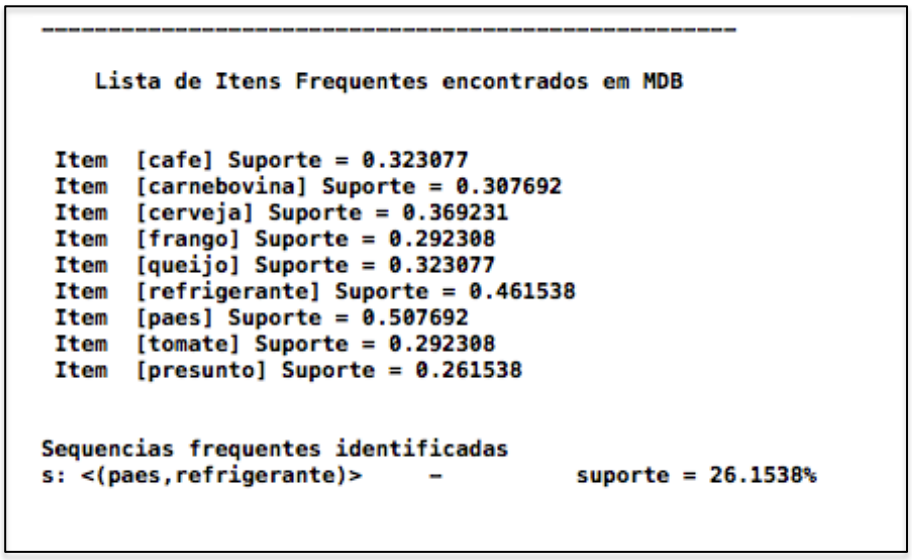


Figura 5.7 – Padrões frequentes identificados pelo MEMISP+AR na BDOC (*min_sup*=0,25).

Na execução do MEMISP+AR com *min_sup* = 0,25, s = <(paes,refrigerante)> foi a única sequência frequente identificada, tendo como suporte o valor 0,2615. A fim de identificar uma quantidade maior de padrões na MDB, execuções do algoritmo foram feitas com diferentes valores para *min_sup*; os resultados podem ser observados nas figuras 5.8 e 5.9.

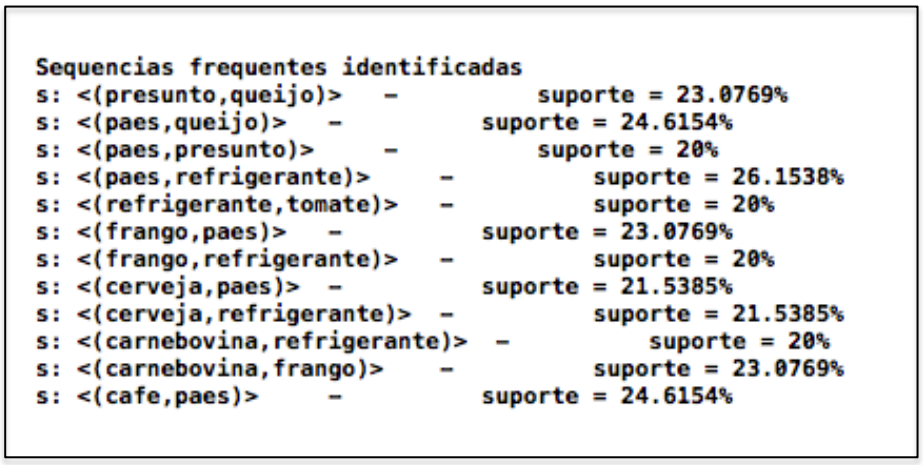


Figura 5.8 – Padrões frequentes identificados pelo MEMISP+AR na BDOC (*min_sup*=0,2).

Os resultados apresentados na Figura 5.8 mostram a saída do algoritmo MEMISP+AR, considerando *min_sup* = 0,2. O MEMISP+AR identifica uma quantidade substancialmente maior do que aquela identificada na execução anterior (*min_sup* = 0,25, Figura 5.7). É importante notar, entretanto, que todas as subsequências encontradas (mostradas na Figura 5.8) são constituídas por apenas dois itens, i.e., sequências de tamanho = 2. Na Figura 5.9 é possível observar a ocorrência de padrões identificados na MDB formados com mais de dois itens, como

é o caso de <(paes,presunto,queijo)> e <(carnebovina,frango,paes)>, com valores de suporte iguais a 0,18461 e tamanho 3.

```
Sequencias frequentes identificadas
s: <(presunto,queijo)>   -          suporte = 23.0769%
s: <(paes,queijo)>   -          suporte = 24.6154%
s: <(paes,presunto,queijo)>  -          suporte = 18.4615%
s: <(paes,presunto)>     -          suporte = 20%
s: <(paes,suco)>     -          suporte = 18.4615%
s: <(paes,refrigerante)>     -          suporte = 26.1538%
s: <(refrigerante,tomate)>   -          suporte = 20%
s: <(frango,paes)>   -          suporte = 23.0769%
s: <(frango,refrigerante)>   -          suporte = 20%
s: <(cerveja,paes)>  -          suporte = 21.5385%
s: <(cerveja,refrigerante)>  -          suporte = 21.5385%
s: <(carnebovina,paes)>  -          suporte = 18.4615%
s: <(carnebovina,refrigerante)>  -          suporte = 20%
s: <(carnebovina,frango,paes)>   -          suporte = 18.4615%
s: <(carnebovina,frango)>    -          suporte = 23.0769%
s: <(carnebovina,cerveja)>   -          suporte = 18.4615%
s: <(cafe,paes)>     -          suporte = 24.6154%
s: <(cafe,refrigerante)>     -          suporte = 18.4615%
s: <(cafe,margarina)>    -          suporte = 18.4615%
```

Figura 5.9 – Padrões frequentes identificados pelo MEMISP+AR na BDOC (*min_sup*=0,18).

Assim como visto anteriormente, o algoritmo Apriori proposto e descrito em (AGRAWAL et al., 1994), implementa um processo de extração de regras de associação entre itens de uma transação. O algoritmo MEMISP+AR faz uso do Apriori como subsidio para implementar a técnica de extração de regras de associação, parte que estende o algoritmo e o difere do MEMISP. As regras são geradas a partir dos padrões identificados na MDB na primeira etapa de execução do MEMISP+AR. A incorporação do procedimento de geração de regras na implementação MEMISP+AR, nomeada *memisp_rules* é apresentada pelo Algoritmo 5.1, no qual é possível observar como entrada, uma sequência frequente *s* em que cada um dos seus *itemsets* será utilizado para gerar um conjunto de regras.

No próximo passo de execução do MEMISP+AR, é gerado um conjunto de regras de associação geradas pelo *memisp_rules*, utilizando como entradas um dos padrões identificados na BDOC considerando min_sup = 0,18. Foram identificados vários padrões frequentes (apresentados na Figura 5.9), dentre eles o padrão s = <(carnebovina,frango,paes)>, foi escolhido pra exemplificar a extração de regras de associação. Do padrão s selecionado, é gerado um conjunto de duas regras de associação, combinando seus itens entre si, de forma que conservem a ordem de ocorrência em cada lado da regra e que nem o antecedente nem o consequente

sejam vazios. Devido ao pré-processamento feito na base de dados, os itens estão dispostos em ordem lexicográfica e assim permanecem nas regras.

A Tabela 5.3 apresenta o conjunto de regras de associação geradas a partir do padrão s selecionado, bem como traz o valor calculado da confiança de cada uma delas e uma classificação se a regra é considerada forte ou não (i.e., tem valor de confiança maior ou igual ao valor min_conf pré-estabelecido). O cálculo da confiança da regra é feito pela Equação 2.1, apresentada no Capítulo 2, considerando um valor constante de limite inferior de confiança, para identificar a força de uma regra (min_conf) definido em 0,8.

Tabela 5.3 – Conjunto de regras de associação geradas a partir da sequência frequente s = <(carnebovina,frango,paes)>. As regras são apresentadas com seus respectivos valores de confiança e validação quando a regra de associação for considerada forte, admitindo-se um valor de min_conf = 0,8.

Regra	Confiança	Regra considerada?
carnebovina, frango ⇒ paes	$\mathbf{\frac{12}{15} = 0,8}$	**Sim**
carnebovina ⇒ frango, paes	$\frac{12}{20} = 0,6$	Não

É visto, nesse estudo de caso, que o valor de confiança (min_conf) necessariamente teve que ser alto para que uma regra de associação pudesse ser considerada.

Das combinações geradas a possíveis regras de associação, no caso apresentado, só uma é considerada uma regra de associação efetivamente, carnebovina,frango ⇒ paes com valor de confiança igual a 0,8. Tal valor de confiança associado à regra garante que em 80% dos registros da BD, em que os três produtos (carnebovina,frango e paes) foram encontrados nas vendas que compõem a BDOC, o item paes foi identificado, i.e., pode ser afirmado que há uma grande possibilidade de ocorrer o item paes toda vez que a sequencia carnebovina,frango for encontrada em uma venda.

5.5.2 Sobre o Processo de Identificação de Padrões do MEMISP+AR Considerando Sequências com Vários *Itemsets* - Estudo de Caso II

Da mesma forma como a BDOC, utilizada no exemplo apresentado, a $BDOC_2$ também foi extraída da BDO seguindo o seguinte critério: para cada dia da

semana, uma amostra de dez registros diários de venda naquele dia, realizadas via Ca, durante o período da manhã, manualmente. A $BDOC_2$ também foi pré-processada da mesma forma que a BDOC, seguindo as tarefas de: (1) renomear alguns produtos, de maneira a remover determinados tipos de caracteres, e.g., cedilha, brancos, acentos, etc...; (2) renomear determinados itens com um mesmo nome (e.g., vendas de produtos caracterizados como pão integral, pão sírio e pão francês, foram todas renomeadas como paes*;* processo semelhante sofreram muitos outros produtos, tais como biscoitos, salgados, etc.).

Após a tarefa de pré-processamento da $BDOC_2$, cada um dos registros de compras armazenados na $BDOC_2$ tiveram os itens que os compõem ordenados lexicograficamente. Dessa forma, cada compra é constituída por um *itemset* contendo todos os itens registrados por Ca naquela compra e cada sequência é constituída por dez *itemsets*, que representam a amostra do dia analisado, ou seja, um dia. Ao final da tarefa de pré-processamento, a $BDOC_2$ resultou em um conjunto com sete sequências contendo 10 *itemsets* cada uma, i.e., cada sequência refere-se às vendas registradas em um dia analisado. A Tabela 5.4 apresenta um trecho da $BDOC_2$ que representa o registro de um dia analisado, ou seja, uma sequência constituída por dez itemsets. Tal registro foi escolhido aleatoriamente da $BDOC_2$ após ter sido submetida ao pré-processamento.

Tabela 5.4 – Trecho extraído da $BDOC_2$ após pré-processamento.

Dia	**Sequências de *itemsets* referentes às compras de um dia analisado (cada itemset é uma compra registrada no dia)**
d_1	(cebola,cerveja,refrigerante,tomate)(adocante,cafe,margarina,paes)(detergente,esponja)(laranja,manga,rúcula,tomate)(carnebovina,carvao,cebola,cerveja,frango,paes,pepino,refrigerante,tomate)(acucar,biscoito,cafe,cerveja,cha,maionese,oleo,paes,presunto,queijo,refrigerante,tortas,vinagre)(leite,óleo,ovos)(carnebovina,carvao,cerveja,frango,refrigerante)(acucar,azeitona,bacon,carnebovina,champignon,condicionador,macarrao,polpadetomate,queijo,queijoralado,refrigerante,sabonete,tomate,xampu)(cafe,carnebovina,frango,paes)

Para a realização do cálculo do suporte e identificação dos itens (unitários) frequentes, cada uma das sequências da MDB foi considerada. Uma a uma, todas as sequências da MDB foram analisadas e quando identificada a ocorrência do item, em qualquer *itemset* da sequência, o valor de suporte do item é incrementado.

É possível relacionar o resultado da execução do algoritmo MEMISP+AR, portanto, diretamente ao tamanho da base de dados, no que diz respeito à quantidade de sequências que a compõem. A $BDOC_2$ é constituída por uma quantidade aproximada de 1200 itens; 180 itens distintos de um supermercado que estão dispostos em 7 registros. Portanto, a utilização do min_sup com um valor relativamente baixo, como foi feito no estudo de caso anterior, acarretaria em uma quantidade muito grande de padrões identificados, naturalmente muitos itens ocorrem em pelo menos uma compra do dia. Para esse estudo de caso, foi pré-estabelecido o valor de min_sup = 0,8.

Como resultado da execução, são identificadas (1) subsequências frequentes com apenas um *itemset*, o que representa padrões que ocorrem em uma compra e (2) subsequências com mais de um *itemset*, representando padrões que ocorrem durante o dia, i.e., envolvem mais de uma compra. Dentre os resultados, alguns foram selecionados e são apresentadas na Tabela 5.5 com seus respectivos valores de suporte.

Tabela 5.5 – Subsequências frequentes identificadas. Parte do resultado da execução do algoritmo MEMISP+AR na $BDOC_2$ (min_sup = 0,8).

#ID	Subsequência frequente	Suporte
s_1	<(carnebovina,frango,paes,refrigerante)>	0,8571
s_2	<(biscoito,refrigerante)(paes,refrigerante)>	1,0
s_3	<(cafe,sal,xampu)>	1,0
s_4	<(cafe,desodorante,refrigerante,sal)>	0,8571
s_5	<(arroz,paes,refrigerante,sacodelixo)>	1,0

A opção pela sequência constituída por dez *itemsets* (cada um sendo um registro de venda) possibilita a identificação de padrões que consideram mais de uma compra, ou seja, padrões da semana, de determinados dias ou períodos, de forma diferente do exemplo anterior, onde cada subsequência identificada diz respeito a cada uma das vendas isoladamente. Esse aspecto pode ser visto na subsequência s_2 apresentada pela Tabela 5.5, onde o padrão identificado é composto por dois *itemsets*, ou seja, envolve duas vendas de um mesmo dia e, portanto, um comportamento recorrente entre duas, ou mais, compras durante a semana.

A Tabela 5.6 apresenta o conjunto de regras de associação criado a partir da subsequência frequente (carnebovina, frango, paes, refrigerante) identificada pelo MEMISP+AR. As regras geradas conservam a ordem de ocorrência dos itens e como a subsequência tem tamanho igual a 4, como resultado dessa etapa de execução, somente podem ser geradas, no máximo, 3 regras de associação entre os itens do padrão (nem a parte antecedente nem a consequente podem ser vazias e a ordem dos itens deve ser considerada).

Tabela 5.6 – Conjunto de regras de associação geradas a partir da sequência frequente s = <(carnebovina,frango,paes,refrigerante)>. Também, seus respectivos valores de confiança e validação quando a regra de associação for considerada forte, admitindo-se um valor de min_conf = 0,8.

Regra	Regra	Confiança
R_1	carnebovina, frango, paes ⇒ refrigerante	$\frac{6}{6} = 1{,}0$
R_2	carnebovina, frango ⇒ paes,refrigerante	$\frac{6}{6} = 1{,}0$
R_3	carnebovina ⇒ frango, paes,refrigerante	$\frac{6}{7} = 0{,}857$

Das regras apresentadas pela Tabela 5.6, os maiores valores de confiança são vistos nas regras R_1: carnebovina, frango, paes ⇒ refrigerante e R_2: carnebovina, frango ⇒ paes, refrigerante, ambas com valor de confiança = 1,0 . Tais valores de confiança podem ser utilizados para entender as regras de associação geradas da seguinte forma:

a) Em R_1, de todos os casos em que os três itens (carnebovina,frango,paes) foram encontrados nos registros da BDOC, o item refrigerante também foi identificado. Não houve nenhuma compra registrada, no período, contendo os três itens sem que o item refrigerante não estivesse presente também.
b) Em R_2, de todos os casos em que os dois itens (carnebovina,frango) foram encontrados nos registros da BDOC, o *itemset* (paes,refrigerante) também foi identificado.
c) Em R_3, entretanto, o item carnebovina foi identificado em 7 sequências da BDOC enquanto o padrão identificado em somente 6. O que indica que existem ocorrências do item carnebovina que não necessariamente são sucedidas pelo *itemset* (frango, paes,refrigerante).

