Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

MINERAÇÃO DE PADRÕES SEQÜENCIAIS MÚLTIPLOS

Daniel Antônio Furtado

Uberlândia - MG
Junho de 2005

MINERAÇÃO DE PADRÕES SEQÜENCIAIS MÚLTIPLOS

Por

Daniel Antônio Furtado

DISSERTAÇÃO APRESENTADA À
UNIVERSIDADE FEDERAL DE UBERLÂNDIA, MINAS GERAIS,
COMO PARTE DOS REQUISITOS EXIGIDOS
PARA OBTENÇÃO DO TÍTULO DE MESTRE
EM CIÊNCIA DA COMPUTAÇÃO

JUNHO DE 2005

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Mineração de Padrões Seqüenciais Múltiplos**" por **Daniel Antônio Furtado** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 13 de junho de 2005

Orientador: _____

Profª. Drª. Sandra de Amo

Universidade Federal de Uberlândia UFU/MG

Banca Examinadora: _____

Prof. Dr. Ilmério Reis da Silva

Universidade Federal de Uberlândia UFU/MG

_____

Prof. Drª. Marina T. Pires Vieira

Universidade Federal de São Carlos UFSCar/SP

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Data: Junho, 2005

Autor: **Daniel Antônio Furtado**

Título: **Mineração de Padrões Seqüenciais Múltiplos**

Faculdade: **Faculdade de Computação**

Grau: **Mestrado**

_____

Autor

Dedicatória

A meus pais Rosário e Elza e a meu irmão Domiro.

A meu amigo Diogo Nunes.

Agradecimentos

Gostaria de expressar minha profunda gratidão à minha orientadora e conselheira Sandra de Amo. Agradeço por sua contínua orientação durante a pesquisa e redação desta dissertação, por suas idéias e conselhos, pelo incentivo, pelo tempo dedicado a meu favor, pela confiança, pelas oportunidades oferecidas e por sua amizade.

Sou grato à todos os colegas e amigos do Laboratório de Computação Científica da Universidade Federal de Uberlândia que se mostraram companheiros durante esses dois anos de estudo, em especial à Saulo Damasceno, à Felipe Rezende pelo trabalho realizado nos geradores de dados sintéticos, à Luiza Rangel e Lucas Vallinotto pelo trabalho na interface gráfica do programa minerador e à Elaine Ribeiro, pelos conselhos e apoio.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, que financiou o desenvolvimento desde trabalho.

Finalmente, gostaria de agradecer a meus pais Rosário Furtado Paim e Elza Helena Soares Paim pelo amor incondicional e por encorajarem-me continuamente.

"Nunca seremos suficientemente gratos a Deus,

aos nossos pais e aos nossos mestres!"

(Aristóteles)

Resumo

A descoberta de padrões seqüenciais constitui um importante problema em mineração de dados e possui aplicações nas mais diversas áreas tais como mercado financeiro, medicina, análise de mercado, telecomunicações, comércio eletrônico, etc. A maioria das pesquisas já realizadas sobre a mineração de padrões seqüenciais concentra-se na descoberta de padrões temporais que podem ser especificados, de alguma maneira, na Lógica Temporal *Proposicional*. Entretanto, existem alguns padrões seqüenciais interessantes que necessitam de um formalismo mais expressivo, o da Lógica Temporal de *Primeira Ordem*.

Nesta dissertação estamos propondo um novo padrão temporal, que denominamos de *padrão seqüencial múltiplo*, que é um padrão temporal de primeira ordem e tem como objetivo representar o perfil de indivíduos/objetos relacionados entre si, ao longo do tempo. Nosso padrão possui aplicações em várias áreas, como no mercado financeiro e no varejo. Propomos dois algoritmos para efetuar a mineração de *todos* esses padrões freqüentes em um banco de dados: o algoritmo PM (*Projection Miner*), que realiza a mineração decompondo o padrão de primeira ordem em componentes proposicionais e adapta idéias do algoritmo GSP (que minera padrões seqüenciais proposicionais); e o algoritmo SM (*Simultaneous Miner*), que efetua a mineração do padrão de primeira ordem sem decompô-lo. Nossos resultados experimentais mostram que a performance de SM é superior a de PM.

Também exploramos um mecanismo que permite o controle por parte do usuário com relação aos padrões múltiplos que são minerados. Propomos o algoritmo *MSP-Miner*, que incorpora no processo de mineração uma restrição especificada pelo usuário através de expressões regulares. *MSP-Miner* encontra somente os padrões múltiplos satisfazendo a restrição informada. A performance e a escalabilidade desse algoritmo foi avaliada através de um conjunto de testes realizados em bancos de dados sintéticos.

Abstract

Discovering sequential patterns is an important problem in data mining with a lot of application domains including financial market, medicine, retailing, telecommunications, e-commerce, etc. Previous studies on mining sequential patterns have focused on temporal patterns specified by some form of *propositional* temporal logic. However, there are some interesting sequential patterns whose specification needs a more expressive formalism, the *first-order* temporal logic.

In this dissertation, we propose a new temporal pattern, called *multi-sequential pattern*, which is a first-order temporal pattern (not expressible in propositional temporal logic) and aims at representing the behaviour of individuals/objects *related to each other* by some criteria, throughout time. Our pattern appears in many application domains, like financial market and retailing. We propose two Apriori-based algorithms to find *all* frequent patterns in a given dataset: the PM algorithm (Projection Miner), that performs the mining task by projecting the first-order pattern in two propositional components and adapts the key idea of the classical GSP algorithm (for propositional sequential pattern mining); and the SM (Simultaneous Miner) algorithm, that finds out the first-order pattern without decomposing it. Our extensive experiments shows that SM scales up far better than PM.

Beyond that, we extend a well-known user-controlled tool, based on regular expressions constraints, to the multi-sequential pattern context. This specification tool enables the incorporation of user focus into the multi-sequential patterns mining process. We also present MSP-Miner, an Apriori-based algorithm to discover all frequent multi-sequential patterns satisfying a user-specified regular expression constraint. We perform detailed experiments on synthetic data to study the performance and scalability of MSP-Miner.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O crescimento explosivo da quantidade e do tamanho dos bancos de dados existentes nas mais diversas áreas, como no comércio, na indústria, no governo, na Internet, etc, tem ultrapassado substancialmente a nossa capacidade de interpretar e compreender esses dados. Temos em mãos grandes quantidades de dados, mas muitas vezes ainda estamos sedentos do conhecimento em si. A *mineração de dados* surge então como uma ferramenta poderosa para suprir a necessidade de descobrir informações úteis, tais como padrões escondidos em grandes bancos de dados.

A mineração de padrões freqüentes é uma área de pesquisa bastante ativa em mineração de dados. Ela engloba várias tarefas de mineração, incluindo a mineração de regras de associação e a mineração de padrões seqüenciais.

O problema de descobrir padrões seqüenciais em dados temporais tem sido bastante estudado em vários artigos [1, 2, 3, 4] e sua importância é seguramente justificada pelo grande número de setores onde a mineração de padrões seqüenciais pode ser aplicada com êxito, tais como o mercado financeiro (evolução de cotações de ações), o varejo (evolução de compras de clientes), a medicina (evolução dos sintomas dos pacientes), a previsão do tempo, etc. Diferentes tipos de padrões seqüenciais e padrões temporais generalizados [5, 6, 7, 8] já foram propostos, assim como formalismos e inúmeros algoritmos para expressar e minerar esses padrões [9, 10, 2, 11, 12, 13, 14]. A maioria desses padrões pode ser especificada através de formalismos que, de alguma forma, podem ser reduzidos à Lógica Temporal Proposicional. Por exemplo, considere o padrão seqüencial (proposicional) da

forma $< i_1, i_2, \ldots, i_n >$ (onde i_j, $j \in \{1, ..., n\}$, são conjuntos de itens) que já foi bastante estudado na literatura nos anos passados [1, 2, 11, 12, 13, 14]. Esse padrão é considerado *freqüente* em um banco de dados de transações de clientes se há uma porcentagem α de clientes que compraram os conjuntos de itens i_j seqüencialmente, isto é, os itens de i_1 são comprados no instante t_1, os itens de i_2 são comprados no instante t_2 e assim por diante, com $t_1 < t_2 < ... < t_n$. Se denotamos por p_k^j $(k = 1, ..., n_j)$ as variáveis proposicionais representando os itens no conjunto de itens i_j, então este padrão pode ser expresso na Lógica Temporal Proposicional pela fórmula $i_1 \wedge \Diamond(i_2 \wedge \Diamond(i_3 \wedge (\ldots \wedge \Diamond i_n)...))$, onde i_j é a fórmula $(p_1^j \wedge p_2^j \wedge \ldots p_{n_j}^j)$ e $\Diamond$ é o operador temporal *"em algum momento no futuro"*.

Nesta dissertação estamos propondo um novo padrão temporal, que denominamos de *padrão seqüencial múltiplo*, assim como três algoritmos para minerá-lo. Diferentemente dos padrões seqüenciais já estudados, nosso padrão não pode ser expresso através da Lógica Temporal Proposicional e necessita do maior poder de expressividade da Lógica Temporal de Primeira Ordem.

1.1 Motivação

Um *padrão seqüencial múltiplo* aparece em vários domínios de aplicações, tais como no mercado financeiro, no varejo e de alguma forma, tem como objetivo representar o perfil de indivíduos/objetos relacionados entre si por algum critério, ao longo do tempo. A seguir são dados dois exemplos de situações onde aparecem os padrões seqüenciais múltiplos.

Exemplo 1.1.1 As cotações das ações x e y de uma *mesma* empresa freqüentemente apresentam o seguinte comportamento: um incremento de n pontos de x é seguido por um aumento de m pontos de y e um posterior decremento de k pontos de x. Tal seqüência de eventos representa um padrão seqüencial múltiplo expressando o comportamento *relativo* das ações de uma *mesma* empresa. Assim, estamos interessados em saber se existem muitos *grupos* formados por ações de uma mesma empresa dentro dos quais existem duas ações que apresentam o comportamento descrito acima.

Exemplo 1.1.2 Agora, suponha que estamos interessados em descobrir como o ambiente de trabalho pode influenciar no perfil de compra dos clientes. Por exemplo, considere que

clientes trabalhando em um *mesmo local* são agrupados e que tais grupos são armazenados na tabela G abaixo. Além disso, considere a tabela Tr armazenando as transações efetuadas por esses clientes durante um determinado período. O atributo T informa o instante em que a transação foi realizada. Para simplificar a representação, consideramos que cada transação contém apenas um item, ao invés de um conjunto de itens.