5.6 Dificuldades de Implementação

É fato que a capacidade de processamento difere em cada configuração de computadores e influencia diretamente, tanto na velocidade de execução do algoritmo quanto no volume de dados que pode ser manipulado em uma varredura na base de dados. Durante o processo de implementação do MEMISP+AR, o aspecto de maior preocupação foi a possibilidade do algoritmo gerar combinações duplicadas de possíveis padrões. Como exemplo, observa-se na BD_2, apresentada na Tabela 5.1 e considerando o padrão frequente (a), unitário. No segundo passo do algoritmo, uma das possíveis combinações é o padrão (a)(b), que pode ser gerado a partir da união do stem item (a) ao P-pat <(b)>. Na BD_2 é possível observar que tal padrão poderia ser gerado nas sequências C_1, C_3 e C_4. Desta forma seriam gerados três vezes a mesma combinação, consequentemente o calculo de suporte e os procedimentos *IndexSet* e *mine* seriam executado para cada uma das três combinações.

O processo de geração de padrões sequenciais utilizado pelo MEMISP+AR foi projetado de forma que impossibilite a ocorrência de tais multiplicidades. Isso é possível com a inclusão de uma estrutura de lista de sequências, estruturalmente equivalente à visualizada na Figura 5.4, que lista as combinações a padrões frequentes já geradas. Por meio da utilização de tal lista o sistema pode, toda vez que uma nova combinação de possível padrão é gerada, executar uma busca e se tal combinação já foi analisada (pertence à lista) o MEMISP+AR não executa os procedimentos *IndexSet* e *mine* para tal combinação novamente, evitando a redundância de tarefas. Essa contribuição não é descrita em nenhuma das propostas consideradas nesta pesquisa.

Capítulo 6

O USO DO S_MEMISP+AR EM DADOS REAIS – EXPERIMENTOS REALIZADOS E ANÁLISES DE RESULTADOS

6.1 Considerações Iniciais

Como apresentado no Capítulo 5, o algoritmo MEMISP+AR (*MEMory Indexing for Sequential Pattern mining + Association Rules),* implementado como o sistema computacional S_MEMISP+AR, estende o funcionamento do algoritmo MEMISP (ver Capítulo 3) incorporando um processo de geração de regras de associação a partir dos padrões identificados, à semelhança daquele apresentado pelo algoritmo Apriori (ver Capítulo 2).

O S_MEMISP+AR também difere do MEMISP ao incorporar o tratamento de padrões temporais, de forma similar àquela do algoritmo ARMADA (ver Capítulo 4). O MEMISP+AR contribui, pois, tanto na identificação de padrões temporais, quanto na geração de regras de associação a partir dos padrões identificados.

Além desta seção de considerações iniciais, o capítulo está organizado em mais quatro seções. Na Seção 6.2 é apresentada a BD (nomeada BDO), utilizada nos experimentos, bem como as customizações dos dados originais, de maneira a adequá-la ao S_MEMISP+AR. Na Seção 6.3 são apresentados os primeiros resultados do uso do S_MEMISP+AR na BDO completa. A Seção 6.4 aborda as

formas de tratar o aspecto temporal utilizando a BDO. Na discussão de cada uma das formas, os resultados obtidos bem como as regras de associação geradas em cada uma delas são também apresentados. A Seção 6.5 apresenta uma comparação entre os resultados obtidos em cada experimento e, por fim, uma análise sucinta dos resultados obtidos nos experimentos é apresentada, ressaltando os principais aspectos positivos e negativos, identificados em cada uma das abordagens.

6.2 A BDO – Base de Dados Original Utilizada

A base de dados original (BDO) com dados reais, utilizada para os experimentos do S_MEMISP+AR, foi obtida junto à um supermercado na cidade de Osvaldo Cruz-SP. A BDO é composta por 1.500 registros de transações, relativos às vendas realizadas pelo supermercado, durante o período de uma semana (compreendido entre 01/08/2014 à 07/08/2014). Todos os registros são referentes ao mesmo caixa (Ca), selecionado entre os caixas rápidos do supermercado. A escolha de um particular caixa rápido Ca foi motivada pelo fato que, geralmente, caixas rápidos de um supermercado registram vendas que contém um número relativamente pequeno de itens, o que facilita a operacionalidade do sistema S_MEMISP+AR.

A Figura 6.1 apresenta uma imagem gerada a partir de um trecho extraído da BDO. Nela é possível observar todas as informações referentes a uma compra registrada por Ca, tais como: data e hora da compra, código dos produtos pertencentes à venda, descrição dos produtos, quantidade, dentre outros.

MATRIZ - EDIVALDO MARCONATO & CIA LTDA

COMPRA

DATA	HORA	MAQUINA	N CUP	C.P.F./C.N	NOME
01/08/2014	09:42:50	CAIXA-01	75925		

S	CÓDIGO	DESCRIÇÃO DO PRODUTO	UNID	QTDE	VENDA	TOT VENDA	CUSTO	TOT CUSTO	LUCRO	C REAL	TOT C REAL	L REAL
A	72945	MOLHO CASTELO ROSE P/SALADA 236ML	und	1.000	3.79	3.79	2.69	2.69	40.89	2.70	2.70	40.63
A	2392	REFRIG. FANTA LARANJA 2L	fco	1.000	3.89	3.89	3.29	3.29	18.24	3.29	3.29	18.24
A	15233	LEITE F. NESTLE CHAMYTO 6/1	pct	1.000	3.75	3.75	3.07	3.07	22.15	3.07	3.07	22.15
A	95161	LING. MISTA SADIA 240G	pct	1.000	4.15	4.15	2.85	2.85	45.61	2.86	2.86	45.16

DESCONTO	TOTAL VENDA	TOTAL P_COMPRA	LUCRO
0.00	15.58	11.90	30.92%

Figura 6.1 – Trecho extraído da BDO referente a uma compra registrada por Ca.

Entretanto, nem todos os atributos que os registros das vendas possuem, são essenciais para os experimentos com o S_MEMISP+AR. Código dos produtos, valores de cada item, total da compra, etc., não precisam fazer parte da BDO, como será detalhado oportunamente. Já os atributos data e hora são essenciais para a caracterização temporal do registro de compra.

O fato do registro de compra ser descrito por atributos com diferentes relevâncias, evidencia a necessidade de um pré-processamento da BDO, com o objetivo de remover aqueles irrelevantes para o uso do S_MEMISP+AR.

Para a realização de tal tarefa foi desenvolvido um *script* (nomeado *script_I*) em linguagem *php* (apresentado em Apêndice A) que, em sua primeira etapa, ao receber como entrada a BDO:

a) lê cada um dos seus registros, ignorando aqueles atributos considerados não relevantes;

b) transforma os atributos relevantes do registro recém processado em uma sequência tal que cada produto da compra é um item e cada registro de compra é uma sequência com apenas dois *itemsets*: o primeiro contendo dois itens (data e hora da compra) e o segundo, com os produtos relevantes da compra.

O resultado da primeira etapa do pré-processamento, via *script_I*, do registro da BDO mostrado na Figura 6.1, pode ser visto na Tabela 6.1.

Na segunda etapa o *script_I* lê a BDO pré-processada durante a primeira etapa e remove todas as ocorrências de espaços e de caracteres especiais das descrições dos produtos, tais como cedilha, acentos, barras, etc.

Tabela 6.1 – Trecho da BDO gerada pelo *script_I*, após a primeira etapa do pré-processamento.

Sequência com dois *itemsets*: (data,hora do registro) e (prod1, prod2,...,prodn)
<(01/08/2014,09:42:50)(molho castelo rose p/salada 236ml, refrig. fanta laranja 2l, leite f. nestle chamyto 6/1, ling. mista sadia 240g)>

Ainda durante a segunda etapa, o *script_I* considera a ocorrência de produtos duplicados em um mesmo registro de compra e mantém apenas uma única ocorrência. É importante salientar que em muitas compras, produtos idênticos são registrados separadamente devido à disposição no carrinho de compras.

Para que as diversas ocorrências de um mesmo produto possam ser ignoradas é preciso que, primeiro, um processo de agrupamento de produtos seja conduzido. Para a realização de tal processo, entretanto, é requerida a intervenção

do operador do sistema, para a tarefa de definição do nome (no caso, uma cadeia de caracteres) que identificará o agrupamento em questão.

Com o objetivo de definir um procedimento que permita a eliminação de itens duplicados em uma mesma transação, alguns cuidados foram tomados com vistas a padronizar o processo, por meio do pré-processamento de toda a BDO. Inicialmente, de maneira automática, é criada uma lista ordenada lexicograficamente, de todos os itens que comparecem em todas as transações. A seguir, um *processo de simplificação* é conduzido manualmente, de acordo com seguintes regras:

a) Variações em embalagem/volume bem como de quantidade de um determinado produto, por embalagem, são considerados um único produto. Por exemplo, independente da marca, refrigerantes Coca-Cola em embalagens com 12 unidades, cada uma delas com 200 ml e refrigerantes Coca-Cola em embalagens com 6 unidades, cada uma delas com 600 ml, e refrigerantes Fanta, em embalagens individuais com 1,5 l e refrigerantes Guaraná, em embalagens com 36 unidades, cada uma delas com 300 ml, etc..., são considerados um único produto, sob o nome refrigerantes. Devido à ordenação lexicográfica adotada, todos os produtos vendidos, das mais diferentes marcas e nas mais diferentes embalagens, estão listados consecutivamente;
b) Produtos identificados por nomes diferentes, tais como carne bovina acém, carne bovina contra, carne bovina musculo, etc... são manualmente identificadas como o produto carne-bovina. O mesmo acontece com carne suína e carne de frango. Partes específicas de boi, porco ou frango são também consideradas carne-bovina, carne-suina e carne-de-franco, respectivamente.
c) As convenções adotadas em (a) e (b) são, então, implementadas na BDO. Presentemente esse processo está sendo realizado manualmente.

No que segue é apresentado um exemplo simples do funcionamento do *script_I*, para um conjunto de itens, com o objetivo de detalhar a segunda etapa, na qual a intervenção do operador é requerida.

Na tarefa de agrupamento dos itens de compras, realizada pelo *script_I*, a BDO, após ser submetida à primeira etapa de execução do *script_I*, é varrida por completa e todos os itens são ordenados lexicograficamente (para facilitar a inspeção visual pelo operador). Independentemente da transação à qual os itens

pertencem, todos eles são apresentados ao operador do sistema; cada um deles é apresentado juntamente com um valor inteiro que representa seu identificador único (contador de itens da BDO). Tal listagem permite ao operador do sistema selecionar todos os itens que têm uma certa similaridade (a seu critério) e redefinir um novo nome (único e que não comparece na lista fornecida) que identificará cada um dos itens selecionados.

Um trecho do resultado do agrupamento realizado pelo *script_I* utilizando a BDO após ter sido submetida à primeira etapa do pré-processamento, pode ser observado na Tabela 6.2. Nela é possível observar os itens das compras registradas na BDO e que se encontram ordenados lexicograficamente.

Por meio da listagem gerada é possível ao operador do sistema inspecionar informações importantes, tais como: (1) várias ocorrências de um mesmo item de venda, e.g. o item *cremedentcolgatetacaom* e, também (2) a ocorrência de vários itens com nomes distintos mas que representam um mesmo produto. Em ambos os casos os itens podem ser essencialmente considerados um mesmo tipo de item.

Tabela 6.2 – Trecho da listagem lexicográfica da BDO, feita na etapa 2 do *script_I*, parte do S_MEMISP+AR.

ID (contador de itens)	Item da compra
3411	cremedeleitemococa200g
3412	cremedeleitenestle300g
3413	cremedentcolgatetacaom
3414	cremedentcolgatetacaom
3415	cremedentcolgatetacaom
3416	cremedentcolgatetacaom
3417	cremedentcolgatetacaom
3418	cremedentoralbmentasuave90
3419	cremedentoralbcomplimpprof
3420	cremedentoralbcomplimpprof
3421	cremedentoralbprosaudermin
3422	cremedentsensodyneorig5
3423	cremedentsorriso90g
3424	cremedentsorriso90g
3425	cremedentsorriso90g
3426	cremedentsorriso90g
3427	cremedentsorriso90g
3428	cremedentsorriso90g
3429	crememaggicebola68g
3430	cremepentsedacachos300m
3431	cremepentdovereconstco

A intervenção do operador do sistema é necessária nesse ponto com o objetivo de simplificar o conjunto de itens originais, considerando aqueles que são o mesmo, a menos de pequenos detalhes (tais como volume, peso, etc.), como visto anteriormente, quando da descrição detalhada do processo de simplificação.

É possível observar na Tabela 6.2 que quase todos os itens listados, ou são variações ou redundâncias de um mesmo produto, cremes dentais. Portanto os itens compreendidos no intervalo entre ID = 3413 e ID = 3428, da listagem, podem ter seus nomes substituídos por um mesmo novo termo comum, que os redefina. Por meio de tal processo, por exemplo, todos os itens listados na Tabela 6.2, à critério do operador, podem ser renomeados como *cremedental.*

É importante ressaltar que, ao final desta etapa do processamento, a BDO mantém a mesma quantidade de itens nas compras, porém renomeados de forma que não existam itens que representam um mesmo produto com nomes distintos. É importante lembrar, entretanto, que podem acontecer ocorrências de itens duplicados em uma mesma transação; é o caso, por exemplo, de uma transação que contém dois tipos de cremes dentais que, após a etapa 2, passa a conter itens duplicados, ou seja, duas ocorrências de cremedental. Tais redundâncias são eliminadas ao final do passo 2 de execução do *script_I*, como descrito a seguir. A repetição de itens/transação é tratada automaticamente via ordenação lexicográfica de cada transação e eliminação de todas repetições de um mesmo produto.

Ao final do processo de pré-processamento da BDO, realizado pelo *script_I*, cada transação é representada por uma sequência contendo dois *itemsets*:

a) o primeiro contém as informações de data e hora em que a compra foi registrada e
b) o segundo apresenta todos os produtos distintos registrados na compra, em ordem lexicográfica e sem a ocorrência de caracteres especiais ou espaços em branco.

A Tabela 6.3 mostra a transação apresentada na Figura 6.1, juntamente com mais quatro transações da BDO, após o pré-processamento da BDO pelo *script_I*.

O pré-processamento da base de dados é uma tarefa muito importante na execução do S_MEMISP+AR, pois reduz consideravelmente a quantidade de dados que serão processados, sem perda de informações importantes. Em média, cada transação da BDO possuía, antes de ser processada, 15 itens e, após o seu pré-processamento, passou a ter 11 itens.

Tabela 6.3 – Trecho da BDO após seu pré-processamento.