Tr

Cliente	*T*	*Item*	*Cliente*	*T*	*Item*
Paul	1	Comp. MX	Charles	7	DVD
Mary	2	Comp. MY	Susan	8	DVD
Sally	2	VCR	John	8	TV
Paul	3	Impres. MZ	Gina	9	Câmera
Mary	5	Impres. MW	Gloria	10	TV
Charles	6	TV	Bill	11	Celular
Susan	4	TV	Frank	12	VCR

G

IdGr	*Grupo*
1	{Paul,Mary,Sally}
2	{Charles,Susan}
3	{John,Gina}
4	{Gloria,Bill,Frank}

Considere ainda, a seguinte seqüência de transações realizadas por dois clientes:

(1) o ***primeiro*** *cliente compra um computador modelo MX, (2) depois disso, o* ***segundo*** *cliente compra um computador modelo MY, (3) em seguida, o* ***primeiro*** *cliente compra uma impressora MZ e (4) finalmente o* ***segundo*** *cliente compra uma impressora MW.*

Observe que tais seqüências de transações são efetuadas pelos clientes *Paul* e *Mary* pertencentes ao grupo 1 da tabela de grupos G. Se o modelo MY é mais sofisticado que MX e o modelo MW também é mais sofisticado que MZ, então a seqüência acima poderia expressar de alguma forma que o ambiente de trabalho exerce influência *competitiva* ou *gananciosa* no perfil de compra de seus clientes, isto é, *"se meu colega comprou tal produto, eu comprarei um produto melhor"*. Se os modelos MX e MY são exatamente os mesmos, assim como os modelos MZ e MW então a seqüência acima poderia indicar de alguma forma que o ambiente de trabalho exerce uma influencia de *confiança* com relação às compras dos clientes, isto é, *"se meu colega comprou esse produto, eu confio em sua escolha e comprarei o mesmo produto"*. Se tal seqüência de transações fosse realizada por clientes de pelo menos um número mínimo α de grupos (α é informado pelo usuário), então essa seqüência corresponderia a um *padrão seqüencial múltiplo*. Assim, neste exemplo estamos interessados em saber se existem muitos *grupos* formados por clientes que *trabalham em um mesmo lugar* dentro dos quais *existem* pelo menos 2 clientes

que realizam tal seqüência de transações. Observe que o padrão seqüencial expresso aqui não pode ser especificado por nenhuma fórmula da Lógica Temporal Proposicional. A especificação do mesmo pode ser feita através da seguinte fórmula da Lógica Temporal de Primeira Ordem: $\exists c_1 \exists c_2 (grupo(c_1, c_2) \wedge compra(c_1, C.MX) \wedge \Diamond (compra(c_2, C.MY) \wedge \Diamond\ (compra(c_1, I.MW) \wedge \Diamond\ compra(c_2, I.MZ))))$.

De forma análoga, o padrão seqüencial múltiplo expresso no exemplo 1.1.1 pode ser especificado através da seguinte fórmula temporal de primeira ordem: $\exists x_1 \exists x_2 (grupo(x_1, x_2) \wedge sobe(x_1, n) \wedge \Diamond (sobe(x_2, m) \wedge \Diamond\ abaixa(x_1, k)))$.

1.2 Contribuições

Em nossa opinião, as principais contribuições desta dissertação podem ser descritas como segue:

- Estamos propondo um novo padrão temporal, denominado de *padrão seqüencial múltiplo* (*psm*), que possui aplicações em áreas como no mercado financeiro e varejo e que, diferentemente dos padrões seqüências já estudados, necessita de um formalismo mais poderoso para ser expresso: o formalismo da Lógica Temporal de Primeira Ordem.

- Além de propor um novo padrão, também desenvolvemos dois algoritmos para minerar *todos* os *psm's* freqüentes em um dado banco de dados. São os algoritmos **PM** (***P**rojection **M**iner*) e **SM** (***S**imultaneous **M**iner*). O primeiro é baseado em uma técnica que decompõe o padrão em dois padrões seqüenciais proposicionais, durante as fases de geração e poda, e o processo de mineração é alcançado através de uma adaptação do algoritmo GSP (que minera padrões seqüenciais proposicionais). No segundo algoritmo, uma técnica de mineração é executada sem decompor o padrão. Comprovamos através dos resultados experimentais, que SM apresenta melhor desempenho do que PM. Nesse aspecto, mostramos que uma técnica convencional para mineração de padrões seqüenciais (proposicionais) pode ser adap-

tada para minerar padrões temporais de primeira ordem, mas que uma técnica de "*primeira ordem pura*" (SM) produz melhores resultados.

- Também exploramos o problema de incorporar restrições especificadas pelo usuário no processo de mineração para o contexto de padrões seqüenciais múltiplos. Nossas restrições são especificadas por meio de expressões regulares.

- Propomos um terceiro algoritmo (*MSP-Miner*) para mineração desses padrões de maneira que a restrição especificada seja incorporada no processo de mineração.

1.3 Organização da Dissertação

O conteúdo desta dissertação está organizado em três partes, como descrito a seguir.

Na primeira parte apresentaremos o estado da arte. O capítulo 2 introduz alguns conceitos básicos em mineração de dados e tarefas de mineração e o capítulo 3 aborda os principais trabalhos relacionados e descreve alguns dos algoritmos já propostos para mineração de padrões seqüenciais (simples).

Na parte II exploramos o problema de mineração de padrões seqüenciais múltiplos sem restrições. O problema de mineração é formalizado no capítulo 4, no capítulo 5 apresentamos o algoritmo PM para mineração de *psm's* e no capítulo 6 apresentamos o algoritmo SM. Os resultados experimentais de ambos algoritmos são mostrados e analisados no capítulo 7.

Na parte III, estendemos o problema de mineração de padrões seqüenciais múltiplos para considerar restrições impostas pelo usuário. O problema é formalizado no capítulo 8, no capítulo 9 apresentamos o algoritmo MSP-Miner para mineração dos padrões de acordo com a restrição imposta e no capítulo 10 são apresentados os resultados experimentais para o algoritmo MSP-Miner.

Finalmente, concluímos o trabalho no capítulo 11 e apresentamos algumas perspectivas para pesquisas futuras.

Parte I

Estado da Arte

Capítulo 2

Mineração de Dados

Com o rápido crescimento do tamanho e do número de bancos de dados disponvíveis em aplicações comerciais, na indústria, na Internet, entre muitas outras aplicações, surge o interesse e a necessidade de estudar como extrair conhecimentos automaticamente de grandes bases de dados [15, 16]. Com a extração de conhecimentos em grandes bancos de dados, tais bancos podem ser vistos como verdadeiras e ricas fontes de geração de conhecimentos e os conhecimentos por meio deles obtidos podem ser aplicados para gerenciamento de informações, processamento de consultas, tomadas de decisões e para muitos outros fins.

A mineração de dados ou *data mining* é apenas uma etapa de todo o processo de descoberta de conhecimentos em bancos de dados. Brevemente, a mineração de dados corresponde em aplicar a uma base de dados adequadamente preparada, algoritmos e técnicas para descoberta de padrões interessantes nesses dados. O processo como um todo, em que conhecimentos interessantes são obtidos em grandes bancos de dados é explicado em mais detalhes na próxima seção.

2.1 O Processo de Descoberta de Conhecimento

O processo de **Descoberta de Conhecimentos em Bancos de Dados** (*Knowledge Discovery in Databases* - KDD) é definido em [15] como *o processo não trivial de identificação de padrões válidos, novos, potencialmente úteis e compreensíveis em dados.* Em

tal definicão, *dados* são um conjunto de fatos e *padrão* corresponde a uma expressão em alguma linguagem descrevendo um subconjunto dos dados. O termo *processo* informa que KDD compreende vários passos, envolvendo a preparação dos dados, a busca por padrões e a avaliação do conhecimento obtido, podendo tais passos serem repetidos em múltiplas iterações. O termo *não trivial* indica que o processo não é direto e pode envolver o uso de técnicas de busca e algoritmos.

As várias etapas que constituem o processo KDD podem ser resumidas em: (1) *seleção dos dados*, (2) *pré-processamento e transformação*, (3) *mineração de dados* e (4) *interpretação e avaliação dos resultados*. A figura 2.1 ilustra esses passos, que são brevemente descritos em seguida.

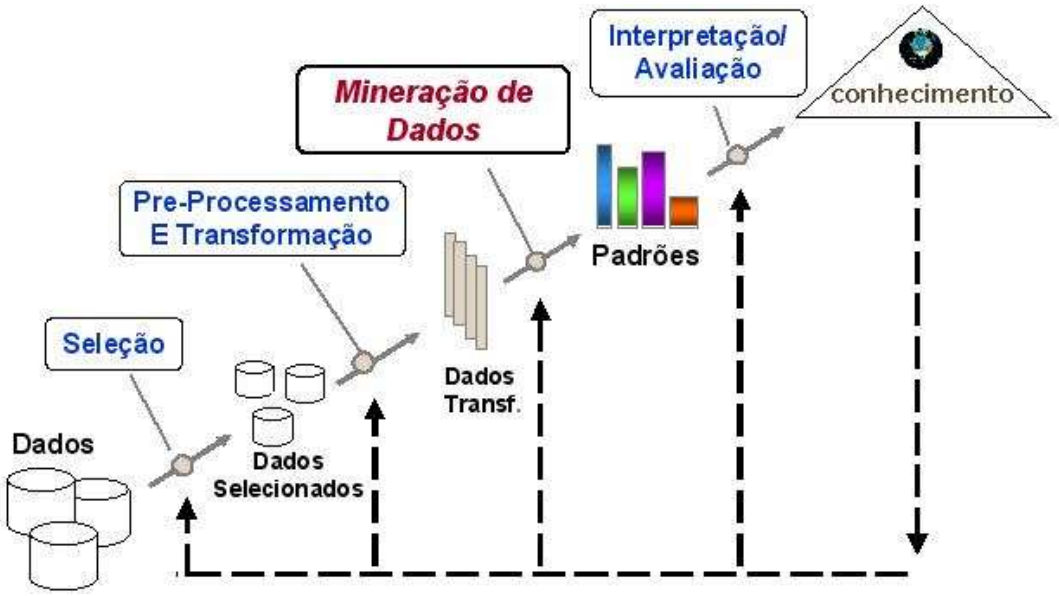


Figura 2.1: Etapas do KDD *(modificado de [15])*

Na etapa da *seleção dos dados*, os mesmos podem ser obtidos de várias fontes, sendo selecionados aqueles que são viáveis à descoberta do conhecimento. Uma limpeza dos dados é realizada na etapa de *pré-processamento e transformação* (por exemplo, podem ser descartados registros contendo atributos com valores incorretos). Também nesta fase pode ser realizada a conversão dos dados para um formato comum (pode ser necessária uma normalização, por exemplo). Na seqüência, temos a etapa que corresponde ao núcleo de todo o processo, que é a *mineração de dados*. Nessa etapa são executados os algoritmos

de mineração, que são específicos para a tarefa de mineração em questão. Os padrões obtidos aqui serão interpretados e avaliados na última etapa, sendo mostrados para o usuário de maneira compreensível ao mesmo.

2.2 Tarefas de Mineração de Dados

São várias as tarefas de mineração de dados já propostas [17, 1, 4, 18, 15], assim como as metodologias e algoritmos para melhor solucionar cada problema de mineração. A seguir, descrevemos brevemente algumas das principais tarefas de mineração de dados.

1. **Classificação.** A tarefa de classificação corresponde em fazer a categorização de dados em um conjunto de classes conhecidas. Um banco de dados de treinamento (isto é, um conjunto de objetos em que se conhece as classes às quais os mesmos pertencem) é dado e analisado, e um modelo de classificação é construído baseado nas características dos dados no banco de dados de treinamento. Um conjunto de regras de classificação é então gerado do modelo de classificação, as quais podem ser usadas para classificar novos dados. Por exemplo, regras de classificação sobre doenças podem ser extraídas de casos conhecidos (dados de treinamento) e usadas para diagnosticar novos pacientes através de seus sintomas [19, 20, 21].