Sequência com dois *itemsets*: ((data e hora do registro) (produtos da transação))	
***Itemset* 1**	***Itemset* 2**
<(01/08/2014,09:42:50)	(leite,linguica-kg,refrigerante,temperoparasalada)>
<(01/08/2014,09:54:35)	(carne-bovina,carvao,cerveja,chiclete,linguica-kg,refrigerante)>
<(01/08/2014,10:02:57)	(carne-bovina,carne-de-frango,cerveja linguica-kg, margarina,refrigerante,sabaoempo,tomate-kg)>
<(01/08/2014,10:04:32)	(aguasanitaria,amaciante,iogurte)>
<(01/08/2014,10:09:19)	(abacaxi,alho-kg,amaciante,azeite,banana,biscoito,cenoura-kg,leite,maionese,paes,pao-de-queijo,pao-frances,queijo-kg,quiabo-kg,refrigerante,tempero-pronto,vinagre)>

É esperado que, ao considerar itens similares como sendo um único item, o suporte do item resultante aumente; isso de certa forma contribui para tornar mais eficiente a identificação dos padrões que ocorrem nas transações da BDO.

6.3 Experimento Inicial, Resultados e Análises

No que segue, é discutida a execução de um primeiro experimento envolvendo o S_MEMISP+AR usando dados reais (BDO completa após pré-processamento (ver Tabela 6.3)), com vistas a avaliar e testar o funcionamento do sistema. Para o experimento em questão:

a) *min_sup* = 0,05 (ou seja, um padrão para ser considerado frequente deve ter sido identificado em pelo menos 5% das transações da BDO, i.e., ocorrer em 75 transações) e
b) o valor mínimo de confiança para considerar uma regra forte foi estabelecido como *min_conf* = 0,035.

A Tabela 6.4 apresenta os resultados da primeira fase de execução do S_MEMISP+AR, em que são identificados todos os itens frequentes. Nela é possível observar que os produtos (itens) mais frequentes são: refrigerante, identificado em 41,47% (622) das transações da BDO e cerveja, identificada em 29,05% (435) das transações da BDO.

As escolhas dos valores dos parâmetros *min_sup* e *min_conf* foram feitas tendo em conta que, considerando um valor de suporte menor que 0,05, a quantidade de padrões encontrados é muito grande, o que provoca um aumento considerável no tempo de processamento do S_MEMISP+AR. Em muitos casos chega até a gerar erros de estouro de pilha. Por outro lado, definindo o *min_conf* com valor acima de 0,035, restringe o S_MEMISP+AR, a ponto de gerar apenas uma regra de associação com os padrões identificados.

Tabela 6.4 – Itens frequentes identificados na BDO via S_MEMISP+AR (*min_sup* = 0,05).

Item	Suporte	Item	Suporte
banana-kg	0,1438	queijo-kg	0,1114
laranja-kg	0,0620	acucar	0,0620
refrigerante	0,4146	cafe	0,0944
cebola-kg	0,1227	feijao	0,0634
cenoura-kg	0,0818	leite	0,1537
carne-bovina	0,2256	mortadela	0,0592
detergente	0,0860	oleo-de-soja	0,0916
arroz	0,0832	sabao-em-po	0,1015
linguica-kg	0,1438	milhoverde	0,0677
pao-frances	0,1889	salsicha	0,0691
papel-higienico	0,1057	carne-de-frango	0,1664
tomate-kg	0,1706	presunto-kg	0,0803
cerveja	0,2905	iogurte	0,0592
batata-kg	0,1269	sabonete	0,0874
maionese	0,0662		

Recordando o funcionamento do S_MEMISP+AR em sua segunda etapa de execução, fase em que é criada uma lista de índices para cada um dos itens identificados no passo anterior, o algoritmo combina cada um dos itens com seus sucessores na transação, a fim de gerar padrões frequentes. Tal processo é executado de forma recursiva, até que todas as combinações tenham sido geradas, considerando todas as transações que a lista de índices do item registra ocorrência dele. Todos os itens são analisados, juntamente com suas respectivas listas de índices.

O resultado do processo de identificação de padrões frequentes na BDO pelo S_MEMISP+AR é apresentado, dividido em duas tabelas: a Tabela 6.5 (padrões com 2 itens) e a Tabela 6.6 (padrões com mais de 2 itens). Note que todos os

padrões gerados apresentados em ambas as tabelas possuem apenas um único *itemset*.

É importante lembrar que o exemplo apresentado no Capítulo 5, em que o S_MEMISP+AR foi utilizado em um experimento cuja BD continha apenas 70 registros (i.e., um subconjunto da BDO utilizada nesta seção) e, portanto, o valor de *min_sup* teve de ser relativamente baixo (0,2 e 0,18) para que fosse possível identificar uma quantidade razoável de padrões frequentes. Entretanto, para esse experimento, com o uso da BDO e *min_sup* = 0.05 (i.e., relativamente baixo), o S_MEMISP+AR conseguiu identificar uma quantidade significativa de padrões frequentes.

Observando a Tabela 6.6 é possível extrair informações sobre a relação existente entre alguns itens de uma mesma transação da BDO. Considere o padrão frequente #ID = 10, da tabela, cujo valor de suporte = 0,0606. É possível afirmar, com base nesse padrão frequente identificado, que os itens carne-bovina, carne-de-frango, cerveja e refrigerante são comumente comprados juntos, por uma grande quantidade de fregueses que utilizam o estabelecimento (90 transações). Situação semelhante ocorre com o padrão frequente #ID = 7, com valor de suporte igual a 0,0634.

Como visto no Capítulo 5 (ver Seção 5.5), cada *itemset* e_i ($e_i \in s$), identificado no passo anterior, dá origem a um conjunto de regras $R_i = \{R_{i1}, R_{i2}, R_{ik}\}$ quando da utilização do procedimento *gera_regras_do_itemset*. Nessa etapa de execução são geradas todas as combinações possíveis, a fim de construir regras de associação com consequentes contendo de 1 a m−1 itens (m, tamanho do *itemset* e_i). Tanto o antecedente quanto o consequente da regra sempre conservam a ordem de ocorrência dos itens no *itemset* analisado. Para cada combinação gerada do conjunto de regras R_i, i=1,...,N, o valor de confiança é calculado (ver Equação 2.1) e, caso seja maior ou igual ao valor pré-estabelecido *min_conf*, a regra é considerada, caso contrário é descartada. Para o experimento descrito nesta seção, *min_conf* = 0,035. Todas as regras que permanecerem no conjunto R_i, i=1,...,N, são adicionadas ao conjunto final de regras de associação consideradas fortes i.e., o conjunto All_R. Ao final da execução do procedimento *memisp_rules*, tal conjunto irá conter todas as regras de associação geradas.

Tabela 6.5 – Padrões frequentes, contendo 2 itens, identificados na BDO via S_MEMISP+AR, para *min_sup* = 0,05.

Itemset	Suporte	*Itemset*	Suporte
(acuca,café)	0,0606	(cerveja,refrigerante)	0,1241
		(cerveja,linguica-kg)	0,0521
(banana-kg,refrigerante)	0,0719	(cerveja,tomate-kg)	0,0578
(banana-kg,carne-bovina)	0,0521		
(banana-kg,tomate-kg)	0,0493	(feijao,oleo-de-soja)	0,0505
(banana-kg,batata-kg)	0,0564		
(banana-kg,leite)	0,0564	(leite,refrigerante)	0,0902
		(leite,tomate-kg)	0,0578
(batata-kg,refrigerante)	0,0747		
(batata-kg,cebola-kg)	0,0662	(linguica-kg,refrigerante)	0,0761
(batata-kg,carne-bovina)	0,0564		
(batata-kg,tomate-kg)	0,0648	(oleo-de-soja,refrigerante)	0,0505
(batata-kg,carne-de-frango)	0,0505		
		(pao-frances,refrigerante)	0,0832
(carne-bovina,refrigerante)	0,1269		
(carne-bovina,linguica-kg)	0,7052	(papel-higienico,refrigerante)	0,0620
(carne-bovina,pao-frances)	0,5077	(papel-higienico,sabao-em-po)	0,0507
(carne-bovina,tomate-kg)	0,0662		
(carne-bovina,carne-de-frango)	0,0719	(presunto-kg,refrigerante)	0,0509
		(presunto-kg,queijo-kg)	0,0705
(carne-de-frango,refrigerante)	0,0775		
(carne-de-frango,linguica-kg)	0,0507	(queijo-kg,refrigerante)	0,0677
(carne-de-frango,tomate-kg)	0,0634		
(cebola-kg,refrigerante)	0,0662	(refrigerante,tomate-kg)	0,0846
(cebola-kg,tomate-kg)	0,0662		

Tabela 6.6 – Padrões frequentes, contendo mais de 2 itens, identificados na BDO via S_MEMISP+AR, para *min_sup* = 0,05.

#ID	*Itemset*	Suporte
1	(banana-kg,refrigerante, tomate-kg)	0,0507
2	(batata-kg,cebola-kg,refrigerante)	0,0521
3	(carne-bovina,carne-de-frango,cerveja)	0,0620
4	(carne-bovina,carne-de-frango,refrigerante)	0,0662
5	(carne-de-frango,refrigerante,tomate-kg)	0,0505
6	(cerveja,linguica-kg,refrigerante)	0,0521
7	(banana-kg,cerveja,refrigerante)	0,0634
8	(leite,refrigerante,tomate-kg)	0,0509
9	(presunto-kg,queijo-kg,refrigerante)	0,0505
10	(carne-bovina,carne-de-frango,cerveja,refrigerante)	0,0606

A próxima etapa de execução do S_MEMISP+AR consiste em utilizar os padrões frequentes identificados (ver Tabela 6.5) para a geração de um conjunto de regras de associação via *memisp_rules*. A Tabela 6.7 apresenta o conjunto de

combinações possíveis associadas ao padrão #ID = 10, que é um dos padrões frequentes identificados pelo S_MEMISP+AR. Na tabela são também mostrados o valor de confiança para cada uma das possibilidades geradas bem como indicado se a combinação em questão pode ser considerada uma regra forte ou não (será forte se seu valor de confiança for maior ou igual ao valor *min_conf* (estabelecido como 0,035).

Tabela 6.7 – Conjunto de combinações geradas a partir do padrão frequente #ID = 10, com seus respectivos valores de confiança. Serão consideradas regras aquelas combinações com conf ≥ 0,035.

Regra	Confiança	Regra considerada?
carne-bovina,carne-de-frango,cerveja ⇒ refrigerante	0,0505	Sim
carne-bovina,carne-de-frango ⇒ cerveja,refrigerante	0,0366	Sim
carne-bovina ⇒ carne-de-frango,cerveja,refrigerante	0,0181	Não

6.4 O Aspecto Temporal e a BDO

Assim como evidenciado anteriormente, o foco da mineração de dados é a extração da maior quantidade possível de informações relevantes de um domínio específico. A incorporação do tratamento temporal ao processo de mineração de dados, agrega à analise, a possibilidade de identificar uma possível relação existente entre os padrões encontrados e o fator tempo. Alguns padrões são mais fáceis de serem compreendidos quando são conhecidas características temporais referentes ao domínio (de dados) que tais padrões costumam ocorrer.

Os resultados obtidos no experimento apresentado na Seção 6.3 são, de certa forma, padrões temporais quando analisados juntamente com o intervalo de duração em que os dados ocorreram. Como descrito anteriormente, os dados da BDO são referentes ao período compreendido entre 01/08/2014 à 07/08/2014, i.e., primeira semana do mês de agosto. Logo, é possível relacionar o período temporal caracterizado como início do mês, considerado a época de pagamento de salários, com um alto volume de registro de compras contendo itens, caracterizados como itens da 'cesta básica', tais como: arroz, feijão, açúcar, etc.,

Descobrir informações relevantes contribui, de forma significativa, com uma melhor compreensão do domínio analisado e, consequentemente, com a

possibilidade de melhor administração das atividades subjacentes. Agregar relações temporais ao conhecimento adquirido torna tal conhecimento ainda mais informativo. Entretanto, existem inúmeras formas de considerar o fator tempo nas análises efetuadas sobre as bases de dados; uma nova abordagem pode ser proposta para cada necessidade detectada.

No que segue são apresentadas duas abordagens diferenciadas (daquela descrita na Seção 6.3), relativas ao tratamento temporal, cada uma delas acompanhada de um experimento de mineração de padrões.

a) A primeira abordagem também faz o pré-processamento da BDO pelo *script_I.* Seu foco, entretanto, concentra-se na descoberta de padrões relacionados a cada dia da semana, identificando tanto padrões que se repetem durante toda a semana, quanto aqueles relacionados apenas a um determinado dia.
b) A segunda abordagem realiza também o pré-processamento da BDO pelo *script_I.* Assim como o ARMADA, essa abordagem é baseada na definição de *itemset* proposta em (HÖPPNER, 2001), como apresentada no Capítulo 4 (ver Seção 4.3). Considera intervalos de estados como *itemsets*, de forma que possa ser possível representar as relações existentes entre os *itemsets* do padrão identificado via AIA (ver Capítulo 4 - Seção 4.2).

6.4.1 Abordagem 1 — Padrões Temporais Relacionados a Dias da Semana

Seguindo a mesma metodologia empregada na Seção 6.3, é possível usar o S_MEMISP+AR para identificar apenas aqueles padrões que ocorrem em dias específicos da semana, motivado pela possibilidade de encontrar padrões considerando o fator tempo (representado pelo dia da semana), sem alterar a BDO. A estratégia adotada nessa abordagem/experimento não faz, ainda, uso do formalismo proposto por Allen (AIA).

A busca (e identificação) de um padrão temporal pode ser utilizada para evidenciar quais tendências (associadas a compras de produtos) podem ser inferidas relativas a um determinado dia da semana que, eventualmente, não se repetem nos outros dias. Para tal abordagem a BDO foi particionada em 7 blocos;

cada bloco da partição (considerado uma base de dados *per se*) é composto pelos registros de compras realizadas em um determinado dia da semana. Os sete blocos que compõem a partição são nomeados como: BDO-Dom, BDO-Seg, BDO-Ter, BDO-Qua, BDO-Qui, BDO-Sex e BDO-Sab. Cada uma das bases de dados resultantes (i.e., cada bloco) tem, em média, 230 registros, com exceção da BDO-Dom que possui 184.

A estratégia adotada para esse experimento foi de manter a mesma definição do valor de *min_sup* = 0,05 i.e., para um padrão ser considerado frequente, ele deve ocorrer, em média, em 15 registros de compras de um determinado dia. Os resultados obtidos em cada bloco da partição da BDO, além de servirem como subsídio para elaboração de regras de associação caracterizando o dia da semana em questão, foram também selecionados e buscados nos demais blocos da BDO (demais dias da semana). Tal estratégia visou evidenciar quais padrões se mantém frequentes nos diversos dias da semana e quais são aqueles específicos de um determinado dia.

Como resultado do experimento, alguns dos padrões encontrados podem ser visualizados na Tabela 6.8. A tabela contém, também, uma identificação referente a qual dia da semana o padrão está relacionado. Aqueles marcados com (*) são padrões recorrentes durante a semana, i.e., padrões identificados em mais de 50% (mais de 3) dos dias da semana considerada, intitulados *padrões globais*.

É importante notar que os padrões apresentados na Tabela 6.8 são apenas alguns dentre aqueles que possuem mais de dois itens que os compõem. Tal escolha foi motivada com vistas, tanto a reduzir a tabela, quanto a evidenciar apenas aqueles padrões que têm maiores quantidade de itens, talvez seja esse um indicativo de maior 'importância' da compra, considerando ser uma informação mais difícil de ser obtida por inspeção humana.

Tabela 6.8 – Padrões identificados em dias específicos da semana. Os padrões identificados com (*) são recorrentes durante a semana, i.e., ocorrem em 4 dias ou mais.

Dia da semana (BDO-Dia)	Padrões encontrados	Suporte	Dia da semana (BDO-Dia)	Padrões encontrados	Suporte
BDO-Dom	***(carne-bovina,carne-de-frango,cerveja,refrigerante)**	0,0543	BDO-Qui	(biscoito,cafe,pao-frances)	0,1745
	(carne-bovina,carvao,cerveja)	0,1630		(leite,maionese,oleo-de-soja)	0,1320
	***(carne-bovina,cerveja,refrigerante)**	0,3532		(iogurte,leite,ovos)	0,1509
	(carne-bovina,carne-suina,cerveja)	0,0978		(chocolate,leite-condensado,pao-frances)	0,1367
	(carne-de-frango,macarrao,refrigerante)	0,2228		(cafe,linguica-kg,limao-kg)	0,1084
	***(carne-bovina,carne-de-frango,refrigerante)**	0,3478		(carne-suina,queijo-kg,refrigerante)	0,1556
	(batata-kg,creme-de-leite,macarrao)	0,1630		(acucar,queijo-kg,refrigerante)	0,1698
BDO-Seg	(banana-kg,carne-bovina,leite)	0,3162	BDO-Sex	(cerveja,refrigerante,sabao-em-po)	0,1567
	(arroz,iogurte,leite)	0,1674		(alface,tomate-kg,xampu)	0,1604
	(leite,oleo-de-soja,pao-frances)	0,1813		(alafce,alho-kg,batata-kg)	0,1716
	(leite,pao-frances,presunto-kg)	0,2883		(refrigerante,sabao-em-po,xampu)	0,1940
	(biscoito,sabonete,xampu)	0,0697		(banana,cerveja,refrigerante)	0,2164
	(arroz,acucar,quejo-kg)	0,1720		(cerveja,pao-frances,tomate-kg)	0,2947
	(feijao,papel-higienico,sabonete)	0,1953		(batata-kg,pao-frances,refrigerante)	0,3432
BDO-Ter	(acucar,iogurte,macarrao-instantaneo)	0,0921	BDO-Sab	***(carne-bovina,carne-de-frango,cerveja)**	0,2837
	(cheiro-verde,detergente,leite)	0,0833		***(carne-de-frango,refrigerante,tomate-kg)**	0,2145
	(cafe,biscoito,oleo-de-soja)	0,2324		(carne-bovina,carvao,refrigerante)	0,1557
	(creme-de-leite,salgado,extrato-de-tomate)	0,1271		(carne-bovina,linguica-kg,refrigerante)	0,1799
	(alface,batata-kg,cebola-kg)	0,2105		(carne-bovina,maca-kg,refrigerante)	0,9688
	***(batata-kg,cebola-kg,refrigerante)**	0,2982		(carne-bovina,ovos,refrigerante)	0,7266
	(banana-kg,refrigerante,tomate-kg)	0,4473		(queijo-kg,refrigerante,tomate-kg)	0,2214
BDO-Qua	(cerveja,biscoito,salgado)	0,1711			
	(cerveja,refrigerante,salgado)	0,1787			
	(carne-bovina,cerveja,pao-frances)	0,2965			
	(carne-bovina,linguica-kg,refrigerante)	0,2775			
	(carne-bovina,cerveja,refrigerante)	0,3041			
	***(cerveja,linguiça-kg,refrigerante)**	0,2129			
	(acucar,oleo-de-soja,refrigerante)	0,1825			

Como pode ser visto na Tabela 6.8 foram apresentados somente 7 padrões relativos a cada dia da semana (o número de padrões/dia da semana está mostrado na Figura 6.2). Padrões que ocorrem em mais de um dia são apresentados apenas uma única vez, no dia que o padrão ocorre com maior valor de suporte.

Uma breve análise dos resultados apresentados na Tabela 6.8 é suficiente para identificar algumas características que relacionam os padrões identificados com o dia da semana. Isso ocorre, por exemplo, na segunda e terça-feira, dias em que os padrões com maior valor de suporte possuem itens variados, tais como frutas, leite, produtos de higiene, etc. Nesses dias, os padrões identificados têm valor de suporte menor que nos demais, devido ao fato que, ao possuírem muitos produtos variados, não há muitas compras com itens similares. Já nos finais de semana é mais comum a ocorrência de padrões com itens tais como carnes, bebidas, carvao, pao-frances, etc; padrões do final de semana apresentam valores de suporte mais elevados.

As quantidades de padrões identificados, em cada um dos dias da semana, considerando o valor *min_sup* = 0,05, podem ser vistos na Figura 6.2, juntamente com a quantidade de padrões globais identificados (setor hachurado). Na figura é possível ver que a quantidade de padrões identificados durante a semana é, em média, menor que aquela dos finais de semana e, também, que o dia que tem o menor número de padrões identificados é a terça-feira (11 padrões com tamanho maior ou igual a 3 identificados), contrapondo-se com domingo, que apresenta o maior número de padrões identificados, 25 considerando o mesmo tamanho.

Uma breve análise dos resultados, apresentados na Figura 6.2, pode ser suficiente para verificar que a maior incidência de padrões que se repetem, foi identificada durante o final de semana. É senso comum considerar que o fluxo de vendas seja maior nesse período. É importante, porém, considerar que no domingo, dia em que foi identificado a maior quantidade de padrões (25), é também o dia em que, geralmente, estabelecimentos desse segmento ficam abertos apenas em um período (pela manhã, como é o caso do supermercado considerado neste trabalho).

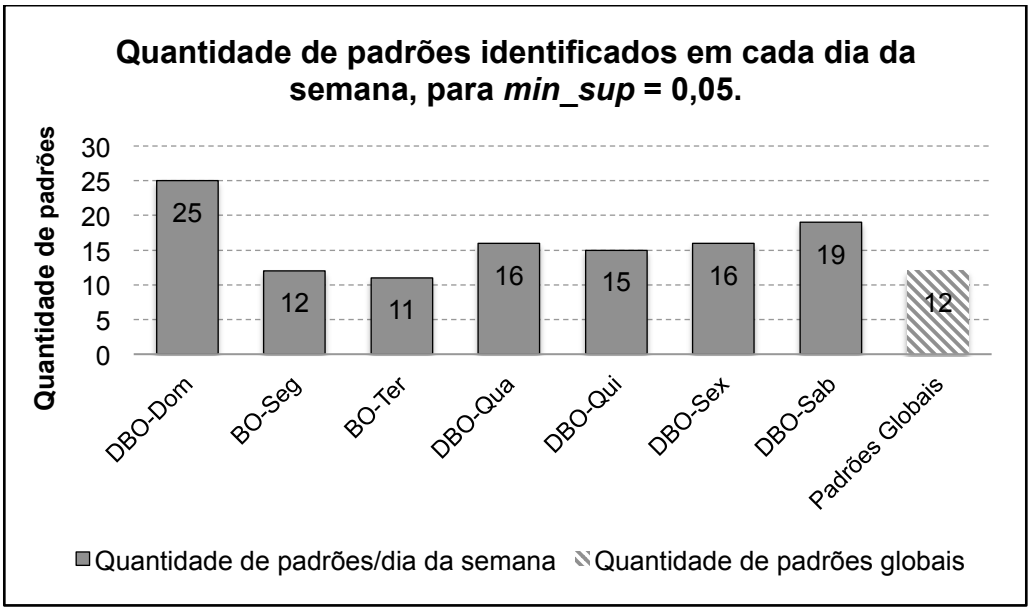


Figura 6.2 – Gráfico comparativo quantitativo de padrões identificados, contendo três ou mais itens de compras por dia da semana, bem como aos padrões globais identificados, para *min_sup* = 0,05.

Tal resultado, no mínimo curioso, pode ser melhor compreendido ao analisar os resultados apresentados na Tabela 6.8. No domingo, a variedade de itens de compras é menor, ou seja, o fluxo de compras (quantidade de registros de compras na BDO-Dom) é alto, porém muitas compras contém produtos similares, tais como: carne-bovina, carne-de-frango, carvao, cerveja, refrigerante, pao-frances, macarrao, dentre outros (ou seja, geralmente itens associados a refeições tipicamente consideradas de fim de semana).

Pode ser observado na Figura 6.8 que a quantidade de padrões frequentes nas quartas-feiras é razoavelmente alta (em comparação com os outros três dias, segunda, terça e quinta). Uma possível explicação pode estar relacionada ao fato que tradicionalmente, às quartas-feiras ocorrem jogos de futebol (noturnos) de campeonatos importantes, que são transmitidos pela televisão aberta. Tal explicação, também, pode ser subsidiada pelo fato dos itens participantes dos padrões frequentes identificados na quarta-feira serem, via de regra, cerveja, salgados, carne-bovina e refrigerante, considerados importantes por aqueles que acompanham tal esporte.

É importante evidenciar que nem todos os padrões identificados, quando a BDO completa é utilizada são considerados padrões globais (ver tabelas 6.6 e 6.8). Tal fato pode ser explicado devido à disposição dos registros de compras na BDO, i.e., alguns padrões podem ocorrer por várias vezes durante um ou dois dias da

semana, porém não têm valor de suporte que os habilitem a serem considerados frequentes nos demais dias. Dessa forma, o padrão é considerado frequente durante a semana ao analisar a BDO completa, porém não global quanto a mesma é particionada em bases menores referentes aos dias da semana.

A Tabela 6.9 apresenta um conjunto de regras de associação, consideradas fortes, geradas a partir dos padrões frequentes identificados por meio do uso do S_MEMISP+AR nas bases de dados dos dias da semana. Nota-se que a maioria das regras consideradas são aquelas que possuem a menor quantidade de itens na parte do consequente da regra. Para tal geração de regras de associação, o valor de *min_conf* foi mantido em 0,035.

Tabela 6.9 – Conjunto de combinações geradas a partir dos padrões frequente identificados, com seus respectivos valores de confiança. Serão consideradas regras de associação aquelas combinações com valor de confiança ≥ 0,035.

#ID Regra	Regra	Valor de Confiança
1	carne-bovina,carne-de-frango,cerveja ⇒refrigerante	0,0505
2	carne-bovina,carne-de-frango ⇒ cerveja,refrigerante	0,0366
3	carne-bovina ⇒ cerveja,refrigerante	0,0366
4	carne-bovina,cerveja ⇒ refrigerante	0,0535
5	carne-bovina,refrigerante ⇒ tomate-kg	0,0366
6	carne-de-frango,refrigerante ⇒ tomate-kg	0,0724
7	batata-kg,cebola-kg ⇒ refrigerante	0,0412
8	cerveja,linguiça-kg ⇒ refrigerante	0,0505
9	banana-kg,carne-bovina ⇒ leite	0,0602
10	batata-kg,pao-frances ⇒ refrigerante	0,0712

6.4.2 Abordagem 2 – Padrões Temporais Relacionados a Características Intrínsecas

Como mencionado anteriormente, existem várias abordagens que viabilizam a incorporação do aspecto temporal ao processo de mineração de padrões. Como visto na descrição do algoritmo ARMADA (ver Capítulo 4 – Seção 4.3), intervalos temporais são utilizados para representar o espaço temporal em que determinados eventos ocorrem, compondo os chamados intervalos de estados. O experimento descrito na Seção 6.3 considera intervalos temporais que não são representados por pontos temporais, mas sim pelos dias da semana.

Via de regra existem características e informações relativas a um determinado domínio que não estão explicitamente armazenadas na base de dados. Algumas dessas características são mais fáceis de serem identificadas com a colaboração de um especialista do domínio em questão. Além das informações que caracterizam as transações do domínio considerado, existem diversas características importantes associadas aos dados, que podem colaborar com a ampliação do escopo do conjunto de regras de associação inferidas. Um conjunto de tais características foi obtido via análise da BDO, com a colaboração de um especialista no domínio (o mesmo que disponibilizou a BDO). São características tais como períodos com fluxo alto ou baixo de compras, quantidade elevada de compras de determinados produtos, tais como bebidas, carnes, produtos de limpeza, etc. Cada característica é associada ao intervalo de tempo em que é identificada, de forma a viabilizar a descrição formal das relações temporais que existem entre os padrões identificados, via Álgebra Intervalar de Allen (ver Capítulo 4).

As principais características identificadas na BDO estão mostradas na Tabela 6.10, que também apresenta uma descrição sucinta sobre a sua caracterização. É importante, entretanto, evidenciar que tais características não são as únicas que representam o domínio e, tampouco, são as mais importantes a serem consideradas em qualquer análise. O conjunto das 13 características selecionadas teve o objetivo único aquele de mostrar a flexibilidade do S_MEMISP+AR.

Após a definição manual das características intrínsecas, a BDO foi novamente visitada, a fim de subsidiar a construção de uma base de dados de características, a BDO-C, composta pelas informações:

- dia da semana,
- características identificadas em cada dia e
- pontos no tempo (discretizado por hora) em que cada uma das ocorrências das respectivas características se inicia e termina.

Para melhor representar o espaço temporal que corresponde a um dia de atividades do estabelecimento, foi definido que em dias compreendidos entre a segunda-feira e o sábado, inclusive, os horários das atividades do estabelecimento, que se iniciam às 08:00hs e encerram às 19:00hs, são representados por intervalos de tempo de 1 hora, cujos limites estão no conjunto de inteiros {0, 1, ..., 11}. Entretanto, como no domingo o horário de funcionamento é das 08:00hs às 13:00hs,

o novo intervalo é representado por intervalos de duração de uma hora, cujos limites estão no conjunto de inteiros {0, 1, 2, 3, 4, 5}.

Tabela 6.10 – Conjunto de características intrínsecas sobre o domínio representado pela BDO.

#ID	Característica	Descrição
C_1	fluxo-de-compras-alto	Quantidade de compras registradas acima da média (≥ 20 compras / hora)
C_2	fluxo-de-compras-baixo	Quantidade de compras registradas abaixo da média (≤ 10 compras / hora)
C_3	muitas-compras-de-itens-basicos	Pelo menos 5 de cada 10 compras registradas contendo itens de consumo básico, tais como: arroz, feijão, farinha, macarrão, açúcar, óleo, sal, etc.
C_4	muitas-compras-de-bebidas	Pelo menos 5 de cada 10 compras registradas contendo bebidas alcoólicas.
C_5	muitas-compras-de-refrigerantes	Pelo menos 5 de cada 10 compras registradas contendo refrigerantes ou similares.
C_6	muitas-compras-de-limpeza	Pelo menos 5 de cada 10 compras registradas contendo itens de limpeza, tais como: detergentes, sabão em pó, desinfetantes, etc.
C_7	muitas-compras-de-frutas	Pelo menos 5 de cada 10 compras registradas contendo frutas.
C_8	muitas-compras-de-feirinha	Pelo menos 5 de cada 10 compras registradas contendo legumes e/ou verduras.
C_9	muitas-compras-de-higiene	Pelo menos 5 de cada 10 compras registradas contendo itens de higiene pessoal, tais como: xampu, sabonete, pasta de dente, desodorante, etc.
C_{10}	muitas-compras-de-acougue	Pelo menos 5 de cada 10 compras registradas contendo carnes bovina, suína ou de frango.
C_{11}	muitas-compras-de-doces	Pelo menos 5 de cada 10 compras registradas contendo doces industrializados.
C_{12}	muitas-compras-de-enlatados	Pelo menos 5 de cada 10 compras registradas contendo produtos enlatados.
C_{13}	muitas-compras-de-padaria	Pelo menos 5 de cada 10 compras registradas contendo produtos de padaria, tais como: pães, bolos, tortas, etc.

Nesse modelo de representação sendo proposto, é possível que múltiplas ocorrências de uma mesma característica possam ocorrer em um mesmo dia. Por exemplo, as características fluxo-de-compras-alto (C_1) e fluxo-de-compras-baixo (C_2) podem se alternar durante o expediente; a característica C_1 pode ter ocorrido durante $[0,1] \cup [1,2] = [0,2]$, a $C_2 = [3,4]$ e a C_1 voltado a acontecer em [7,8], ou seja,

C_1 ocorre duas vezes e C_2 uma vez no dia em questão. Simultaneamente à primeira ocorrência de C_1, pode acontecer da característica muitas-compras-de-padaria (C_{13}) também ocorrer.