2. **Agrupamento (*Clustering*).** Agrupamento é o processo de agrupar um conjunto de objetos em classes de objetos semelhantes, de acordo com seus atributos. No agrupamento não existem classes pré-definidas (como na classificação); estas são obtidas no decorrer do processo. Os grupos são formados de maneira que os objetos de um mesmo grupo possuam maior similaridade entre si e objetos de grupos diferentes possuam baixa similaridade. Por exemplo, os links retornados por um site de busca em resposta a uma consulta podem ser agrupados de acordo com o grau de similaridade entre os documentos encontrados [18, 22, 23].

3. **Mineração de Regras de Associação.** Corresponde à descoberta de associações ou relacionamentos entre objetos. Uma regra de associação possui o seguinte formato: $A_1 \wedge \cdots \wedge A_i \rightarrow B_1 \wedge \cdots \wedge B_j$ e informa que os objetos $B_1 \wedge \cdots \wedge B_j$ tendem

a aparecer com os objetos $A_1 \wedge \cdots \wedge A_i$ em um dado banco de dados. Por exemplo, com a mineração de regras de associação pode-se descobrir um conjunto de produtos em um supermercado que é freqüentemente vendido com outro grupo de produtos. Tal informação pode ser útil, por exemplo, para melhor estruturação dos produtos nas prateleiras do supermercado [17, 24, 25, 26, 27].

4. **Mineração de Padrões Seqüenciais.** A mineração de padrões seqüenciais tem como objetivo identificar seqüências de eventos (ou objetos) que ocorrem freqüentemente em bancos de dados temporais. Os padrões seqüenciais aparecem nos mais variados domínios de aplicações. Sua mineração pode ser utilizada para descobrir a evolução de sintomas apresentados em pacientes, para descobrir a evolução de compras realizadas por clientes, na análise de seqüências de DNA, na descoberta de caminhos (seqüências de páginas acessadas) na Web freqüentemente percorridos pelos usuários, etc.

A mineração de padrões seqüenciais constitui a tarefa de mineração mais importante no contexto desta dissertação. No capítulo 3, estudaremos o problema em detalhes e apresentaremos alguns dos principais algoritmos e técnicas utilizadas no processo de descoberta de seqüências em grandes bancos de dados. Os padrões seqüenciais, assim como o novo padrão sendo proposto nesta dissertação situam-se dentro de uma área de pesquisa mais geral, que considera dados contendo informações temporais. Dizemos respeito à Mineração de Dados Temporais. O assunto é abordado no próximo tópico.

2.3 Mineração de Dados Temporais

A Mineração de Dados Temporais é uma área de pesquisa que engloba uma variedade de tarefas de mineração, que de uma forma geral, objetivam encontrar padrões e regularidades em conjuntos de dados contendo informações temporais. A seguir, descrevemos alguns problemas de mineração de dados temporais.

2.3.1 Regras de Associação Cíclicas

A descoberta de *regras de associação* interessantes em banco de dados de transações constitui uma importante tarefa em mineração de dados [28]. De uma forma geral, a mineração dessas regras objetiva descobrir dependências significativas entre conjuntos de itens. Por exemplo, em um banco de dados mantido por um supermercado, uma regra de associação pode ser da forma "cerveja → batatas (suporte: 4%, confiança: 67%)", o que significa que 4% de todas as transações no banco de dados contêm os itens cerveja e batatas, e 67% das transações que possuem o item "cerveja" também possuem o item "batatas".

Uma *regra de associação cíclica* é aquela que ocorre em intervalos de tempo regulares, isto é, as transações que contêm a regra ocorrem periodicamente. Dessa forma, uma regra cíclica não precisa estar presente no banco de dados de transações inteiro, podendo aparecer somente em transações realizadas em intervalos de tempo periódicos. Por exemplo, a regra "café → pão" pode não ser considerada interessante quando analisada sobre todo o banco de dados de transações. Entretanto, se focamos nas transações ocorrendo em faixas específicas de horário, podemos descobrir que café e pão tendem a aparecer juntos, com grande freqüência, durante o intervalo de tempo entre as 7:00h e 9:00h da manhã, ou seja: "café → pão entre as 7:00h e 9:00h, diariamente".

2.3.2 Episódios Freqüentes em Seqüências

Em [5], os autores apresentam o problema de encontrar *episódios* freqüentes em uma longa seqüência de eventos. Um episódio é definido como um conjunto de eventos ocorrendo em uma ordem parcialmente definida dentro de um dado intervalo de tempo. Para serem considerados interessantes, os eventos de um episódio devem ocorrer suficientemente próximos no tempo. O usuário define o grau de proximidade fornecendo a largura da *janela de tempo* dentro da qual o episódio deve ocorrer. Além da largura da janela, o usuário especifica também o quanto um episódio precisa ocorrer para ser considerado freqüente. Os episódios procurados em uma seqüência de eventos e podem ser seriais ou paralelos. Um *episódio serial* ocorre na seqüência e somente se todos os seus eventos ocorrem em e

na mesma ordem em que estão dispostos no episódio. Um *episódio paralelo* é considerado estar dentro de uma seqüência e se todos os eventos do episódio aparecem dentro de uma *janela de tempo* em e sem nenhuma restrição com relação a ordem de ocorrência dos eventos.

Como em GSP [2] (o algoritmo GSP será descrito em detalhes no capítulo 3), o algoritmo proposto em [5] obtém os episódios candidatos de tamanho $k+1$ a partir de pares de episódios freqüentes de tamanho k que se sobrepõem em $k-1$ eventos. Em seguida, esses episódios são testados para determinar se eles são episódios seriais ou paralelos.

2.3.3 Séries Temporais

Uma seqüência composta por uma série de valores reais medidos em vários pontos do tempo é comumente chamada de *série temporal*. Uma característica verificada que distingui os dados de uma série temporal de outros tipos de dados é que, em geral, os valores de uma série em diferentes instantes do tempo são correlacionados [29]. Por exemplo, séries temporais poderiam ser usadas para traçar a variação da temperatura durante uma reação explosiva [4]. Neste caso, a temperatura registrada em qualquer instante está fundamentalmente relacionada com o número de segundos transcorridos desde o inicio da reação.

Uma séria temporal muitas vezes precisa ser representada em uma forma mais adequada à sua manipulação. Uma forma de representar e modelar séries temporais é realizar uma transformação da série original (com valores reais) para uma seqüência discretizada, desta vez composta por símbolos de um alfabeto. O primeiro passo nesse processo de representação é definir o alfabeto de símbolos e então mapear a série inicial para uma seqüência de símbolos. O mapeamento é feito considerando as transições de um instante para o seguinte e associando um símbolo do alfabeto para cada transição [30].

2.4 Mineração de *Itemsets* Freqüentes

A mineração de conjuntos de itens ou *itemsets* freqüentes é uma tarefa normalmente executada sobre bancos de dados de transações de clientes como tarefa preliminar na obtenção de padrões de regras de associação. Na literatura, são encontrados diversos algoritmos para descoberta de *itemsets* freqüentes: *AIS* [17], *Apriori*, *AprioriTid*, *AprioriHybrid* [24], *FPGrowth* [31], *Max-Miner* [32], entre outros. Nesta seção, descreveremos brevemente o clássico e pioneiro algoritmo *Apriori* proposto em [24], que é fundamental para a boa compreensão do restante da dissertação.

2.4.1 A Propriedade *Apriori*

Seja $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ um conjunto de itens e $\mathcal{D}$ um banco de dados de transações, onde cada transação possui um identificador único (IdT) e é composta por um sub-conjunto de itens de $\mathcal{I}$ (um *itemset*). Um *itemset* é denotado por $(x_1, x_2, ..., x_k)$, onde cada x_i, para i $\in$ {1,...,k}, é um item. Um *itemset* com k itens é denominado de *k-itemset* (também o referenciamos por *itemset* de comprimento k). O *suporte* de um *itemset* I com relação a um banco de dados $\mathcal{D}$, denotado por *sup(I)*, corresponde ao número de transações em $\mathcal{D}$ que contém I. Um *itemset* é dito *freqüente* em $\mathcal{D}$ se seu suporte é maior ou igual ao valor do *suporte mínimo* (sup_min) especificado pelo usuário.

Com o objetivo de minerar padrões freqüentes de forma eficiente, uma propriedade de *antimonotonia* para *itemsets* freqüentes, denominada *propriedade Apriori*, foi identificada em [24].

Propriedade *Apriori*. *Nenhum super-conjunto de um itemset não freqüente pode ser freqüente.* Ou em outras palavras: *todo subconjunto de um itemset freqüente também é freqüente.*

Como será notado no próximo capítulo, a propriedade *Apriori* constitui uma base para o desenvolvimento de vários algoritmos de mineração posteriormente propostos. Na seção a seguir, descreveremos o *algoritmo Apriori*, proposto em [24] a partir da propriedade enunciada acima.

2.4.2 O Algoritmo *Apriori*

Apriori é um algoritmo iterativo, proposto em [24] para minerar *itemsets* freqüentes. A cada iteração o algoritmo produz *itemsets* candidatos de comprimento $k+1$ a partir de *itemsets* freqüentes de comprimento k, executando uma operação de junção seguida de uma poda de candidatos. O algoritmo começa percorrendo todas as transações no banco de dados e calculando os itens (1-*itemsets*) freqüentes. A partir destes, o conjunto dos 2-*itemsets candidatos* potencialmente freqüentes é obtido. O banco de dados é novamente percorrido para obter seus suportes. Os 2-*itemsets* freqüentes serão agora utilizados no próximo passo, na obtenção dos 3-*itemsets* candidatos. Este processo se repete até que todos os *itemsets* freqüentes sejam obtidos. Nesta seção, denotamos o conjunto dos k-*itemsets* freqüentes e candidatos por L_k e C_k respectivamente.

O algoritmo é executado iterativamente em 3 passos principais:

1. **Geração.** Gera *itemsets* candidatos de comprimento k a partir de $(k-1)$-*itemsets* freqüentes, através de uma *junção* de L_{k-1} com L_{k-1}. Neste processo, dois *itemsets* de L_{k-1} são associados quando seus $(k-2)$ itens são iguais. Por exemplo, se L_2 = {(A,B), (A,C), (A,D), (B,C), (B,D)}, então C_3 = {(A,B,C), (A,B,D), (A,C,D), (B,C,D)}. Neste caso, uma junção entre os *itemsets* (A,B) e (A,C) produz o *itemset* (A,B,C), uma junção entre (A,B) e (A,D) produz (A,B,D), e assim por diante.

2. **Poda.** São eliminados aqueles candidatos que possuem um subconjunto não freqüente. No exemplo anterior, o 3-*itemset* candidato (B,C,D) é podado, pois seu subconjunto (C,D) não é freqüente ((C,D) não está em L_2). O novo conjunto de candidatos é C_3 = {(A,B,C), (A,B,D), (A,C,D)}.

3. **Cálculo do Suporte.** Percorre todas as transações do banco de dados para obter a freqüência em que os candidatos aparecem. Os candidatos não freqüentes são eliminados.

O algoritmo *Apriori* é dado na figura 2.2. Para maiores detalhes, recomendamos ao leitor consultar [24].

L_1 = { 1-*itemsets* freqüentes };
Para($k = 2$; $L_{k-1} \neq \emptyset$; $k++$)
  C_k = conjunto dos k-*itemsets* candidatos obtidos de L_{k-1};
  Elimine os *itemsets* c em C_k tais que $\exists\ c' \subset c$ e $c' \notin L_{k-1}$;
  Para cada transação $t \in \mathcal{D}$
    Para cada k-subconjunto s de t
      Se ($\exists c \in C_k$ tal que $c = s$) c.cont++;
  L_k = { c $\in C_k$ | c.cont $\geq$ *sup_min* };
Conjunto de todos *itemsets* frequentes = $\bigcup_k L_k$

Figura 2.2: O Algoritmo *Apriori*

Capítulo 3

Mineração de Seqüências

Neste capítulo vamos rever alguns dos principais trabalhos relacionados à mineração de padrões seqüenciais. Começamos na seção 3.1 descrevendo o problema de mineração de padrões seqüenciais (simples/proposicionais), assim como alguns importantes algoritmos para mineração desses padrões. Na seção 3.2 abordamos o problema de mineração de seqüências considerando restrições especificadas pelo usuário e concluímos o capítulo na seção 3.3 expondo de maneira sucinta os padrões seqüenciais multi-dimensionais, que são uma generalização dos padrões seqüenciais simples.

3.1 Mineração de Padrões Seqüenciais Simples

A mineração de padrões seqüenciais é um importante problema na área de mineração de dados e possui amplas aplicações. Tal mineração tem sido bastante utilizada não somente no contexto de mineração de dados temporais (como por exemplo na análise de padrões em compras de clientes), mas também em outras áreas, tais como na descoberta de caminhos de páginas da Web freqüentemente percorridos, na análise de seqüências de DNA, etc. Um exemplo típico de aplicação da mineração de seqüências é descrito a seguir.

Imagine que o gerente de uma loja esteja interessado em descobrir a evolução das compras de seus clientes ao longo do tempo. Ele quer saber quais produtos são comumente vendidos para um mesmo cliente em compras consecutivas. Ora, se o gerente descobre que os produtos p_1, p_2 e p_3 são freqüentemente vendidos para seus clientes, nesta ordem, ele poderá, por exemplo, lançar uma campanha publicitária exclusiva do produto p_3 dirigida àqueles clientes que já compraram os produtos p_1 e p_2, sabendo que existirá maior

probabilidate desse produto ser vendido para tais clientes.

Na próxima seção abordaremos formalmente o problema de mineração de padrões seqüenciais simples.

3.1.1 O Problema de Mineração

O problema de mineração de padrões seqüenciais foi introduzido por Agrawal e Srikant em [1]. Nesse trabalho, os autores propõem, entre os outros, o algoritmo *AprioriAll* para mineração de padrões seqüenciais em um banco de dados de transações de clientes.

Neste tópico, apresentaremos primeiramente algumas definições e conceitos essenciais para o contexto da mineração de seqüências para em seguida descrever o problema de mineração em si.

Definição 3.1.1 (Seqüência). Seja I um conjunto de itens. Uma **seqüência** s é uma lista ordenada de *itemsets* e é denotada por $s = < s_1 s_2 ... s_n >$, onde cada s_i é um *itemset*, ou seja, $s_i \subseteq I$. O **comprimento** de uma seqüência s, *comp(s)*, corresponde ao número total de itens dessa seqüência, isto é, $comp(s) = \sum_{i=1}^{n} |s_i|$, onde $|s_i|$ denota o número de itens do i-ésimo *itemset* de s. Uma seqüência de comprimento k é referenciada como uma k-seqüência.

Exemplo 3.1.1 Suponha um conjunto de itens $I = \{a, b, c, d, e\}$. $s =< (a, b)(a)(b, c, d) >$ é um exemplo de uma seqüência de comprimento 6 possuindo 3 *itemsets*. Um *itemset* pode representar, por exemplo, um conjunto de produtos comprados por um cliente em uma única transação. Repare que cada ocorrência de um item na seqüência é contabilizada (veja o item a).

Definição 3.1.2 (Sub-seqüência). Dizemos que uma seqüência $\alpha =< a_1, a_2, ..., a_n >$ é uma **sub-seqüência** de outra seqüência $\beta =< b_1, b_2, ..., b_m >$, $m \geq n$, denotado por $\alpha \sqsubseteq \beta$, se existem inteiros $j_1 < j_2 < ... < j_n$ tais que $a_1 \subseteq b_{j_1}$, $a_2 \subseteq b_{j_2}$, ..., $a_n \subseteq b_{j_n}$. Também chamamos β de uma **super-seqüência** de α e dizemos que β *contém* α ou que α *está contida* em β.

Exemplo 3.1.2 As seqüências <(b)(a)(b,d)> e <(b)(c)> são sub-seqüências de <(a,b)(a)(b,c,d)>, mas <(b)(b)(c)> não é uma sub-seqüência de <(a,b)(a)(b,c,d)>.

Definição 3.1.3 (Seqüência maximal). Dado um conjunto de seqüências C, dizemos que a seqüência $s \in C$ é **maximal** se não há nenhuma seqüência em $C - \{s\}$ contendo s.

Exemplo 3.1.3 Considere o conjunto de seqüências A = {<(a)(b,c)(d)>, <(a,e)(b)>, <(a)(c)(d)>, <(a)(b)>}. A seqüência <(a,e)(b)> é maximal, pois nenhuma outra no conjunto a contém. Por outro lado, a seqüência <(a)(c)(d)> não é maximal, pois está contida na seqüência <(a)(b,c)(d)>.

Definição 3.1.4 (Banco de dados de seqüências). Um **banco de dados de seqüências** D é um conjunto de tuplas $<ids, s>$, onde s é uma seqüência e ids o identificador da seqüência s. Dizemos que uma tupla $<ids, s>$ em um banco de dados de seqüências D *contém* uma seqüência α, se α é sub-seqüência de s, ou seja, $\alpha \sqsubseteq s$.

Exemplo 3.1.4 Na figura 3.1 temos um banco de dados de seqüências com quatro tuplas.

IdSeqs	**Seqüências**
1	<(a,b)(c)(b)(d)>
2	<(c,d)(a)(a,b)>
3	<(d)(a,b)>
4	<(b,c,d)(c)(a,f)>

Figura 3.1: Exemplo de um banco de dados de seqüências

Definição 3.1.5 (Suporte). O **suporte** de uma seqüência α em um banco de dados de seqüências D, $sup(\alpha)$, corresponde à porcentagem de tuplas em D que contém α, isto é, $sup(\alpha) = \frac{|\{s \mid \exists id, <id,s> \in D \ e \ \alpha \sqsubseteq s\}|}{|D|}$.

Definição 3.1.6 (Seqüência freqüente). Dado um nível mínimo de suporte, sup_min, uma seqüência α é dita **freqüente** em um banco de dados de seqüências D se a porcentagem de tuplas de D contendo α é maior ou igual a sup_min. Uma seqüência freqüente também é chamada de **padrão seqüencial**.

Exemplo 3.1.5 Considere o banco de dados da figura 3.1. A seqüência <(b)(c)> está contida nas seqüências 1 e 4 desse banco de dados. Logo, o suporte de <(b)(c)> é 2. Se consideramos um suporte mínimo igual a 1 (25%) então <(b)(c)> é tida como uma seqüência freqüente desse banco de dados, visto que duas seqüências contêm <(b)(c)>.

Apresentamos anteriormente os principais conceitos envolvidos na mineração de padrões seqüenciais. Agora, faremos uma definição geral desse problema de mineração, como segue:

- **Dado:** um banco de dados de seqüências D e um nível mínimo de suporte *sup_ min*.
- **Encontrar:** todas as seqüências *freqüentes* com relação a D e *sup_ min*.

Muitos estudos têm contribuído para a mineração eficiente de padrões seqüenciais ou outros padrões freqüentes em dados temporais [24, 1, 2, 5, 33, 8, 11]. Em [2] Agrawal e Srikant generalizam suas definições de padrões seqüenciais introduzidas em [1] para incluir restrições de tempo e taxonomias, além de apresentarem um algoritmo mais eficiente para mineração desses padrões. Em [5], os autores apresentam o problema de mineração de episódios freqüentes em uma seqüência de eventos (veja capítulo 2, seção 2.3.2). Em [33] propõe-se incorporar expressões regulares fornecidas pelo usuário no processo de mineração afim de restringir os padrões seqüenciais gerados.

Os algoritmos já propostos para solução do problema formulado anteriormente são vários. Podemos citar *AprioriAll* [1], GSP [2], PSP [34], SPADE [13], *FreeSpan* [11], *PrefixSpan* [12], *DepthProject* [3], entre outros. É importante enfatizar que tais algoritmos mineram **todos** os padrões seqüenciais no banco de dados. Em [14], no entanto, é proposto o algoritmo *CloSpan*, projetado para minerar apenas as seqüências freqüentes maximais. Neste caso, o algoritmo minera somente aquelas seqüências não contidas em outras de tamanho maior.

Nas seções seguintes, abordaremos as idéias principais dos algoritmos GSP, SPADE e *FreeSpan*.

3.1.2 O Algoritmo GSP

O algoritmo GSP foi proposto em [2] para a mineração de padrões seqüenciais (simples) e também se baseia na propriedade *Apriori* proposta em [24] (seção 2.4.1).

GSP é um algoritmo iterativo que encontra, a cada iteração, o conjunto dos padrões seqüenciais freqüentes de comprimento k, denotado por L_k. Como descrito em [2], cada iteração de GSP é realizada em duas fases: a fase da *geração de candidatos* e a fase do *cálculo do suporte*. A fase de geração de candidatos, por sua vez, é sub-dividida na etapa da *junção* e na etapa da *poda*. Na primeira iteração, GSP encontra o conjunto de todos os itens freqüentes no banco de dados (1-seqüências). Tal conjunto é utilizado na segunda iteração, para o cálculo das 2-seqüências candidatas. Uma 2-seqüência candidata pode conter um único *itemset* com 2 itens freqüentes, ou dois *itemsets*, cada um contendo um item freqüente. Por exemplo, se os itens 1 e 2 são freqüentes, na segunda iteração serão geradas as 2-seqüências candidatas <(1),(2)>, <(2),(1)> e <(1,2)>. Em seguida, o banco de dados é percorrido e o suporte das 2-seqüências candidatas é calculado. GSP elimina as seqüências candidatas com suporte inferior ao suporte mínimo. A partir da terceira iteração, o conjunto das k-seqüências candidatas é gerado através da combinação de (k-1)-seqüências freqüentes, em seguida as k-seqüências candidatas que não podem ser freqüentes são *podadas* e então o banco de dados é percorrido para cálculo do suporte das seqüências candidatas restantes. O algoritmo é dado na figura 3.2.