Com vistas à simplificação, na Tabela 6.11, que mostra as características identificadas por dia levando em conta toda a base (i.e., mostra uma simplificação da BDO-C), as ocorrências que apresentaram multiplicidade por dia, foram representadas por uma única ocorrência i.e., a com o maior intervalo de duração. A tabela mostra o conjunto de características identificadas em cada dia da semana, acompanhado do correspondente intervalo de tempo.

O S_MEMISP+AR não foi projetado para utilizar uma base de dados com a arquitetura semelhante à BDO-C. Dessa forma, algumas alterações nas estruturas de dados utilizadas pelo sistema foram necessárias para possibilitar a mineração dos padrões em tal base. À semelhança do ARMADA, o S_MEMISP+AR faz uso do formalismo proposto por Höppner (2001), de tal forma que cada característica é considerada como um estado, associada a um ponto temporal que determina seu início e outro que determina seu fim, i.e., um intervalo de estado.

Relembrando a formalização introduzida por Höppner (2001), um intervalo de estado pode ser representado pela terna (b_s, s, f_s), em que b_s representa o ponto no tempo que o estado s se inicia e f_s o ponto em que s termina, dessa forma o espaço temporal entre b_s e f_s é o espaço em que o estado s ocorre, i.e., o intervalo de duração de s. Para representar as relações entre os intervalos de estado, Höppner utiliza a AIA (descrita na Seção 4.2).

Tabela 6.11 –Base de dados de características intrínsecas (BDO-C), construída a partir da BDO.

Dia	t_início	t_fim	Características intrínsecas
Domingo	2	5	C_1 – fluxo-de-compras-alto
	1	5	C_4 – muitas-compras-de-bebidas
	2	5	C_5 – muitas-compras-de-refrigerantes
	1	5	C_{10} – muitas-compras-de-acougue
	2	4	C_{11} – muitas-compras-de-doces
	0	4	C_{13} – muitas-compras-de-padaria
Segunda-feira	7	10	C_1 – fluxo-de-compras-alto
	0	4	C_2 – fluxo-de-compras-baixo
	6	10	C_3 – muitas-compras-de-itens-basicos
	4	6	C_5 – muitas-compras-de-refrigerantes
	6	10	C_7 – muitas-compras-de-frutas
	5	10	C_{10} – muitas-compras-de-acougue
	6	9	C_{12} – muitas-compras-de-enlatados
	4	10	C_{13} – muitas-compras-de-padaria
Terça-feira	0	4	C_2 – fluxo-de-compras-baixo
	6	8	C_3 – muitas-compras-de-itens-basicos
	4	9	C_5 – muitas-compras-de-refrigerantes
	8	9	C_7 – muitas-compras-de-frutas
	6	9	C_8 – muitas-compras-de-feirinha
	4	8	C_9 – muitas-compras-de-higiene
	6	11	C_{10} – muitas-compras-de-acougue
	0	3	C_{13} – muitas-compras-de-padaria
Quarta-feira	8	11	C_1 – fluxo-de-compras-alto
	0	2	C_2 – fluxo-de-compras-baixo
	8	11	C_4 – muitas-compras-de-bebidas
	4	10	C_5 – muitas-compras-de-refrigerantes
	6	7	C_6 – muitas-compras-de-limpeza
	4	8	C_8 – muitas-compras-de-feirinha
	6	11	C_{10} – muitas-compras-de-acougue
	4	11	C_{13} – muitas-compras-de-padaria
Quinta-feira	4	8	C_3 – muitas-compras-de-itens-basicos
	6	10	C_5 – muitas-compras-de-refrigerantes
	2	6	C_7 – muitas-compras-de-frutas
	2	7	C_{10} – muitas-compras-de-acougue
	1	7	C_{13} – muitas-compras-de-padaria
Sexta-feira	5	11	C_1 – fluxo-de-compras-alto
	0	2	C_2 – fluxo-de-compras-baixo
	7	11	C_4 – muitas-compras-de-bebidas
	5	11	C_5 – muitas-compras-de-refrigerantes
	9	11	C_8 – muitas-compras-de-feirinha
	8	10	C_9 – muitas-compras-de-higiene
	7	11	C_{10} – muitas-compras-de-acougue
	8	11	C_{13} – muitas-compras-de-padaria
Sábado	4	11	C_1 – fluxo-de-compras-alto
	6	11	C_4 – muitas-compras-de-bebidas
	5	11	C_5 – muitas-compras-de-refrigerantes
	6	8	C_6 – muitas-compras-de-limpeza
	7	11	C_{10} – muitas-compras-de-acougue
	5	6	C_{11} – muitas-compras-de-doces
	6	11	C_{13} – muitas-compras-de-padaria

Com relação à implementação do sistema, foram feitas modificações nas estruturas de dados do S_MEMISP+AR para possibilitar que o algoritmo identificasse os padrões temporais, com seus respectivos intervalos, viabilizando assim o uso da AIA para representar as relações existente entre as características que compõem o padrão.

A principal alteração realizada foi aquela relacionada diretamente à forma como um *itemset* é representado e armazenado pelo sistema, ou seja, na estrutura de dados que armazena um *itemset*. Tal estrutura (introduzida no Capítulo 5 – Seção 5.3) foi alterada como segue:

- O ponteiro (**it_cabeca*) que indica o início da lista de itens que definem um *itemset* é removido, juntamente com o valor inteiro (*it_tamanho*) que armazena o tamanho do *itemset* (quantidade de itens que o define);
- Uma cadeia de caracteres (*it_nome*) que identifica qual a característica armazenada é inserida, juntamente com um par de inteiros que definem os pontos temporais de início (*it_inicio*) e fim (*it_fim*), respectivamente, da ocorrência da característica na BDO-C.

A variável que armazena o valor de suporte do *itemset* foi mantida, assim como o ponteiro para um eventual próximo *itemset* em uma sequência. Na Figura 6.3 (a) a antiga estrutura utilizada para armazenar um *itemset* é mostrada e, em (b) é mostrada a nova estrutura.

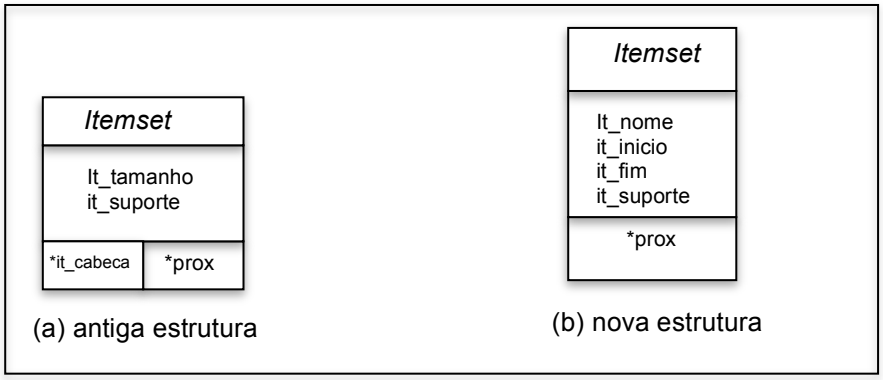


Figura 6.3 – Nova estrutura de dados que armazena um *itemset* contemplando os pontos temporais de início e fim em cada característica.

Um padrão identificado na BDO-C, via S_MEMISP+AR, é constituído por uma sequência de características intrínsecas, cada uma associada a seu respectivo intervalo temporal. O padrão é, pois, um conjunto ordenado de ternas do tipo (inicio_caracteristica, nome_característica, fim_característica). O espaço temporal que descreve o padrão é constituído pelo espaço temporal formado pelos intervalos

das características que compõem o padrão, como formalizado em (HÖPPNER, 2001).

É importante lembrar, entretanto, que na etapa de geração de novos padrões (combinação do padrão com seus respectivos sucessores na transação da BD) a partir dos padrões já identificados, ao fazer uso da representação formal de Höppner, há somente a possibilidade da construção de padrões do tipo 1 (incorporando um novo *itemset* ao final padrão já identificado).

Um novo *script* em linguagem *php*, nomeado *script_II* (apresentado em Apêndice B), foi proposto com a finalidade de realizar um pré-processamento na BDO-C e customizá-la, de forma a possibilitar sua utilização como entrada para o S_MEMISP+AR. Via *script_II* cada um dos registros armazenados na BDO-C é lido e reescrito seguindo o formalismo de Höppner. Como pode ser visto na Tabela 6.12, cada dia da semana contem um conjunto de características e, consequentemente, ao final do pré-processamento, a saída do *script_II* é um conjunto de 7 sequências de características referentes a cada dia da semana. Cada uma das sequências apresenta seus *itemsets* dispostos de maneira ordenada segundo os intervalos que armazenam, de maneira semelhante ao formalismo de Höppner (ver Capítulo 4).

Um trecho da saída do *script_II* pode ser visto na Tabela 6.12. Nele é possível observar a sequência gerada com as características referentes a segunda-feira (ver Tabela 6.11); cada característica é apresentada juntamente com seus respectivos intervalos temporais.

Tabela 6.12 – Trecho da saída do *script_II*. Sequência de características, reescritas no formalismo de Höppner. Cada uma das ternas representa uma característica intrínseca que ocorre em um dia da semana; no caso, a segunda-feira.

#ID	Sequência de características referentes à segunda-feira
s_segunda-feira	$<(0,C_2,4)(4,C_5,6)(4,C_{13},10)(5,C_{10},10)(6,C_{12},9)(6,C_3,10)(6,C_7,10)(7,C_1,10)>$

A partir da BDO-C, reescrita em temos de intervalos de estados (características), o S_MEMISP+AR é utilizado para identificar os padrões que ocorrem em cada uma das sequências geradas, i.e., em cada dia da semana. Os resultados obtidos podem ser vistos na Tabela 6.13, que apresenta também os respectivos valores de suporte para cada um dos padrões identificado.

Tabela 6.13 – Conjunto de padrões identificados via S_MEMISP+AR na BDO-C, com seus respectivos valores de suporte. São considerados padrões frequentes aqueles que sobrepõem *min_sup* = 0,428.

#ID	Padrões identificados	Suporte
1	$<(C_4)(C_5)(C_{10})>$	4/7 = 0,5714
2	$<(C_1)(C_4)(C_5)(C_{13})>$	3/7 = 0,4285
3	$<(C_3)(C_{10})(C_{13})>$	3/7 = 0,4285
4	$<(C_1)(C_{10})(C_{13})>$	4/7 = 0,5714
5	$<(C_1)(C_4)(C_5)>$	4/7 = 0,5714
6	$<(C_3)(C_5)(C_7)>$	3/7 = 0,4285
7	$<(C_5)(C_8)(C_{10})>$	4/7 = 0,5714
8	$<(C_1)(C_2)(C_{13})>$	3/7 = 0,4285
9	$<(C_1)(C_5)(C_{10})>$	5/7 = 0,7142
10	$<(C_1)(C_8)(C_{10})>$	3/7 = 0,4285

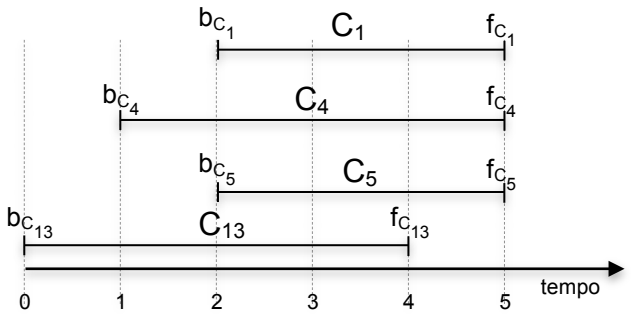
Os resultados apresentados na Tabela 6.13 foram obtidos usando o valor pré-definido para o *min_sup* = 0,428, ou seja, para ser considerado um padrão frequente, a combinação tem que ocorrer em 3 das 7 sequências que compõem a BDO-C.

Os resultados apresentados na Tabela 6.13 não consideram os intervalos temporais associados a cada uma das características, i.e. não utilizam as relações da AIA como pré-requisito para a busca de padrões, devido a isso eles não estão ordenados por seus intervalos temporais, como define Höppner e sim pelo nome do intervalo de estado (característica). Essa estratégia se deve, nesse experimento, ao fato que se considerados os intervalos de cada característica, levando em conta a BDO-C, a quantidade de padrões encontrados seria reduzida consideravelmente e, provavelmente, nenhum seria considerado como um padrão frequente. Pode, pois, ocorrer que uma combinação de *itemsets* ocorra em muitas sequências da BDO-C quando desconsiderados seus intervalos, porém não ocorrer em pelo menos 3 sequências quando considerados, o que impossibilita o padrão de ser considerado frequente.

É fácil ver um exemplo dessa afirmação ao considerar o padrão #ID = 2 ($<(C_1)(C_4)(C_5)(C_{13})>$). Note que o padrão ocorre na quarta-feira, sexta-feira e domingo. Se fossem analisados seus respectivos intervalos de duração, seriam utilizadas diferentes relações da AIA para representar seus relacionamentos em cada um dos três dias da semana em que foram identificados, inviabilizando, assim, tal sequência ser considerada um padrão.

A Figura 6.4 apresenta uma representação pictórica de uma ocorrência do padrão #ID = 2 (ver Tabela 6.11 – Domingo), com seus respectivos intervalos de estado. Uma vez identificado o padrão, as relações da AIA podem ser utilizadas para representá-lo na BDO-C. Esse processo 'traduz' a representação intervalar do padrão em uma representação relacional. A figura apresenta, também, a matriz M_{4x4} das relações entre os intervalos temporais associadas às características, em termos da AIA.

Sequência de características da ocorrência do padrão (<(2,C_1,5)(1,C_4,5)(2,C_5,5)(0,C_{13},4)>):



Matriz das relações ($M_{4.4}$):

eq(C_1,C_1)	f(C_1,C_4)	eq(C_1,C_5)	io(C_1,C_{13})
fb(C_4,C_1)	eq(C_4,C_4)	fb(C_4,C_5)	io(C_4,C_{13})
eq(C_5,C_1)	f(C_5,C_4)	eq(C_5,C_5)	io(C_5,C_{13})
o(C_{13},C_1)	o(C_{13},C_4)	o(C_{13},$C_{5)}$	eq(C_{13},C_{13})

Legenda (eq: equal, f: finishes, o: overlaps, io: is_overlaped_by, fb: finished_by)

Figura 6.4 –Na parte superior é mostrada a representação pictórica dos intervalos temporais de uma ocorrência do padrão #ID = 2 (Sequência <(2,C1,5)(1,C4,5)(2,C5,5)(0,C13,4)>). Na parte inferior é mostrada a matriz (M_{4x4}) que representa as relações dos intervalos em termos da AIA.