A seguir, descrevemos em mais detalhes como é realizada cada uma das fases do algoritmo GSP:

1. **Geração de Candidatos:** realizada em duas etapas:

   - **Etapa da Junção:** dado um par de (k-1)-seqüências (s_1,s_2), onde descartando o primeiro item de s_1 e o último item de s_2 obtemos seqüências idênticas, é criada uma nova k-seqüência candidata acrescentando-se o último item de s_2 em s_1. O item adicionado estará em um *itemset* separado, caso assim ele estava em s_2, ou fará parte do último *itemset* de s_1, caso contrário. Por exemplo, dadas as 3-seqüências <(1,3),(2)>, <(3),(2),(4)> e <(2,5),(4)>, podemos realizar a junção da primeira seqüência com a segunda, pois obtemos seqüências iguais

Entrada: Um banco de dados de seqüencias **D** e um nível mínimo de suporte α
Saída: O conjunto completo de padrões seqüenciais freqüentes
Início
$L_1 = \{$ 1-seqüências freqüentes $\}$
$L = F_1$ // conjunto de todas as seqüencias freqüentes
Para$(k = 2;\ L_{k-1} \neq \emptyset;\ k++)$
$C_k =$ conjunto das k-seqüencias candidatas // combina seqüencias de L_{k-1}
$C_k = C_k$ - $\{\sigma \in C_k \mid \exists \tau \sqsubseteq \sigma, comp(\tau) = k-1\ e\ \tau \notin L_{k-1}\ \}$ // poda
Para cada seqüência de entrada s do banco de dados D
incremente o contador de todas as seqüencias $c \in C_k$ contidas em s.
$L_k = \{$ c $\in C_k \mid$ c.suporte $\geq \alpha\ \}$
$L = L \cup L_k$
retorne L
Fim

Figura 3.2: O Algoritmo GSP

eliminando-se o item 1 de <(1,3),(2)> e o item 4 de <(3),(2),(4)>. A nova seqüência obtida é a 4-seqüência candidata <(1,3),(2),(4)>.

- **Etapa da Poda:** nesta etapa são eliminadas as k-seqüências candidatas que possuem pelo menos uma (k-1)-subseqüência não freqüente, ou seja, uma (k-1)-subseqüência que não se encontra no conjunto L_{k-1} de seqüências freqüentes gerado na iteração anterior.

2. **Cálculo do Suporte:** dado um conjunto de seqüências candidatas C_k, o banco de dados é percorrido uma vez e o suporte de cada k-seqüência candidata é calculado. As k-seqüências candidatas cujos suportes não atingem o suporte mínimo α são eliminadas e o conjunto L_k de seqüências freqüentes é obtido. Para cada seqüência s do banco de dados, o algoritmo verifica quais k-seqüências candidatas são subseqüências de s, incrementando o contador de suporte dessas seqüências candidatas. Para reduzir o número de seqüências candidatas que são testadas para cada seqüência s do banco de dados, GSP armazena o conjuntos das seqüências candidatas em uma *árvore hash* [35]. Um nó folha dessa árvore armazena uma lista de seqüências candidatas enquanto um nó interno mantém uma *tabela hash* [35] com ponteiros

para outros nós do nível seguinte. Maiores detalhes de como a árvore é utilizada nesta fase do algoritmo podem ser encontrados em [2].

3.1.3 O Algoritmo SPADE

Nesta seção, descrevemos brevemente as principais idéias do algoritmo SPADE para mineração de padrões seqüenciais. Recomendamos ao leitor consultar [13], caso deseje obter mais detalhes.

O algoritmo GSP descrito na seção anterior, assim como o algoritmo *AprioriAll* proposto em [1], efetua a mineração considerando o banco de dados em seu formato *horizontal*, como o banco ilustrado na figura 3.3. Um banco de dados de transações no formato horizontal contém um campo relativo ao cliente (idC), outro campo informando o instante da transação (idT) e um terceiro correspondendo ao conjunto de itens comprados na transação em questão. Por outro lado, o algoritmo SPADE faz uso do banco de dados em um formato *vertical*, em que é associado a cada item uma lista das transações do banco em que o item ocorre. Tal lista é denominada de *id-list*. A figura 3.4 ilustra, no formato vertical, o banco de dados da figura 3.3. Repare que esse banco é composto por 3 *id-lists*, sendo uma para cada item.

idC	**idT**	**Itens**
1	10	D
1	15	A B
1	20	A B
1	25	A D
2	15	A B
3	10	A B
4	10	D
4	20	B
4	25	A

Figura 3.3: Banco de dados horizontal

A	
IdC	**IdT**
1	15
1	20
1	25
2	15
3	10
4	25

B	
IdC	**IdT**
1	15
1	20
2	15
3	10
4	20

D	
IdC	**IdT**
1	10
1	25
4	10

Figura 3.4: Banco de dados vertical (*id-lists*)

Diferente do algoritmo GSP, que percorre o banco de dados a cada iteração, SPADE utiliza técnicas de busca em reticulado e realiza operações de junções em *id-lists* para encontrar os padrões seqüenciais freqüentes. O algoritmo inicialmente encontra o conjunto

dos itens freqüentes (as 1-seqüências) percorrendo o banco de dados no formato vertical. Para cada item existente, sua *id-list* é percorrida e seu contador de suporte é incrementado para cada novo IdC encontrado.

Para o cálculo das 2-seqüências freqüentes, a parte do banco de dados no formato vertical correspondente às *id-lists* dos itens freqüentes é convertida novamente para um outro banco de dados horizontal. A figura 3.5 ilustra o banco de dados vertical da figura 3.4 convertido para o novo formato horizontal. Tal banco é obtido como segue. Para cada item freqüente i, consideramos cada par (idC_j, idT_j) de sua *id-list* e acrescentamos o par (i, idT_j) na linha do banco horizontal correspondente a idC_j.

IdC	**Pares (Item, IdT)**
1	(A,15)(A,20)(A,25)(B,15)(B,20)(D,10)(D,25)
2	(A,15)(B,15)
3	(A,10)(B,10)
4	(A,25)(B,20)(D,10)

Figura 3.5: Retorno para banco de dados horizontal

O conjunto das 2-seqüências freqüentes é gerado a partir desse banco de dados horizontal, como descrito a seguir. Cria-se inicialmente uma matriz $n \times n$ (n corresponde ao número de itens freqüentes) para armazenar na posição i, j $(i \leq n$ e $j \leq n)$ o contador de suporte para a 2-seqüência $< (i), (j) >$. Para cada idC_p de IdC, forma-se uma lista de todas as 2-seqüências possíveis de serem obtidas dos pares relativos a idC_p e a cada 2-seqüência obtida, atualiza-se seu contador de suporte na matriz de contadores criada.

Todas as demais k-seqüências ($k > 2$) são obtidas através de junções temporais sobre as (k-1)-seqüências já encontradas. Dependendo dos pares de (k-1)-seqüências utilizados na junção, há três possibilidades para uma k-seqüência candidata resultante. Por exemplo, se fazemos a junção das seqüências <{a},{b},{b}> e <{a},{b},{c}>, podemos obter as seqüências <{a},{b},{b,c}>, <{a},{b},{b},{c}> ou <{a},{b},{c},{b}>. Diferente de GSP ou *AprioriALL*, o algoritmo SPADE realiza o cálculo do suporte de uma k-seqüência candidata efetuando uma junção das *id-lists* relativas ao par de (k-1)-seqüências utilizadas na obtenção da k-seqüência candidata. De posse da *id-list* para a nova k-seqüência candidata, o suporte da mesma é facilmente calculado contando-se os idC's

distintos dessa *id-list*.

3.1.4 O Algoritmo *FreeSpan*

FreeSpan difere fundamentalmente dos algoritmos baseados na técnica de geração e validação de *Apriori*, com repetidas varridas no banco de dados. *FreeSpan* projeta recursivamente o banco de dados de seqüências em um conjunto de bancos de dados menores e então minera cada banco de dados projetado para encontrar os padrões freqüentes. O algoritmo é explicado a seguir.

Dada uma seqüência s, denominamos o conjunto de itens que nela aparecem de *padrão transação* de s, denotado por $\Phi(s)$. Por exemplo, o padrão transação de <(a,b)(a,c)(b,d)> é {a,b,c,d}.

Definição 3.1.7 A função $\Psi(\omega, s)$ é definida como a função que retorna o conjunto de sub-seqüências de s possuindo o padrão transação ω, ou seja, $\Psi(\omega, s) = \{s' \sqsubseteq s |\ \Phi(s') = \omega\}$. Por exemplo, $\Psi(\{a, b\}, < (a, b)(b)(c) >) = \{$<(a,b)>, <(a)(b)>$\}$.

A figura 3.6 ilustra um banco de dados de seqüências e associado a cada seqüência temos seu padrão transação, formando o respectivo banco de dados de padrões transação. *FreeSpan* é baseado na seguinte relação entre esses dois bancos de dados:

Uma seqüência não pode ser freqüente se seu padrão transação não é freqüente no respectivo banco de dados de padrões transação.

A relação acima fornece uma heurística para diminuir o espaço de busca na mineração das seqüências: se um padrão transação não é freqüente, não é necessário verificar seus padrões seqüenciais correspondentes.

idC	**Seqüência**	**Padrão Transação**
1	(3,4)(1,2,3)(1,2,6)(1,3,4,6)	{1,2,3,4,6}
2	(1,2,6)(5)	{1,2,5,6}
3	(1,2,6)	{1,2,6}
4	(4,7,8)(2,6)(1,7,8)	{1,2,4,6,7,8}

Figura 3.6: Banco de dados de seqüências e padrões transação

Dado um banco de dados D e um suporte mínimo sup_min, *FreeSpan* inicialmente varre D, calcula o suporte de cada item e encontra o conjunto dos itens freqüentes, que são listados em ordem decrescente do suporte. Essa lista de itens assim organizada é denominada de *f-list(D)*.

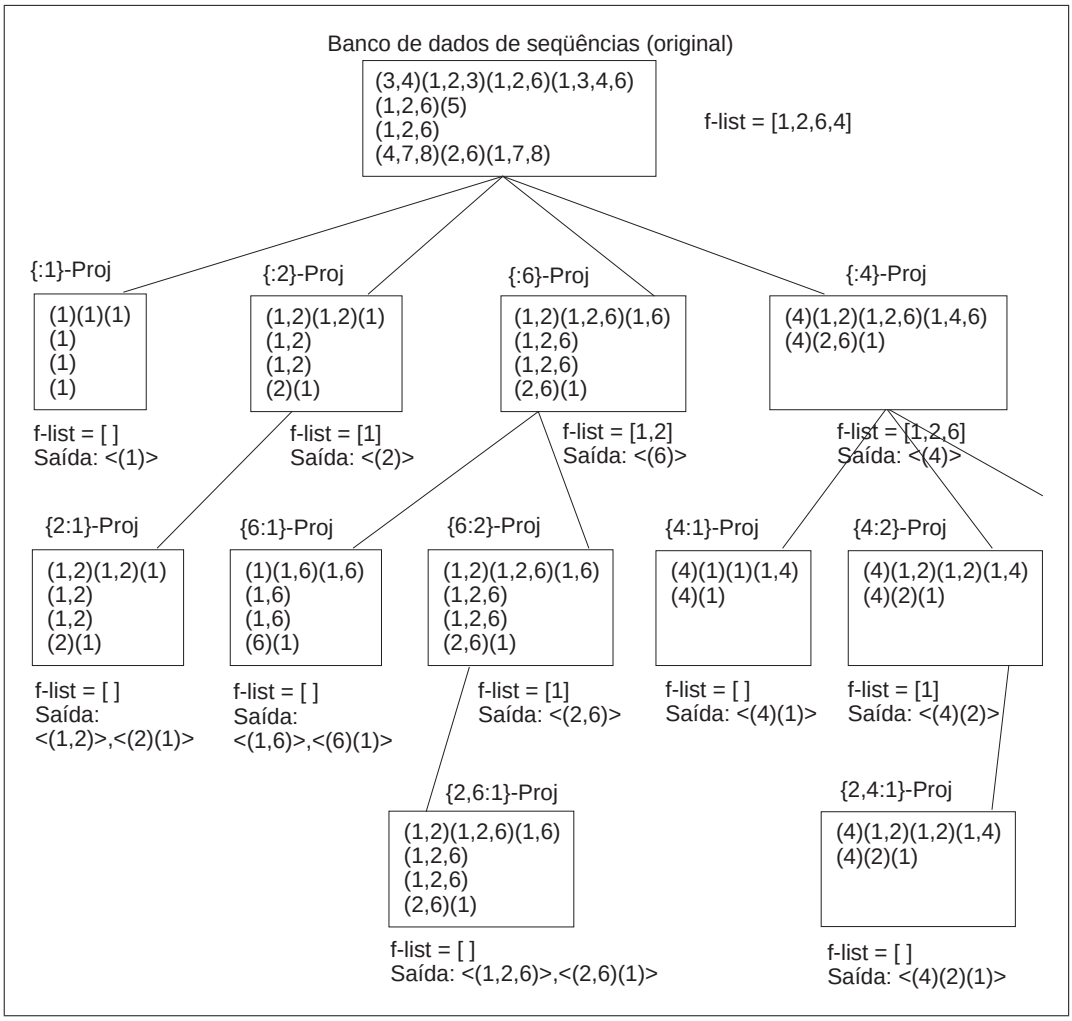


Figura 3.7: Projeções de bancos de dados em FreeSpan

Definição 3.1.8 Seja D um banco de dados de seqüências. $\Pi(\omega,\ j,$ *f-list(D)*$,\ s)$ é definida como a função que retorna uma *sub-seqüência projetada* $s' \sqsubseteq s$, tal que seu padrão transação $\Phi(s')$ contenha $\omega \cup \{j\}$ e somente os itens de *f-list(D)* aparecendo antes de j.

Por exemplo, considere D sendo o banco de dados de seqüências da figura 3.6 e

sup_ min = 50%. Neste caso, *f-list(D)* = [1,2,6,4], $\Pi(\emptyset,\ 2,\ [1,2,6,4],\ <(1,2,6)(5)>)$ = <(1,2)> e $\Pi(\{4\},\ 1,\ [1,2,6,4],\ <(4)(2,6)(1)>)$ = <(4)(1)>.

Definição 3.1.9 Seja D um banco de dados de seqüências. O banco de dados $\{\omega : j\}$-projetado é obtido aplicando-se a função $\Pi(\omega,\ j,$ *f-list(D)*, $\ s)$ a cada seqüência s de D. O banco de dados $\{\omega : j\}$-projetado é referenciado como $D|_\rho$, onde $\rho = \omega \cup j$.

A figura 3.7 ilustra como *FreeSpan* projeta recursivamente o banco de dados da figura 3.6 em bancos de dados menores, em seu processo de mineração. O suporte mínimo considerado é de 50%. De acordo com a lista de itens freqüentes obtida para tal banco de dados, [1,2,6,4], *FreeSpan* projeta o mesmo em 4 bancos de dados menores: o banco de dados {:1}-projetado, o {:2}-projetado, o {:6}-projetado e o banco de dados {:4}-projetado, ilustrados no segundo nível da figura 3.7.

FreeSpan pode ser descrito em quatro principais passos executados sobre cada banco de dados projetado $D|\rho$ recursivamente. Inicialmente $\rho = \{\ \}$ e nesse caso o banco de dados projetado $D|\rho$ representa o banco de dados original. Tais passos são:

1. Percorra o banco de dados $D|\rho$ uma vez e encontre os itens freqüentes que não estão em ρ, isto é, calcule *f-list(D|ρ)* e em seguida retire desta lista os itens pertencentes a ρ. Chamamos essa lista obtida após eliminar os itens que pertencem a ρ de *f-list* apenas. Por exemplo, para o banco de dados original exibido no topo da figura 3.7, a lista desses itens freqüentes é *f-list* = [1,2,6,4] (ρ é vazio). Para o banco de dados {:4}-projetado do nível abaixo, *f-list* = [1,2,6]. Note que o item 4 não faz parte dessa lista (apesar de freqüente), pois 4 está em ρ.

   Ao mesmo tempo, calcule o suporte de todas as seqüências com o padrão transação ρ. (para cada seqüência s do banco de dados em questão, a função $\Psi(\rho, s)$ é utilizada para listar todas as sub-seqüências de s com o padrão transação ρ).

2. Mostre todas as seqüências freqüentes que possuem o padrão transação ρ. Considere, por exemplo, o "ramo" mais a direita da estrutura ilustrada na figura 3.7. Para o banco de dados {:4}-projetado, a seqüência <(4)> é a única freqüente contendo o padrão transação {4}. Para o banco de dados {4:2}-projetado, a única seqüência contendo o padrão transação {4,2} freqüente é <(4)(2)>.

3. Se foi mostrada alguma seqüência freqüente e a *f-list* não é vazia, então para cada item j da *f-list* crie um banco de dados $\{\rho{:}j\}$-projetado. Percorra o banco de dados projetado $D|\rho$ novamente e preencha cada banco de dados $\{\rho{:}j\}$-projetado.

4. Recursivamente minere cada novo banco de dados $\{\rho{:}j\}$-projetado.

Um dos maiores custos do algoritmo *FreeSpan* está associado à função $\Psi(\rho, s)$, que é utilizada para encontrar todas as sub-seqüências que possuem um determinado padrão transação. Outro custo a considerar no algoritmo está associado ao espaço de armazenamento e operações de entrada e saída vinculados aos bancos de dados projetados. O espaço necessário para armazenar os bancos de dados projetados pode ser maior do que o próprio espaço utilizado pelo banco de dados original.

3.2 Mineração de Padrões Seqüenciais com Restrições

Os algoritmos para mineração de padrões seqüenciais descritos nas seções anteriores foram desenvolvidos para realizar a mineração desses padrões considerando apenas a restrição do *suporte mínimo* fornecida pelo usuário, que diz respeito à freqüência mínima dos padrões no banco de dados. Tais algoritmos não fornecem ao usuário mecanismos que lhe possibilite restringir os padrões gerados de acordo com seu interesse. O usuário pode estar interessado, por exemplo, apenas naqueles padrões seqüênciais que envolvam determinados itens ou naquelas seqüências que começam com determinado produto.

Nesta seção, mostramos algumas das restrições que podem ser consideradas no processo de mineração. Também apresentamos os algoritmos da família SPIRIT [33], que realizam a mineração de padrões seqüenciais considerando restrições expressas por expressões regulares. Estes algoritmos são apresentados com algum detalhe, pois constituem peça importante no algoritmo *MSP-Miner*, que será apresentado no capítulo 9 para mineração de seqüências múltiplas

3.2.1 Tipos de Restrições

De uma maneira geral, as restrições consideradas no processo de mineração de seqüências podem ser classificadas em *restrições de geração* e *restrições de validação*. As restrições

de geração são incorporadas na fase de geração de candidatos dos algoritmos de mineração visando diminuir o espaço de busca dos padrões. Já as restrições de validação só podem ser incorporadas na fase de cálculo do suporte desses algoritmos. Neste caso, os padrões candidatos são gerados normalmente e aqueles que não satisfazem a restrição são eliminados na fase de validação do algoritmo.

São exemplos de restrições de validação as restrições de tempo *min-gap* e *max-gap* introduzidas em [2]. Considerando tais restrições, um padrão seqüencial do tipo $<s_1,s_2>$, por exemplo, é considerado interessante apenas se as transações contendo s_1 e as transações contendo s_2 são realizadas em um intervalo de tempo mínimo *min-gap* e esse intervalo não ultrapassa o limiar *max-gap*. Um exemplo de uma restrição de geração é uma restrição expressa por uma expressão regular. Tal restrição é introduzida e incorporada pelos algoritmos da família SPIRIT [33], descritos na próxima seção.

3.2.2 Os Algoritmos *SPIRIT*

Nesta seção descrevemos os algoritmos da família SPIRIT propostos em [33] para mineração de padrões seqüenciais considerando restrições de expressões regulares.

Suponha, por exemplo, que o usuário está interessado somente em padrões seqüenciais começando com o item *TV* e terminando com o item *DVD Player*. Neste caso, os padrões seqüenciais a serem gerados deverão satisfazer uma expressão regular do tipo $(TV)a^*(DVDPlayer)$, onde a^* representa uma seqüência qualquer de *itemsets*.

Formulação do Problema de Mineração

O problema de mineração de padrões seqüenciais considerando uma restrição de expressão regular $\mathcal{R}$ pode ser formulado da seguinte forma:

- **Dado:** um banco de dados de seqüências $\mathcal{D}$, um nível mínimo de suporte α e uma expressão regular $\mathcal{R}$ (ou, equivalentemente, um autômato $\mathcal{A}_\mathcal{R}$).
- **Encontrar:** todas as seqüências freqüentes em $\mathcal{D}$ que satisfazem $\mathcal{R}$.

A estrutura básica dos algoritmos para mineração de seqüências que se baseiam na técnica *Apriori*, tal como o GSP (ver seção 3.1.2), corresponde a uma fase de geração em que

seqüências candidatas de tamanho k (C_k) são obtidas de seqüências freqüentes de L_{k-1}, seguida por uma fase de poda, em que são eliminadas as seqüências de C_k possuindo uma (k-1)-seqüência não contida em L_{k-1}. Para o problema de mineração em questão, os algoritmos da família SPIRIT, semelhantemente, obtêm o conjunto de seqüências candidatas C_k a partir de L_{k-1} considerando $\mathcal{R}$, de maneira que os candidatos gerados satisfaçam a expressão regular $\mathcal{R}$. Porém, a poda aqui não pode ser efetuada simplesmente eliminando as seqüências de C_k contendo uma (k-1)-seqüência não pertencente a L_{k-1}. Note que agora o conjunto L_k é formado por seqüências **freqüentes** e que **satisfazem a expressão regular** $\mathcal{R}$ e no entanto, a condição de satisfazer $\mathcal{R}$ não é *antimonotônica* como a condição de ser freqüente. Por exemplo, a seqüência <(a)(b)(b)> satisfaz a expressão regular ab^*, mas sua sub-seqüência <(b)(b)> não a satisfaz. Entretanto, se uma seqüência candidata σ pertencente a C_k possui uma sub-seqüência $\sigma' \subset \sigma$ que **satisfaz $\mathcal{R}$ e não está em** L, onde $L = L_1 \cup L_2 \cup ... \cup L_{k-1}$, então neste caso podemos ter certeza de que σ não é freqüente e portanto pode ser podada. O conjunto P das seqüências que serão podadas em C_k pode ser definido como segue:

$$P = \{\sigma \in C_k | \ \exists \sigma' \subset \sigma, \ \sigma' \text{ satisfaz } \mathcal{R} \text{ e } \sigma' \notin L\}$$

Observando como o conjunto P é obtido, percebemos que quanto mais restritiva for a expressão $\mathcal{R}$, menor será o conjunto P, ou seja, menos seqüências serão podadas. Assim, ao mesmo tempo que a restrição $\mathcal{R}$ restringe os candidatos gerados, ela também diminui o número de candidatos que são podados na fase da poda, o que não é interessante. Os algoritmos da família SPIRIT consideram relaxamentos para a restrição $\mathcal{R}$ com o objetivo de restringir suficientemente as seqüências geradas e ao mesmo tempo permitir que um número maior de seqüências sejam podadas. Um relaxamento de uma restrição $\mathcal{R}$ corresponde a uma restrição *mais fraca* $\mathcal{R}'$ tal que toda seqüência satisfazendo $\mathcal{R}$ também satisfaz $\mathcal{R}'$.