Cada linha/coluna da matriz M_{4x4} representa uma terna (característica com seu respectivo intervalo) e deve ser lida da seguinte forma:

- Elementos da diagonal principal da matriz R – representam as relações dos intervalos com eles mesmos, i.e., *eq* = *equal* (igual);

- M[1,2] = f(C_1,C_4) – O intervalo associado à característica C_1 tem relação de *finishes* com o intervalo associado à característica C_4;
- M[1,3] = eq(C_1,C_3) – O intervalo associado à característica C_1 tem relação de *equal* com o intervalo associado à característica C_5;
- M[1,4] = io(C_1,C_{13}) – O intervalo associado à característica C_1 tem relação de *is_overlaped_by* com o intervalo associado à característica C_{13};
- M[2,1] = fb(C_4,C_1) – O intervalo associado à característica C_4 tem relação de *finished_by* com o intervalo associado à característica C_1. Relação inversa de R[1,2];
- M[2,3] = fb(C_4,C_5) – O intervalo associado à característica C_4 tem relação de *finished_by* com o intervalo associado à característica C_5;
- M[2,4] = io(C_4,C_{13}) – O intervalo associado à característica C_4 tem relação de *is_overlaped_by* com o intervalo associado à característica C_{13};
- M[3,1] = eq(C_5,C_1) – O intervalo associado à característica C_5 tem relação de *equal* com o intervalo associado à característica C_1. Relação inversa de R[1,3];
- M[3,2] = f(C_5,C_4) – O intervalo associado à característica C_5 tem relação de *finishes* com o intervalo associado à característica C_4. Relação inversa de R[2,3];
- M[3,4] = io(C_5,C_{13}) – O intervalo associado à característica C_5 tem relação de *is_overlaped_by* com o intervalo associado à característica C_{13};
- M[4,1] = o(C_{13},C_1) – O intervalo associado à característica C_{13} tem relação de *overlaps* com o intervalo associado à característica C_1. Relação inversa de R[1,4];
- M[4,2] = o(C_{13},C_4) – O intervalo associado à característica C_{13} tem relação de *overlaps* com o intervalo associado à característica C_4. Relação inversa de R[2,4];
- M[4,3] = o(C_{13},C_5) – O intervalo associado à característica C_{13} tem relação de *overlaps* com o intervalo associado à característica C_5. Relação inversa de R[3,4].

O processo de geração de regras de associação é executado de maneira similar àquela da Abordagem 1 (ver Seção 6.4.1). Para o experimento descrito nesta seção, o padrão #ID = 2 (ρ = <(C_1)(C_4)(C_5)(C_{13})>) é selecionado e suas

combinações são verificadas quanto aos seus respectivos valores de confiança para só então serem consideradas regras de associação. A Tabela 6.14 apresenta o conjunto de combinações geradas a partir do padrão selecionado, considerando *min_conf* = 0,6.

Como pode ser visto na tabela, a primeira combinação gerada é a #ID = 1 (C_1, C_4, $C_5 \Rightarrow C_{13}$) e como seu valor de confiança é maior que 0,6 (valor de *min_conf*) tal combinação é considerada uma regra de associação. As combinações candidatas a serem regras de associação, com um maior número de consequentes, são então geradas enquanto não for obtida uma combinação com valor menor que 0,6, como mostra a Tabela 6.14 (ver descrição detalhada do processo – Capítulo 2, Seção 2.3).

Tabela 6.14 – Conjunto de combinações geradas a partir do padrão frequente #ID = 2, com seus respectivos valores de confiança. Serão consideradas regras de associação, aquelas combinações com conf ≥ 0,6.

#ID Regra	Regra	Valor de Confiança	Regra Considerada?
1	$C_1, C_4, C_5 \Rightarrow C_{13}$	0,4285 / 0,5714 = 0,7499	Sim
2	$C_1, C_4 \Rightarrow C_5, C_{13}$	0,4285 / 0,5714 = 0,7499	Sim
3	$C_1 \Rightarrow C_4, C_5, C_{13}$	0,4285 /0,7142 = 0,5999	Não

Assim como apresentado anteriormente (ver Capítulo 4), a geração de regras de associação implementada pelo ARMADA busca identificar predições futuras baseadas nos registros passados. Dessa forma, as regras geradas pelo S_MEMISP+AR podem subsidiar, também, predições futuras.

Em outras palavras, a matriz de relações $M_{4x4} = \begin{pmatrix} & C_1 & C_4 & C_5 & C_{13} \\ C_1 & eq & f & eq & io \\ C_4 & fb & eq & fb & io \\ C_5 & eq & f & eq & io \\ C_{13} & o & o & o & eq \end{pmatrix}$, apresentada na Figura 6.4, pode ser utilizada, juntamente com o processo de geração de regras de associação, descrito na Tabela 6.14, para afirmar que a ocorrência do padrão (sequência s = <(2,C_1,5)(1,C_4,5)(2,C_5,5)(0,C_{13},4)>), possibilita a geração da regra #ID = 1, do tipo R: X ⇒ M, em que X é o sub-padrão X $= \begin{pmatrix} & C_1 & C_4 & C_5 \\ C_1 & eq & f & eq \\ C_4 & fb & eq & fb \\ C_5 & eq & f & eq \end{pmatrix}$.

É importante relembrar que a diagonal principal da matriz de relações expressa sempre a relação *equal*, pois descreve a relação do intervalo de estado com ele mesmo e as relações descritas nos elementos M[i][j] são as relações inversas das descritas pelos elementos M[j][i], é possível realizar uma análise/predição considerando apenas os elementos da matriz M acima, ou abaixo da diagonal principal.

De fato, X é frequente, pois é um sub-padrão de um padrão frequente, portanto é possível interpretar tal informação como: "*Se C_1 finishes C_4 e C_1 equal C_5 e C_4 is_finished_by C_5 ocorrerem então há uma grande possibilidade (74,99% de chance) de C_1,C_4 e C_5 is_overlaped_by C_{13} também ocorrerem"*.

Da mesma forma, a regra de associação #ID = 2, do tipo R: X' ⇒ M, em que X' é o sub-padrão X' = $\begin{pmatrix} & C_1 & C_4 \\ C_1 & \text{eq} & \text{f} \\ C_4 & \text{fb} & \text{eq} \end{pmatrix}$. De fato, X' é frequente pois é um sub-padrão de um padrão frequente, portanto é possível interpretar tal informação como: "*Se C_1 finishes C_4 ocorrer então há uma grande possibilidade (74,99% de chance) de C_1 equal C_5 e C_4 is_finished_by C_5 e (C_1,C_4 e C_5) is_overlaped_by C_{13} também ocorrerem"*.

6.5 Comentários Adicionais sobre os Três Experimentos Realizados, com Ênfase em Aspectos Comparativos

No experimento descrito na Seção 6.3, a BDO completa e sequências formadas por apenas dois *itemsets* foram considerados. Embora tal experimento possa ser considerado razoavelmente trivial, os resultados obtidos evidenciam determinados perfis de compras (i.e., transações de compras de produtos em um determinado supermercado). Embora a abordagem não faça distinção entre os dias semana e considera o volume total de itens comprados ao longo dos 7 dias (1500 registros), algumas combinações de itens foram consideradas padrões, tendo ocorrido em um mesmo dia ou em dias diferentes, na dependência de sua frequência.

Com a definição do valor de *min_sup* = 0,05 (5% = 75 transações), foram identificados como padrões frequentes, 29 itens, 36 *itemsets* de tamanho 2, 9 *itemsets* de tamanho 3 e 1 *itemset* de tamanho 4. Relembrado que as combinações geradas levam em consideração a ordem lexicográfica dos itens, os 36 *itemsets* de tamanho 2 que foram validados como frequentes, fazem parte de um total de um total de 406 combinações possíveis, i.e., apenas 8,86% das combinações possíveis são efetivamente frequentes.

Ainda analisando os resultados obtidos no primeiro experimento, foi possível caracterizar que uma grande quantidade de transações incorpora inúmeros itens básicos de consumo alimentar, tais como: arroz (suporte = 0,083), feijão (suporte = 0,0634), açúcar (suporte = 0,0620), café (suporte = 0,0944), leite (suporte = 0,1537), etc. Como descrito anteriormente, tal fato pode estar correlacionado ao período do mês em que a coleta das informações foi feita i.e., o início do mês, em que grande parcela da população recebe o salário e, consequentemente, realiza compras de produtos mais essenciais. Essa conclusão reforça a importância do experimento utilizando a BDO completa, sem considerar os dias da semana, pois traz informações referentes aos itens que compõem as transações registradas na BDO, bem como pode ajudar a detectar tendências mensais de consumo.

O segundo experimento, descrito na Seção 6.4.1, adotou o enfoque de particionar a BDO em 7 blocos, cada um deles contendo registros de compra associados a um particular dia da semana. A principal motivação foi a de tentar estabelecer padrões diários de compra. Porém, ao particionar a BDO em blocos de dados menores, referentes aos dias da semana, algumas peculiaridades, particularmente aquelas envolvendo determinados dias, foram notadas. Combinações que se qualificaram para regras de associação na quarta, sábado e domingo, por exemplo, envolviam um conjunto específico de itens semelhantes, razoavelmente diferenciados dos demais dias. Diferentemente de segunda, terça e quinta, dias em que os padrões identificados são formados, em sua maioria, por itens distintos, tais como frutas, leite, produtos de higiene, etc. Em tais dias, devido à grande diversidade de itens que compõem os padrões, os valores de suporte são, em sua maioria, menores que nos demais dias, não há muitas compras com itens similares.

Tais dias tem como característica, também, o fato de apresentarem as menores quantidades de padrões por dia analisado, com destaque para terça-feira,

com uma quantidade de padrões identificados inferior à quantidade de padrões globais (11 e 12, respectivamente).

Em contraposição aos resultados obtidos no primeiro experimento, considerando os 10 padrões frequente identificados com 3 ou mais itens que os compõem, os padrões globais apresentados no segundo experimento somam uma quantidade de 12 padrões frequentes de mesmo tamanho, i.e., 12 padrões que são frequentes em pelo menos 4 dias da semana. O fato de existirem mais padrões globais do que padrões identificados quando considerada a BDO completa (com todos os registros da semana), reforça a importância de utilizar os resultados de ambos os dois experimentos para a análise dos dados da BDO. Alguns dos padrões identificados no primeiro experimento são frequentes pois se repetem muitas vezes em um único dia, porém não podem ser considerados frequentes durante a semana. Por exemplo, o padrão (presunto-kg,queijo-kg,refrigerante) comparece como um padrão frequente (suporte = 0,0505) nos resultados apresentado pelo primeiro experimento, (ver Tabela 6.6), entretanto, não é considerado um padrão global no segundo experimento. A inversa da afirmativa não pode ser considerada, devido ao fato que todos os padrões considerados globais no segundo experimento foram encontrado nos resultados do primeiro experimento.

O padrão ρ = <(carne-bovina,carne-de-frango,cerveja,refrigerante)> foi, em ambos os 2 experimentos, o padrão frequente de maior tamanho identificado. Nenhum outro do mesmo tamanho, ou maior, apresentou valor de suporte que supera *min_sup* = 0,05. Portanto é possível afirmar que, independente do dia da semana, o padrão ρ é frequente e os itens que os compões são, muitas vezes, comprados juntos por uma grande parte dos clientes que frequentam o estabelecimento.

Unidos à contagem da quantidade de padrões identificados em cada dia, os resultados obtidos nesse experimento colaboram tanto na analise dos resultados do primeiro experimento, quanto na identificação das características que são utilizadas no terceiro experimento.

Na terceira abordagem, a AIA foi utilizada para representar as relações existentes entre os intervalos de duração de cada padrão identificado. Com auxílio dos resultados obtidos nos dois experimentos anteriores, características intrínsecas foram identificadas, manualmente, com seus respectivos intervalos de duração. Um

padrão buscado, então, é composto por várias características que ocorrem durante um determinado dia da semana.

Fica evidente que cada uma das abordagem resulta tanto em padrões quanto regras de associação com vistas diferentes. Enquanto as duas primeiras abordagens buscam padrões considerando os itens que compõem as transações da BDO, a terceira tem como objetivo inferir relações em características identificadas nas transações que compõem a BDO. A utilização dos resultados de todas as abordagens pode, por exemplo, viabilizar a identificação de um perfil de clientes que frequentam o estabelecimento em cada dia da semana.

É natural que os resultados obtidos em ambas, as três, abordagens possuam um certo grau de divergência. Tal fato era esperado pois cada um dos experimentos realizados tem como foco principal a busca de padrões de formas distintas. É importante ressaltar, entretanto, não a divergência de resultados, mas sim as possibilidades que a mineração de padrões temporais disponibiliza ao operador do sistema. Como foi comprovado, cada um dos experimentos realizados e apresentados neste trabalho podem colaborar com alguma informação importante que, possivelmente, seria omitida se apenas uma abordagem fosse considerada.

Considerando que a BDO tem um tamanho relativamente pequeno, visto que considera apenas uma semana de atividades do estabelecimento (1500 registros de compras), o S_MEMISP+AR realizou todo o processo de mineração dos padrões, nos 3 experimentos, em tempos consideravelmente satisfatórios. Desconsiderando os tempos utilizados nas tarefas de pré-processamento, via *script_I* e *script_II*, na melhor performance do primeiro experimento, todos os resultados foram obtidos após 4 minutos e 46 segundos. No segundo experimento, a melhor execução do S_MEMISP+AR foi realizada em 6 minutos e 23 segundos, devido ao fato que 7 bases de dados precisaram ser visitadas. Já o terceiro experimento, ao utilizar uma base de dados reduzida, a BDO-C, foi realizado em apenas 2 minutos e 17 segundos.

Capítulo 7

CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo resume os principais objetivos do trabalho desenvolvido, as atividades realizadas para o desenvolvimento e finalização do trabalho de pesquisa. Aborda também as principais dificuldades encontradas durante o desenvolvimento da pesquisa, as principais conclusões derivadas da investigação realizada e algumas propostas para futuras extensões do que foi descrito neste documento bem como possíveis desdobramentos para a continuação da pesquisa.

7.1 Um Histórico do Desenvolvimento do Trabalho de Pesquisa

O trabalho de pesquisa descrito nesta dissertação teve início em Abril/2013, com reuniões e troca de e-mails entre orientadora e aluno, na busca e familiarização de possíveis áreas de pesquisa de interesse comum, considerando a área de pesquisa de aprendizado de máquina e áreas subjacentes.

Durante a disciplina Estudo Orientado, iniciada em Agosto/2013, o plano de pesquisa de mestrado foi delineado e o processo de levantamento bibliográfico e aquisição dos conceitos básicos relacionados à área de pesquisa escolhida, i.e., mineração de padrões temporais frequentes e regras de associação derivadas teve início.

Durante o levantamento bibliográfico e leitura dos principais artigos na área escolhida, ficou claro a falta de rigidez e de padronização dos principais conceitos e formalismos na área. Isso inicialmente se apresentou como uma grande dificuldade

para aprender a conceituação envolvida. Particularmente, a falta de padronização promoveu, inicialmente, uma desorientação com relação aos objetivos de determinadas propostas; a maioria das propostas analisadas faz uso de uma notação própria o que tornou difícil abordá-las comparativamente.

A proposta ARMADA (WINARKO; RODDICK, 2007) foi escolhida para ser investigada com mais detalhes, por duas razões: ser uma das mais recentes que executa a mineração de padrões considerando intervalos temporais e, também, por ser a mais completa, considerando que, além do tratamento do aspecto temporal, o ARMADA realiza um processo de geração de regras de associação.

Inicialmente ficou decidido abordar a contribuição da proposta MEMISP (LIN; LEE, 2002), que fundamenta a proposta ARMADA e, também, investigar o processo de geração de regras de associação, com base na proposta Apriori (AGRAWAL et al.,1993), como feito pelo ARMADA. Diferentemente do Apriori que segue a abordagem de *Geração e Poda de Candidatos*, tanto o MEMISP quanto o ARMADA são algoritmos que seguem a metodologia de *Crescimento de Padrões* e, portanto, a pesquisa foi expandida de modo a investigar, também, o algoritmo PrefixSpan (HAN et al., 2001), que foi utilizado como base para a proposta MEMISP. O PrefixSpan é uma variação do FreeSpan (HAN et al., 2000), o algoritmo mais antigo da abordagem. Os trabalhos que descrevem ambos os algoritmos, tanto o PrefixSpan quanto o FreeSpan, subsidiaram uma quantidade considerável de definições e conceitos, que foram analisados e serviram como base principal para a padronização da nomenclatura e sequenciamento das definições, como apresentadas no Capítulo 3.

Durante o trabalho de pesquisa, várias outras propostas foram estudadas e analisadas, particularmente aquela relativa ao algoritmo SPADE (ZAKI, 2001), que talvez seja o elo mais evidente entre as duas categorias de algoritmos para mineração de padrões (i.e., *Geração e Poda de Candidatos* e *Crescimento de Padrões*). O SPADE herdou do Apriori o processo de geração e poda de candidatos para os primeiros padrões identificados (de tamanho até 2), realizado, como descreve o autor, de uma forma refinada e mais eficiente. O SPADE faz uso da técnica de *dividir-para-conquistar*, inicialmente utilizada pelos algoritmos FreeSpan e PrefixSpan para a etapa de crescimento de padrões.

Também, durante o trabalho de pesquisa e levantamento bibliográfico, várias aplicações reais de processos de mineração de padrões sequenciais foram

identificadas e analisadas, particularmente aquelas relativas a: levantamento de páginas visitadas (VEERAMALAI et al., 2010) e (BONDE; GORE, 2014) e análise de transações comerciais via Web (OMARI et al., 2007), (YANG; FONG, 2009) e (SIDDIQUI et al., 2014).

Para um planejamento exequível do sistema S_MEMISP+AR, tanto o algoritmo Apriori, quanto o MEMISP precisaram ser estudados e analisados em detalhes. Os resultados desse estudo e análise estão descritos em dois capítulos distintos desta dissertação: o Capítulo 2 e Capítulo 3, respectivamente. Cada um deles traz, também, o pseudocódigo do algoritmo e exemplos de execução.

A efetiva implementação do sistema computacional S_MEMISP+AR foi iniciada em Dezembro/2013. Ao final dessa etapa de desenvolvimento (Março/2014), os resultados obtidos e as pesquisas até então realizadas foram agregadas no documento Exame de Qualificação (realizado em Abril/2014). Após a qualificação, o S_MEMISP+AR continuou a ser desenvolvido, até ser capaz de identificar padrões frequentes em uma base de dados. O experimento com a proposta de utilizar ambos sistemas, MEMISP e S_MEMISP+AR, para confirmação dos mesmos resultados, utilizando uma mesma base de dados, foi realizado e está descrito no Capítulo 5.

A próxima etapa do trabalho teve início em Julho/2014 com a inclusão do tratamento de temporalidade pelo sistema. A escolha pelo uso do formalismo de Höppner (2001) para representar os intervalos temporais da base de dados foi feita considerando que é o formalismo utilizado pelo ARMADA, dado que o ARMADA implementa o tratamento de temporalidade via o formalismo AIA (a notação de Höppner é conveniente para a representação via AIA).

É importante notar, entretanto, que existem várias formas de representar formalmente o aspecto temporal e, muitas vezes, pequenas variações podem impactar os resultados obtidos, tais como a escolha em considerar intervalos ou pontos para representar eventos, adotar intervalos fechados, semiabertos ou abertos, formalismos que podem ser utilizados para representar as relações entre os eventos, etc.

Uma pesquisa com a finalidade de compreender melhor o formalismo de Höppner foi conduzida e as possibilidades de implementação foram analisadas, considerando a base de dados utilizada para realizar os experimentos do sistema proposto, obtida em Agosto/2014. Por se tratar de uma base de dados com registros

de transações comerciais, contendo horário e data dos registros, foi possível identificar vários intervalos relacionados aos registros. As escolhas feitas para incorporar o aspecto temporal ao processo de mineração de padrões, executado pelo S_MEMISP+AR, foram de representar um dia como um intervalo e, também, identificar, de forma manual, características consideradas intrínsecas e de interesse, com seus respectivo intervalos de duração.

Uma dificuldade identificada durante o desenvolvimento foi aquela relacionada aos dados. Nas propostas que descrevem o MEMISP e o ARMADA foram utilizadas bases de dados sintéticas o que, apesar de facilitar o processo de identificação, compromete a validação da eficiência de tais propostas. Para os experimentos com S_MEMISP+AR foi decidido utilizar uma base de dados real, com o objetivo de, também, viabilizar a detecção de dificuldades. O fato da base de dados real disponibilizada não ter muitos registros (aproximadamente 1.500), entretanto, provocou a dificuldade na detecção de padrões significativos. Também, o fato dos registros reais disponibilizados serem relativos a apenas uma semana, encurtou, de certa forma, o escopo na investigação de aspectos temporais; por exemplo, impossibilitou uma investigação de padrões mensais. A busca por uma organização disposta a ceder dados (que geralmente são tratados com uma certa confidencialidade) foi trabalhosa, implicando a visita a várias empresas; apenas uma delas se disponibilizou a fornecer os dados e, ainda assim, apenas relativos a uma semana.

Após todas as definições feitas e com a base de dados disponível, os primeiros experimentos puderam ser executados. Foram realizados três experimentos considerando a mesma base de dados porém seguindo abordagens diferentes:

a) O primeiro experimento considerou a base de dados completa e cada registro de compra contido na BD sendo um *itemset*, constituído pelos produtos da compra. Os resultados obtidos (padrões identificados e regras de associação geradas) identificaram combinações de produtos (itens) que ocorreram várias vezes durante a semana. Para definir um valor de *min_sup* foram realizados vários testes até que o sistema conseguisse manipular toda a quantidade de combinações possíveis sem erros.
b) O segundo experimento considerou o mesmo valor para o *min_sup*, porém a base de dados foi particionada em 7 blocos, cada um deles

contendo registros relativos a um dia particular da semana. A motivação para a implementação de tal abordagem se deve ao fato que, além de identificar padrões e gerar regras de associação particulares a cada dia da semana, os resultados obtidos puderam ser utilizados para refinar aqueles identificados em (a).

c) Subsidiado pela análise dos resultados em (a) e (b), o terceiro experimento foi planejado. Nesse experimento a base de dados foi analisada com auxílio de um especialista do domínio e várias características de interesse (para o processo de decisão associado ao supermercado) puderam ser identificadas. Tais características, com seus respectivos valores temporais de início e fim possibilitaram que o sistema pudesse ser implementado de maneira a usar o formalismo de Höppner para armazenar as informações. As características identificadas foram utilizadas para compor uma nova base de dados que, posteriormente, foi pré-processada via um script em php, (*script_II)* desenvolvido com essa finalidade de customizar os dados para serem tratados pelo S_MEMISP+AR.

O sistema computacional S_MEMISP+AR foi desenvolvido, quase que na totalidade, em linguagem C++, sendo utilizado para tal implementação, o ambiente de desenvolvimento Xcode (versão 6.1.1) fornecido nativamente pela Apple. Foi utilizado um microcomputador com o sistema operacional OS-X (versão 10.9.4), um processador *2.7GHz Intel Core i7* e 8GB de memória RAM.

Durante o processo de implementação do sistema, devido à aridez da linguagem C++ e à grande quantidade de combinações entre os itens que o sistema teve que processar, erros em tempo de execução foram reportados pelo sistema, tais como estouro de pilha e sobrecarga na memória confirmando a necessidade de uma configuração de hardware mais robusta. Essa dificuldade foi um dos fatores que mais impactaram o tempo de desenvolvimento, uma vez que em poucas das execuções realizadas o S_MEMISP+AR conseguiram minerar toda a base de dados sem gerar qualquer erro em tempo de execução.

Para o desenvolvimento dos dois scripts (*script_I* e *scrit_II*), para as tarefas de pré-processamento, foi utilizada a linguagem *php* devido à familiaridade do aluno com tal linguagem para manipular arquivos de texto.

7.2 Conclusões

Como evidenciado anteriormente, existem várias formas de incorporar o tratamento do aspecto temporal ao processo de mineração de padrões. Os três experimentos apresentados no Capítulo 6 comprovam tal afirmação; a tarefa de mineração de padrões temporais, executada pelo S_MEMISP+AR, é bem flexível e, consequentemente, pode ser bastante explorada com a finalidade de combinar resultados obtidos em abordagens diferentes, para uma análise mais refinada do domínio. De fato, as abordagens adotadas nos três experimentos apresentados não são as únicas possibilidades para o domínio analisado, tampouco as melhores escolhas, entretanto, colaboraram para revelar várias informações temporais importantes, implícitas na base de dados analisada. A possibilidade de abordar o tratamento temporal de várias formas colabora na redução de uma possível análise tendenciosa pela interferência do operador do sistema. Evidentemente, com a obtenção de vários resultados é possível que análises diferentes possam ser realizadas levando em conta a real necessidade da mineração de padrões temporais, i.e., quais os tipos de informações q se deseja obter como resultados.

Apesar das dificuldades encontradas no uso do S_MEMISP+AR, nas execuções em que o sistema retornou resultados sem erros o tempo de execução foi relativamente baixo (em média de 4 minutos), fato que comprova a capacidade do sistema em minerar grandes volumes de dados em pouco tempo.

É importante ressaltar que além do S_MEMISP+AR atingir o objetivo que era esperado, i.e., conseguir identificar padrões temporais frequentes e gerar regras de associação a partir de tais padrões identificados, uma outra grande contribuição deste trabalho é a apresentação de um conjunto de definições e formalismos claros e coerentes. Como evidenciado anteriormente, a falta de clareza e padrão nas definições dos estudos analisados foi uma das maiores dificuldades encontradas nesse trabalho e, portanto, é muito importante que esse estudo colabore com trabalhos futuros, facilitando o entendimento de tais conceitos.

7.3 Propostas e Possibilidades Futuras

Durante o desenvolvimento do S_MEMISP+AR, muitas outras possibilidades para realização de experimentos e abordagens foram cogitadas, porém somente algumas foram consideradas e implementadas. Dentre algumas das possibilidades que podem ser implementadas futuramente, como extensão deste trabalho, são as descritas a seguir:

a) Considerar a multiplicidades de ocorrências de uma mesma característica e propor uma nova relação *between* para identificar padrões que ocorrem entre duas ocorrências de uma mesma característica – Desta forma, por exemplo, padrões que ocorrem entre duas ocorrências de *fluxo de vendas alto* podem ser utilizados para identificar produtos que impulsionem as vendas e aumentam o fluxo de vendas;
b) Propor uma nova abordagem para a geração de regras de associação, de forma que as regras considerem os dias da semana. Por exemplo, gerar uma regra do tipo R: s_1,$s_3 \Rightarrow s_5$, que representa a informação que se um padrão é identificado no domingo (s_1) e na terça-feira (s_3) tem uma grande possibilidade de ocorrer, também, na quinta (s_5).
c) Utilizar os padrões identificados em cada dia para definir as características de forma automatizada, reduzindo assim a interferência do operador do sistema/especialista e, consequentemente, as chances de uma análise tendenciosa;
d) De forma similar à abordagem utilizada no terceiro experimento, identificar intervalos temporais em cada dia da semana em que uma quantidade específica de compras é registrada. Cada intervalo representa o tempo necessário em cada dia para se atingir a quantidade de vendas. Dessa forma é possível vincular o padrão ao dia da semana e, possivelmente, identificar os melhores dias de vendas logo nas primeiras horas de funcionamento do estabelecimento.

REFERÊNCIAS

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. N. Mining Associations Rules between Sets of Items in Large Databases. In: International Conference on Management of Data, Washington, DC, maio 1993. **Proceedings…** ACM-SIGMOD: p. 207-216.

AGRAWAL, R.; SRIKANT, R. Fast algorithms for mining association rules in large databases. In: International Conference on Very Large Data Bases, Santiago, Chile, setembro 1994. **Proceedings…** 20th VLDB: 1994, p. 487-499.

AGRAWAL, R.; SRIKANT, R. Mining sequential patterns, 1995. **Proceedings…** 11th International Conference on Data Engineering, 1995, p. 3-14.

ALLEN, J. F. An interval-based representation of temporal knowledge. In: International Joint Conference on Artificial Intelligence, Vancouver, Canada, agosto 1981. **Proceedings…** IJCAI-81, Vancouver, Canada: 1981, p. 221-226.

ALLEN, J. F. Maintaining knowledge about temporal intervals.. **Communications of the ACM**, New York, USA, v. 26, n. 11, p. 832-843, novembro 1983.

ALLEN, J. F.; HAYES, P. J. A common-sense theory of time. In: International Joint Conference on Artificial Intelligence, 1985. **Proceedings…** 9th IJCAI: p. 528-531.

ALLEN, J. F.; HAYES, P. J. Moments and points in an interval-based temporal logic, **Computational Intelligence**, v. 5, n. 3, p. 225-238, setembro 1989.

ALLEN, J. F. Time and time again: the many ways to represent time. **International. Journal of Intelligent Systems**, v. 6, n. 4, p. 341-356, julho 1991.

ALLEN, J. F.; FERGUSON, G. Actions and events in a interval temporal logic. **Journal of Logic and Computation,** New York, USA, v. 4, n. 5, p. 531-579, outubro 1994.

ANTUNES, C. ; OLIVEIRA, A. L, Sequential pattern mining algorithms: Trade-offs between speed and memory. In: Workshop on Mining Graphs, Trees and Sequences at, 2004. **Proceedings…** 15th European ECML and the 8th European PKDD: 2004, p. 3-17.

AYRES, J.; FLANNICK, J.; GEHRKE, J.; YIU, T. Sequential PAttern mining using a bitmap representation. In: International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2002. **Proceedings…** eighth ACM SIGKDD (KDD '02), New York, USA: 2002, p. 429-435.

BELLINI, P.; MATTOLINI, R.; NESI, P. Temporal logics for real-time system specification. **ACM Computing Surveys**, New York, USA, v. 32, n. 1, p. 12-42, março 2000.

BETTINI, C. Time-dependent concepts: representation and reasoning using temporal description logics, **Data & Knowledge Engineering**, v. 22, n. 1, p. 1-38, março 1997.

BÖHLEN, M., H.; BUSATTO, R.; JENSEN, C. S. Point-versus interval-based temporal data models, in: International Conference on Data Engineering, Orlando, Florida, USA, 1998. **Proceedings…** 14th (ICDE'98), IEE Computer Society Press: p. 192–200.

BONDE, A.; GORE, D. V. Comparative Study of Association Rule Mining Algorithms with Web Logs. **(IJCSIT) International Journal of Computer Science and Information Technologies**, v. 5, p. 6153-6157, outubro 2014.

CHITARO, L.; MONTANARI, A. Temporal representation and reasoning in artificial intelligence: Issues and approaches, Amsterdan, Holanda, outubro 2000. **Annals…** Mathematics and Artificial Intelligence, v.28, n. 1-4, p. 47-106.

FURIA, C.; MANDRIOLI, D.; MORZENTI, A.; ROSSI, M. Modeling time in computing: *A taxonomy and a comparative survey*. **ACM Computing Surveys**, New York, USA, v. 42, n. 2, p. 1-59, fevereiro 2010.

GOETHALS, B. ZAKI, M. J, Advances in frequent itemset mining implementations. In: Workshop on Frequent Itemset Mining Implementations, Florida, USA, novembro 2003. **Proceedings…** IEEE ICDM (FIMI'03), Florida, USA: 2003, v. 90.

HAN, J.; PEI, J.; MORTAZAVI-ASL, B.; CHEN, Q.; DAYAL U.; HSU, M. –C. FreeSpan: frequent pattern-projected sequential pattern mining. In: Annual International Conference on Knowledge Discovery in Data, Boston, MA, USA, agosto 2000. **Proceedings…** 6th ACM-SIGKDD, New York, USA: 2000, p. 355-359.

HÖPPNER, F. Learning temporal rules from state sequences. In: Workshop on Learning from Temporal and Spatial Data, Seattle, USA, 2001. **Proceedings…** IJCAI'01, 2001, Seatle: p. 25-31.

IBM Intelligent Miner. Disponível em: <http://www-01.ibm.com/software/data/iminer>. Acesso em: 01 março 2014.