Os quatro algoritmos SPIRIT

Os quatro algoritmos da família SPIRIT propostos em [33] são definidos de acordo com 4 relaxamentos considerados para a restrição $\mathcal{R}$. Os algoritmos são: SPIRIT(N), SPIRIT(L),

SPIRIT(V) e SPIRIT(R). O relaxamento $\mathcal{R}'$ varia para cada algoritmo e a seletividade de $\mathcal{R}'$ em SPIRIT($\mathcal{R}'$) aumenta na seguinte ordem: SPIRIT(N) $<$ SPIRIT(L) $<$ SPIRIT(V) $<$ SPIRIT(R). A seguir, descrevemos brevemente a idéia principal de cada um deles.

- **SPIRIT(N):** considera um relaxamento total de $\mathcal{R}$, onde nenhuma restrição é imposta às seqüências geradas. SPIRIT(N) corresponde a uma simples modificação de GSP [2]. O algoritmo apenas exige que todos os itens de uma seqüência candidata s de C_k apareçam na expressão regular $\mathcal{R}$.

- **SPIRIT(L):** considera um relaxamento de $\mathcal{R}$ correspondendo às seqüências *legais* com respeito a algum estado do autômato $\mathcal{A}_\mathcal{R}$ associado à expressão regular $\mathcal{R}$. Uma seqüência $< a_1, a_2, ..., a_n >$ é dita *legal* com relação a um estado q de $\mathcal{A}_\mathcal{R}$ se existe um caminho no autômato que começa no estado q e percorre a palavra $a_1, a_2, ..., a_n$.

- **SPIRIT(V):** considera uma restrição relaxada $\mathcal{R}'$ mais forte que aquela utilizada em SPIRIT(L). SPIRIT(V) considera as seqüências *válidas* com respeito a algum estado de $\mathcal{A}_\mathcal{R}$. Uma seqüência $s =< a_1, a_2, ..., a_n >$ é dita *válida* com respeito a um estado q de $\mathcal{A}_\mathcal{R}$ se existe um caminho em $\mathcal{A}_\mathcal{R}$ correspondente à palavra $a_1, a_2, ..., a_n$ que começa em q e termina em um estado final de $\mathcal{A}_\mathcal{R}$.

- **SPIRIT(R):** a expressão regular $\mathcal{R}$ é totalmente considerada no processo de mineração. Aqui, não há um relaxamento propriamente dito de $\mathcal{R}$, isto é, $\mathcal{R}' \equiv \mathcal{R}$. O algoritmo gera as seqüências que são satisfeitas pela expressão regular $\mathcal{R}$.

De uma maneira geral, os quatro algoritmos da família SPIRIT realizam a mineração em duas etapas. Na **primeira etapa**, os algoritmos encontram as seqüências freqüentes que satisfazem o relaxamento $\mathcal{R}'$ da expressão regular $\mathcal{R}$. Em uma **segunda etapa**, também tida como uma etapa de pós-processamento, os algoritmos eliminam dos conjuntos de seqüências freqüentes ($L_{k's}$) aquelas seqüências que não satisfazem a expressão regular original $\mathcal{R}$, que é fornecida pelo usuário. Repare que o algoritmo SPIRIT(N), por considerar um relaxamento total de $\mathcal{R}$ na geração de candidatos, não restringe as seqüências que são obtidas na primeira etapa e todas aquelas não satisfazendo a restrição $\mathcal{R}$ são eliminadas na segunda etapa. Por outro lado, no algoritmo SPIRIT(R) todas as seqüências

freqüentes satisfazendo $\mathcal{R}$ são obtidas logo na primeira etapa e nenhum pós-processamento é realizado na etapa 2.

Nos resultados experimentais apresentados em [33], o algoritmo SPIRIT(V) é aquele que apresenta, de uma maneira geral, a melhor performance dentre os quatro algoritmos. SPIRIT(V) é descrito em detalhes na seção seguinte. O leitor poderá consultar [33] para obter detalhes dos algoritmos SPIRIT(N), SPIRIT(L) e SPIRIT(R).

O algoritmo SPIRIT(V)

Seja $\mathcal{A}_{\mathcal{R}}$ o autômato correspondente à expressão regular $\mathcal{R}$ especificada pelo usuário e seja $\{q_0, q_1, ..., q_n\}$ o conjunto de estados do autômato $\mathcal{A}_{\mathcal{R}}$. Denotamos por $C_k(q)$ (resp. $L_k(q)$) o conjunto das k-seqüências candidatas (resp. freqüentes) e *válidas* com respeito ao estado q do autômato $\mathcal{A}_{\mathcal{R}}$.

Geração de Candidatos. Para que uma seqüência candidata $s =< s_1, s_2, ..., s_k >$ em C_k seja válida com respeito a algum estado q_x de $\mathcal{A}_{\mathcal{R}}$, é preciso que (1) seu sufixo de comprimento $k-1 < s_2, ..., s_k >$ seja freqüente e válido com respeito a algum estado q_y de $\mathcal{A}_{\mathcal{R}}$ (ou seja, o sufixo deve estar em L_{k-1}) e (2) exista uma transição $q_x \stackrel{s_1}{\rightarrow} q_y$ em $\mathcal{A}_{\mathcal{R}}$. Logo, o conjunto C_k de seqüências candidatas é gerado a partir de L_{k-1} e $\mathcal{A}_{\mathcal{R}}$ como segue. Para cada estado q de $\mathcal{A}_{\mathcal{R}}$, o conjunto das k-seqüências potencialmente freqüentes e válidas com relação a q é obtido da seguinte forma: para cada transição $q \stackrel{s_i}{\rightarrow} q'$, para cada seqüência $< s_1, ..., s_{k-1} >$ de $L_{k-1}(q')$, insira $< s_i, s_1, ..., s_{k-1} >$ no conjunto dos candidatos $C_k(q)$. O conjunto C_k é simplesmente a união desses conjuntos de candidatos para cada estado q.

Poda de Candidatos. São eliminadas de C_k as k-seqüências que não possuem nenhuma chance de serem freqüentes. Para cada seqüência s de C_k, SPIRIT(V) calcula todas as sub-seqüências maximais de s que são válidas com relação a algum estado de $\mathcal{A}_{\mathcal{R}}$ e possuem tamanho inferior a k (as sub-seqüências maximais são as maiores possíveis, não contidas em nenhuma outra sub-seqüência). O procedimento para cálculo dessas sub-seqüências é dado em [33]. Se alguma dessas sub-seqüências não aparece em L ($L = L_1 \cup L_2 \cup ... \cup L_{k-1}$)

então s é removida de C_k.

Condição de Parada. SPIRIT(V) pára quando o conjunto das k-seqüências freqüentes, L_k, for vazio.

Exemplo 3.2.1 Considere a restrição $\mathcal{R}$ dada através do autômato $\mathcal{A}_\mathcal{R}$ da figura 3.8. Suponha que o conjunto das 2-seqüências freqüentes com relação a um banco de dados D e válidas com respeito aos estados de $\mathcal{A}_\mathcal{R}$ seja $L_2 = L_2(q_0) \cup L_2(q_1) \cup L_2(q_2)$, onde $L_2(q_0) = \{< a, c >, < a, e >\}$, $L_2(q_1) = \{< b, c >\}$ e $L_2(q_2) = \{< d, c >, < d, e >\}$. A seguir, mostramos como SPIRIT(V) calcula o conjunto C_3 das 3-seqüências candidatas válidas com relação a $\mathcal{A}_\mathcal{R}$. Devemos calcular $C_3(q_0)$, $C_3(q_1)$, $C_3(q_2)$ e $C_3(q_3)$.

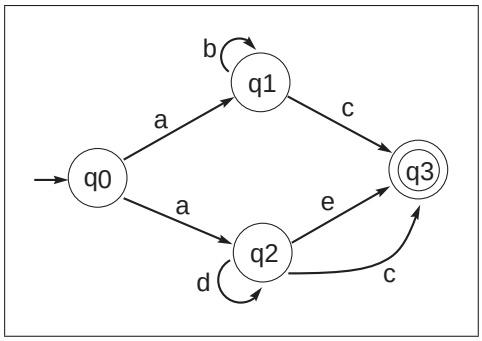


Figura 3.8: Autômato $\mathcal{A}_\mathcal{R}$ correspondente a uma restrição $\mathcal{R}$

- **Com respeito a q_0.** Transições partindo de q_0:

  1. $q_0 \xrightarrow{a} q_1$: temos apenas a seqüência $< b, c >$ em $L_2(q_1)$. Logo, é construída a 3-seqüência $< \mathbf{a}, b, c >$.
  2. $q_0 \xrightarrow{a} q_2$: temos as seqüências $< d, c >$ e $< d, e >$ em $L_2(q_2)$. Logo, são construídas as 3-seqüências $< \mathbf{a}, d, c >$ e $< \mathbf{a}, d, e >$.

  Portanto, o conjunto das 3-seqüências candidatas válidas com relação a q_0 é $C_3(q_0) = \{< a, b, c >, < a, d, c >, < a, d, e >\}$

- **Com respeito a q_1.** Transições partindo de q_1:

  1. $q_1 \xrightarrow{b} q_1$: temos apenas a seqüência $< b, c >$ em $L_2(q_1)$. É construída a 3-seqüência $< \mathbf{b}, b, c >$.
  2. $q_1 \xrightarrow{c} q_3$: não há seqüências em $L_2(q_3)$ e portanto nenhuma 3-seqüência é construída.

  3-seqüências candidatas válidas com relação a q_1: $C_3(q_1) = \{< b, b, c >\}$

- **Com respeito a q_2.** Transições partindo de q_2:

  1. $q_2 \xrightarrow{d} q_2$: temos as seqüências $< d, c >$ e $< d, e >$ em $L_2(q_2)$. Logo, são construídas as 3-seqüências $< \mathbf{d}, d, c >$ e $< \mathbf{d}, d, e >$.
  2. $q_2 \xrightarrow{e} q_3$: $L_2(q_3) = \emptyset$. Nenhuma seqüência é gerada.
  3. $q_2 \xrightarrow{c} q_3$: $L_2(q_3) = \emptyset$. Nenhuma seqüência é gerada.

  3-seqüências candidatas válidas com relação a q_2: $C_3(q_2) = \{< d, d, c >, < d, d, e >\}$

- **Com respeito a q_3.** Não há transições partindo de q_3. $C_3(q_3) = \emptyset$

Portanto, o conjunto das 3-seqüências candidatas válidas com relação a $\mathcal{A}_\mathcal{R}$ é $C_3 = C_3(q_0) \cup C_3(q_1) \cup C_3(q_2) \cup C_3(q_3) = \{< a, b, c >, < a, d, c >, < a, d, e >, < b, b, c >, < d, d, c >, < d, d, e >\}$.