KAO, B.; ZHANG, M.; YIP, C. L.; CHEUNG, D. W.; FAYYAD, U. M. 'Efficient algorithms for mining and incremental update of maximal frequent sequences'. **Data Mining and Knowledge Discovery**. v. 10, p. 87–116, março 2005.

KNIGHT, B.; MA, J. Time representation: A taxonomy of temporal models. **Artificial Intelligence Review**, n. 7, p. 401-419, 1994.

LADKIN, P. Time representation: A taxonomy of interval relations. In: National Conference on Artificial Intelligence, Philadelphia, Pennsylvania, USA, agosto 1986. **Proceedings…** (AAAI-86), Philadelphia, Pennsylvania, USA: p. 360-366.

LIN, M. –Y.; LEE, S. -Y. Fast Discovery of Sequential Patterns by Memory Indexing. Aix-en-Provence, France, setembro 2002. **Proceedings…** 4th International Conference on Data Warehousing and Knowledge, p.150-160.

LONG, D. A review of temporal logics. **The Knowledge Engineering Review**, v. 4, n. 2, p. 141-162, dezembro 1989.

LU, H.; FENG, L.; HAN, J. Beyond intratransaction association analysis: mining multidimensional intertransaction association rules, **ACM Transactions on Information Systems,** v. 18, n. 4, p. 423–454, 2000.

MA, J,; KNIGHT, B. A general temporal theory. **The Computer Journal,** London, v. 37, n. 2, p. 114-123, 1996.

MA, J,; KNIGHT, B. Reified Temporal Logics: An Overview, **Artificial Intelligence Review,** v. 15, n. 3, p. 189-217, maio 2001.

MANI, I.; PUSTEJOVSKY, J.; SUNDHEIM, B. Introduction to the special issue on temporal information processing. New York, USA. **ACM Transactions on Asian Language Information Processing (TALIP)**, v. 3, n. 1, p. 1-10, março 2004.

NICOLETTI, M. C.; LISBOA, F. O. S. S.; HRUSCHKA JR, E. R. Automatic Learning of Temporal Relations Under the Closed World Assumption, **Fundamenta Informaticae,** v. 124, n. 1-2, p. 133-151, junho 2013.

OMARI, A.; CONRAD, S.; ALCIC, S.: Designing a Well-Structured E-Shop Using Association Rule Mining. In: International Conference on Innovations in Information Technology Dubai, novembro 2007. **Proceedings…** 4th ICIIT, Dubai: 2007, p. 18-20.

OZDEN, B.; RAMASWAMY, S.; SILBERSCHATZ, A. Cyclic association rules, In: International Conference on Data Engineering, Orlando, Florida, USA, 1998. **Proceedings…** 14th (ICDE'98), IEE Computer Society Press: 1998, p. 412–421.

PASQUIER, N.; BASTIDE, Y.; TAOUIL, R.; LAKHAL, L. Discovering frequent closed itemsets for association rules. In: International Conference on Database Theory. Jerusalem, Israel, janeiro 1999. **Proceedings…** 7th ICDT: v.1540, 1999, p. 398-416.

PEI, J.; HAN, J.; MORTAZAVI-ASL, B.; PINTO, H.; CHEN, Q.; DAYAL U.; HSU, M. – C. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-projected Pattern Growth. In: International Conference on Data Engineering. Heidelberg, Alemanha, abril 2001. **Proceedings…** 17th ICDE, Heidelberg, Alemanha: 2001, p. 215-224.

SCHOCKAERT, S.; DE COCK, M.; KERRE, E. E. Fuzzifying Allen's Temporal Interval Relations. **IEEE T. Fuzzy Systems**, v. 16, n. 2, p. 517-533, abril 2008.

SIDDIQUI , S.; QADRI, I. Mining Web Log Files for Web Analytics and Usage Patterns to Improve Web Organization, Bhopal, India. **International Journal of Advanced Research in Computer Science and Software Engineering Research**, v. 4, n. 6, junho 2014.

SRIKANT, R.; AGRAWAL, R. Mining Sequential Patterns: Generalizations and Performance Improvements. In: International Conference on Extending Database Technology, Avignon, France, março 1996. **Proceedings…** 5th EDBT, Avignon, France: 1996, p. 3-17.

TAN, P. –N.; STEINBACH, M.; KUMAR, V. **Introduction to Data Mining**, First Edition, Boston, MA, USA: Addison-Wesley Longman Publishing Co. Inc, 2005.

VEERAMALAI, S.; JAISANKAR, N.; KANNAN, A. Efficient Web Log Mining Using Enhanced Apriori Algorithm with Hash Tree and Fuzzy. **International journal of computer science & information technology (IJCSIT)**, India. v. 2, n.4, agosto 2010.

WANG, J.; HAN, J.; PEI, J. CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: International Conference on Knowledge Discovery and Data Mining, Washington, DC, agosto 2003. **Proceedings…** ACM-SIGKDD (KDD'03), Washington: 2003, p. 236-245.

WANG, J.; HAN, J. BIDE: Efficient Mining of Frequent Closed Sequences. In: International Conference on Data Engineering, Boston, MA, USA, abril 2004. **Proceedings…**, 20th ICDE: p.79, 02.

WINARKO, E.; RODDICK, J. F. ARMADA – An algorithm for discovering richer relative temporal association rules from interval-based data. **Data & Knowledge Engineering**, v. 63, n.1, p. 76-90, outubro 2007.

YAH, X.; HAN, J.; AFSHAR, R. CloSpan: Mining Closed Sequential Patterns in Large Databases. In: SIAM International Conference on Data Mining, San Francisco, CA, maio 2003. **Proceedings…** SDM'03: 2003, p.166-177.

YANG, Z.; KITSUREGAWA, M. LAPIN-SPAM: An improved algorithm for mining sequential pattern. In: International Conference on Data Engineering Workshops, abril 2005. **Proceedings…** 21st ICDE Workshops: 2005, p. 1222.

YANG, H.; FONG, S. A Framework of Business Intelligence-driven Data Mining for e-Business. In: International Conference on Networked Computing, Seoul, Korea, agosto 2009. **Proceedings…** 5th ICNC: 2009, p.1964-1970.

ZAKI, M. J. Spade: An efficient algorithm for mining frequent sequences. **Machine Learning**, Hingham, USA, v. 42, n. 2, p. 31–60, janeiro 2001.

ZAKI, M.; HSIAO, C. CHARM. An efficient algorithm for closed itemset mining. In: 2nd SIAM International Conference on Data Mining, Arlington, 2002. **Proceedings…** SDM'02, Arlington: 2002, p. 457-473.

ZHANG, M.; KAO, B.; Yip, C.; CHEUNG, D. A GSP-based efficient algorithm for mining frequent sequences. In: International Conference on Artificial Intelligence, Las Vegas, Nevada, USA, 2001. **Proceedings…** ACAI'01, Las Vegas: 2001, p. 497–503.

Apêndice A

SCRIPT_I – PRIMEIRO PRÉ PROCESSAMENTO DA BDO

```
1.  //  Script I de pré processamento da BDO
2.  //  S_MEMISP+AR
3.  //  Created by Rafael Stoffalette João on 11/1/14.
4.  //  Copyright (c) 2015 Rafael Stoffalette João. All rights reserved.
5.
6. if(isset($primeira_etapa_processamento)){
7. require_once($_SERVER['DOCUMENT_ROOT'] . "../includes/excel_reader2.php");
8.                                                  // plugin para leitura de arquivos xls via php
9. $coluna = 3;
10.                 // disponível em http://code.google.com/p/php-excel-reader/
11.     $fp = fopen("bdo.txt", "w");  // arquivo de saída
12.     for($numarqs = 1; $numarqs < 1500; $numarqs++){
13.                  // cada registro de compra tem como nome os numeros de 1 a 1500
14.       $cesta =  Array();                                    // referente a cada identificador da compra
15.       $data = new Spreadsheet_Excel_Reader($numarqs.'.xls'); //abre cada um dos registros de compra
16.       $totalLinhas = $data->rowcount();               // calcula quantos itens tem na compra (n. de  linhas)
17.       for ($i = 8; $i <= $totalLinhas; $i++) {            // coloca cara item no array $cesta
18.                 if($data->val($i,3) != "")
19.                           array_push($cesta,$data->val($i,3));
20.       }
21.
22.       sort($cesta);                                     // ordena os itens da compra
23.       $dado = $data->val(5,1);                                  // elemento [5][1], data da compra
24.       $dado = strtolower(str_replace(' ', '', $dado));          // remove espaços em branco da data
25.       $dado = str_replace(".", "", $dado);                      // remove pontos da data
26.       fwrite($fp,"\n(");                                // insere uma quebra de linha (cada compra em uma linha)
27.       fwrite($fp,$dado);                                // grava a data como um item no arquivo
28.       fwrite($fp,",");                                          //  insere o separador de itens ","
29.       $dado = $data->val(5,2);                          // elemento [5][1], hora da compra
```

```
30.          $dado = strtolower(str_replace(' ', '', $dado));            // remove espaços em branco da hora
31.          $dado = str_replace(".", "", $dado);              // remove pontos da hora
32.          fwrite($fp,$dado);                        // grava a hora como um item no arquivo
33.          fwrite($fp,")(");                         // finaliza o itemset da data,hora e inicia o de itens da compra
34.
35.          foreach ($cesta as $key => $value) {            // para cada item da compra
36.            $string = strtolower(str_replace(' ', '', $value));  // remove espaços em branco do item
37.            $palavra = ereg_replace("[^a-zA-Z0-9_]", "",$string);  // remove caracteres especiais da descrição
38.            $string = str_replace(".", "", $palavra);                  // remove os pontos da descrição do item
39.            fwrite($fp, trim($string));                                // grava o item da compra  no arquivo
40.            if(!end($cesta)) fwrite($fp, ",");
41.            // se não for o último elemento do array insere uma virgula para separar os itens
42.        }
43.          fwrite($fp, ")");               //finaliza o itemset com ")"
44.        }
45.        fclose($fp);                      // fecha o arquivo
46 .}else{
47.
48.         $lines = file ('bdo.txt');                  // arquivo de entrada (BDO em seu estado bruto)
49.        if(isset($_POST['novaDB'])){ // procedimento para gerar o arquivo de saída, com a BDO processada
50.                  $fp = fopen("bdo-processado.txt", "w");                    // arquivo de saída
51.                  foreach ($lines as $line_num => $line) {         // percorrendo as linhas (compras) da BDO
52.                           $array  = explode(',', $line);   // converte cada registro de compra em um array de itens
53.                           sort($array);                    //ordena lexicograficamente os itens dentro do array
53.                           $result = array_unique($array);         // elimina as redundâncias
55.                           $novalinha = implode(',', $result);
56.                           // transforma o array ordenado e com itens sem repetição em uma nova linha
57.                           fwrite($fp, $novalinha);        // grava a linha no arquivo de saída (BDO processada)
58.                  }
59.                  fclose($fp);
60.        }
61.        elseif(isset($_POST['trocaNome'])){              // procedimento para substituir nomes de itens chamado
62.                  $selected = $_POST['trocaNome'];  // Conjunto de nomes selecionados para
63.                  $novoNome = $_POST['novoNome'];   // novo nome que irá representar os itens selecionados
64.                  $path_to_file = 'bdo.txt';                         // arquivo de entrada (BDO em seu estado bruto)
65.                  $file_contents = file_get_contents($path_to_file); // a BDO inteira é copiada para uma variavel
66.
67.                  foreach ($selected as $key => $value) {        //para cada item selecionado a ter o nome trocado
68.                           $search_string  = $value;
69.                           $replace_string = $novoNome;
70.                           $file_contents = str_replace($search_string,$replace_string,$file_contents);
71.                  }           // é feito a busca e substituição dos nomes
72.                  file_put_contents($path_to_file,$file_contents);
73.        }           // a variável que contém os itens com novos nomes é inserida no arquivo de volta
74.
```

```
75.
76.         echo "\n---------------------------------BDO-------------------------------\n";
77.        $cesta = array();
78.        foreach ($lines as $line_num => $line) {        // percorrendo as linhas (compras) da BDO, uma a uma
79.          $array  = explode(',', $line);                 // converte cada registro de compra em um array de itens
80.
81.
82.          foreach ($array as $key => $value) {                    // para cada item dentro de cada compra da BDO
83.            array_push($cesta,htmlspecialchars($value));          // armazena cada item no array $cesta
84.          }
85.        }
86.        sort($cesta);                                       // ordena lexicograficamente os itens dentro do array
87.        $cont=0;
88.        foreach ($cesta as $chave => $valor) {          // percorre a lista com todos os itens contidos na BDO
89.                    $cont++;
90.                    $valor  = str_replace("(", "",$valor);    // ignora "("
91.                    $valor  = str_replace(")", "",$valor);   // ignora ")"
92.                    $valor  = str_replace(",", "",$valor);   // ignora ","
93.                    echo"<br /><input type='checkbox' name='trocaNome[]' value='".$valor."' />";
94.                    echo " ".$cont." - ".$valor; // imprime cada item de forma ordenado e com uma caixa de seleção
95.        }                                                   // a fim de possibilitar selecionar o item e trocar seu nome.
96 .}
```

Apêndice B

SCRIPT_II – PRÉ PROCESSAMENTO DA BDO A FIM DE CONSTRUIR A BDO-C

```
1.      //  Script II de pré processamento da BDO
2.      //  S_MEMISP+AR
3.      //  Created by Rafael Stoffalette João on 11/2/14.
4.      //  Copyright (c) 2015 Rafael Stoffalette João. All rights reserved.
5.
6.
7.
8.  if(isset($_POST['GeraNovaDB'])){ // procedimento para gerar o arquivo de saída, com a BDO processada
9.                  $lines = file ('bdo-bruta.txt');                    // arquivo de entrada (BDO em seu estado bruto)
10.                 $fp = fopen("bdo-c.txt", "w");                      // arquivo de saída
11.                 $primeiraSeq = 1;
12.
13.     foreach ($lines as $line_num => $line) {        // percorrendo as linhas (ocorrências de características)
14.                         $array  = explode(',', $line);          // converte cada registro em um array de itens
15.                         $dia  = $array[0];              // o primeiro elemento é o dia.
16.                         $t_inicio = $array[1];          // ponto no tempo em que a característica se inicia.
17.                         $t_fim = $array[2];             // ponto no tempo em que a característica termina.
18.                         $nome_carac = $array[3];  // qual a característica ocorrida.
19.
20.                         if($dia_ant != $dia){ // novo dia, nova sequencia
21.                         // Cada registro de característica é gravado, no formalismo de Höppner
22.                         if($primeiraSeq == 1){ // no primeiro registro da BD cria-se a primeira sequência
23.                                         $primeiraSeq=0;
24.                                         $caracteristica  = "\n<(".$t_inicio.",".$nome_carac.",".$t_fim.")";
25.                                 }
26.     //caso contrário é preciso colocar o marcador de final de sequência (>) antes de criar uma nova.
27.                         else $caracteristica  = ">\n<(".$t_inicio.",".$nome_carac.",".$t_fim.")";}
28.                 // Caso a caracteristica seja do mesmo dia da característica lida na iteração anterior
29.     // não é preciso criar uma nova sequência, apenas incluir uma nova ocorrência de característica.
```

```
30.                         else
31.                                 $caracteristica = "(".$t_inicio.",".$nome_carac.",".$t_fim.")";
32.                         $dia_ant =  $dia;
33.                         $novoRegistro.=$caracteristica;
34.
35.                 }
36.                 $caracteristica.=">";
37.                 $novoRegistro =  $caracteristica;
38.                 fwrite($fp, $novoRegistro);  // grava a linha no arquivo de saída (BDO processada)
39.                 fclose($fp);
40.         }
```