3.3 Mineração de Padrões Seqüenciais Multi-Dimensionais

O problema de mineração de padrões seqüenciais multi-dimensionais foi introduzido em [4].

Padrões seqüenciais podem estar associados a diferentes atributos, que formam um espaço dimensional múltiplo. Por exemplo, seqüências de compras de clientes geralmente estão associadas com a profissão, a idade e a região em que o cliente vive. Neste caso, os atributos *profissão*, *idade* e *região* representam três dimensões de informação que podem ser consideradas no processo de mineração dessas seqüências. Os algoritmos propostos para mineração de padrões seqüenciais anteriores a este estudo não levam em consideração a informação de tais dimensões e são capazes apenas de descobrir relacionamentos entre *itemsets*. As dimensões de informação consideradas em [4] são invariantes no tempo, ou seja, no caso do exemplo citado anteriormente os valores para os atributos *profissão*, *idade* e *região* permanecem constantes para um mesmo cliente no decorrer de suas transações. A tabela 3.1 ilustra um pequeno banco de dados de transações de clientes com múltiplas dimensões. Nela aparecem os atributos *CodC*, *Tempo*, *Produto*, *Profissão* e *Idade*, dos quais os dois últimos correspondem a dimensões temporalmente invariantes.

Analisando esta tabela notamos que a seqüência de transações $< (Celular), (Camera) >$ juntamente com os valores *Médico* e *Jovem* para os atributos Profissão e Idade, respectivamente, aparecem para 2 dos 4 clientes existentes. Se consideramos um nível mínimo

CodC	Tempo	Produto	Profissão	Idade
01	1	Celular	Médico	Jovem
01	2	Câmera	Médico	Jovem
01	3	Computador	Médico	Jovem
02	1	TV	Professor	Adulto
02	2	Microondas	Professor	Adulto
03	1	DVD	Médico	Jovem
03	2	Celular	Médico	Jovem
03	3	Câmera	Médico	Jovem
04	1	Celular	Professor	Jovem
04	2	Câmera	Professor	Jovem

Tabela 3.1: Banco de dados de transações multi-dimensionais

de suporte $\alpha = 50\%$ então a seqüência *<(Celular), (Camera)>[Medico,Jovem]* representa um *padrão seqüêncial multi-dimensional*.

Em [36], H. Pinto apresenta os algoritmos *PSFP* e *HYBRID* para mineração de padrões seqüenciais multi-dimensionais. *PSFP* combina o algoritmo para mineração de padrões seqüenciais simples *PrefixSpan* [12] com o algoritmo para mineração de regras de associação *FP-growth* [31] e *HYBRID* utiliza-se de *PrefixSpan* e do algoritmo *BUC* [37] para encontrar as combinações de valores das dimensões.

Em [38], Rosana Santos propõe a mineração desses padrões multi-dimensionais onde as dimensões de informação variam no tempo. Neste caso, os valores para os atributos *Profissão* e *Idade* do banco de dados da tabela 3.1, por exemplo, poderiam variar para um mesmo cliente, no decorrer de suas transações. Ary Santos propõe em [39] o uso da programação genética para mineração de padrões seqüenciais *generalizados*. Tais padrões generalizam os padrões seqüenciais clássicos e multi-dimensionais acrescentando variações temporais a diferentes atributos em uma ou mais tabelas de um banco de dados temporal.

Parte II

Mineração de Padrões Seqüenciais Múltiplos

Capítulo 4

O Problema de Mineração de Padrões Seqüenciais Múltiplos

Como já mencionado no capítulo 1, a maioria dos padrões temporais já estudados até o momento pode ser especificada por formalismos que podem ser reduzidos a fórmulas da Lógica Temporal Proposicional. No capítulo 3 descrevemos vários algoritmos para mineração do padrão seqüencial (proposicional) da forma $< i_1, i_2, ..., i_n >$, onde cada i_k representa um conjunto de itens.

Pesquisa recente em ILP (*Inductive Logic Programming*) [40, 41] propõe formalismos temporais que são mais expressivos que aqueles usados até então para a representação de padrões seqüenciais. Ainda assim, tais formalismos, que são baseados na Lógica Temporal Proposicional, não são expressivos o suficiente para especificar muitos padrões freqüentes, como por exemplo *logs* de usuários no sistema Unix [42]. Sem dúvida, a necessidade de formalismos mais expressivos para a especificação de padrões temporais surgem naturalmente, assim como métodos para minerá-los em grandes bases de dados.

Neste capítulo, formalizamos o problema proposto nesta dissertação correspondente à *mineração de padrões seqüenciais múltiplos* [43], que são padrões temporais de primeira ordem. Inicialmente, apresentamos os conceitos relacionados a *seqüências múltiplas*; em seguida definimos o esquema do banco de dados a ser minerado e o *padrão seqüencial múltiplo* propriamente dito; definimos o termo *suporte* no contexto de seqüências múltiplas e finalizamos o capítulo descrevendo o problema de mineração em questão.

4.1 Formalização do Problema

Suponha a existência de um conjunto finito de itens $\mathcal{I}$ e um conjunto de identificadores de objetos $\mathcal{C}$. Os elementos de $\mathcal{C}$ serão chamados de *clientes*. Os itens são denotados por a, b, c, etc, e identificadores de cliente por c_1, c_2, c_3, etc. Por motivo de simplificação da apresentação, só consideramos padrões seqüenciais cujos elementos são *itens* ao invés de *conjuntos de itens*. Assim, visto que o principal objetivo desta dissertação é explorar o problema de minerar padrões que correspondem a *conjuntos de seqüências*, acreditamos que considerar seqüências de *itemsets* poderia adicionar dificuldade desnecessária ao problema em si no qual estamos interessados.

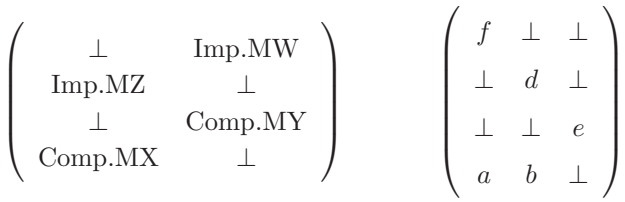
4.1.1 Seqüência Simples e Seqüência Múltipla

Definimos uma *seqüência simples*[1] como sendo uma lista $s = < i_1, ..., i_m >$, onde cada elemento $i_j \in \mathcal{I} \cup \{\perp\}$. O símbolo $\perp$ é utilizado para indicar que a transação em questão não é importante. O número m corresponde ao *comprimento* de s, que é denotado por $\mid s \mid$. Uma *seqüência múltipla* é um conjunto finito $\sigma = \{s^1, \ldots, s^n\}$, onde cada s^i é uma seqüência simples e para cada $i, j \in \{1, \ldots, n\}$ temos $\mid s^i \mid = \mid s^j \mid = k$. O valor k é denominado de *comprimento* de σ e é denotado por $comp(\sigma)$. O número de seqüências em σ é chamado de *rank* de σ e é denotado por $rank(\sigma)$. O j-ésimo elemento da seqüência s^i é denotado por s^i_j. Por exemplo, o conjunto $\{ < a, \perp, \perp, f >, < b, \perp, d, \perp >, < \perp, e, \perp, \perp > \}$ corresponde a uma seqüência múltipla de *rank* 3 e comprimento 4.

Uma seqüência múltipla pode ser representada através de uma matriz, onde cada coluna (ordenada de baixo para cima) está relacionada com uma seqüência simples do conjunto de seqüências. O exemplo a seguir ilustra graficamente duas seqüências múltiplas.

Exemplo 4.1.1 Considere a tabela de transações Tr ilustrada no exemplo 1.1.2. A seqüência de transações efetuada pelos clientes *Paul* e *Mary* informalmente descrita nesse exemplo é expressa pela seqüência múltipla { <Comp. MX, $\perp$, Imp. MZ, $\perp$>, <$\perp$, Comp. MY, $\perp$, Imp. MW> }, cuja representação em matriz é dada na figura 4.1(a) a seguir.

[1]Por abreviação, também chamamos uma *seqüência simples* de apenas *seqüência*



(a) (b)

Figura 4.1: Representação em matriz de seqüências múltiplas

Nessa matriz, a primeira coluna está vinculada às transações efetuadas pelo cliente *Paul* e a segunda coluna diz respeito às transações de *Mary*. Cada linha da matriz informa as transações efetuadas pelos clientes em um determinado instante. A seqüência múltipla $\{ < a, \perp, \perp, f >, < b, \perp, d, \perp >, < \perp, e, \perp, \perp > \}$ é representada pela matriz da figura 4.1(b).

4.1.2 O Banco de Dados

Agora, vamos apresentar o esquema do banco de dados sobre o qual mineramos nossos padrões. Tal banco de dados armazena transações de clientes que estão organizados em grupos. Os clientes pertencentes a um mesmo grupo se relacionam um com o outro por algum critério (por exemplo, clientes que trabalham em um mesmo local). Considere o esquema de banco de dados $\mathcal{D} = \{Tr(IdG, IdCl, T, Item)\}$. Um *banco de dados de transações múltiplas* D é uma instância sobre $\mathcal{D}$. Aqui, T é o atributo tempo cujo domínio é $\mathbb{N}$. Os atributos *IdG*, *IdCl* e *Item* denotam os identificadores dos grupos, clientes e itens, respectivamente. Seus domínios são $\mathbb{N}$, $\mathcal{C}$ e $\mathcal{I}$ respectivamente. A figura 4.2 a seguir ilustra um banco de dados de transações múltiplas.
A primeira tupla desse banco, por exemplo, informa que o cliente c_1 pertencente ao grupo 1, comprou o produto a no instante 1. Note que o grupo 1 possui 3 clientes, assim como os grupos 2 e 4 e o grupo 3 possui 2 clientes.

Um banco de dados de transações múltiplas pode ser facilmente transformado em uma tabela de seqüências múltiplas. A figura 4.3 ilustra a tabela de seqüências múltiplas correspondente ao banco de dados da figura 4.2. Os valores g do atributo IdG são chamados *grupos* e os valores m do atributo $MSeq$ são seqüências múltiplas. Se (g,m) está no banco de dados transformado então denotamos m por $S(g)$.

IdG	$IdCl$	T	$Item$	IdG	$IdCl$	T	$Item$
1	c_1	1	a	3	c_7	1	a
1	c_1	3	b	3	c_7	3	c
1	c_1	5	a	3	c_7	4	b
1	c_2	2	b	3	c_8	2	b
1	c_2	3	c	3	c_8	5	a
1	c_3	4	a	4	c_9	2	b
2	c_4	3	a	4	c_9	4	a
2	c_5	5	a	4	c_{10}	1	a
2	c_6	4	b	4	c_{10}	3	b
				4	c_{11}	5	d

Figura 4.2: Banco de dados de transações múltiplas

IdG	$MSeq$
1	$\{< a, \perp, b, \perp, a >, <\perp, b, c, \perp, \perp>, <\perp, \perp, \perp, a, \perp>\}$
2	$\{< a, \perp, \perp>, <\perp, \perp, a >, <\perp, b, \perp>\}$
3	$\{< a, \perp, c, b, \perp>, <\perp, b, \perp, \perp, a >\}$
4	$\{<\perp, b, \perp, a, \perp>, < a, \perp, b, \perp, \perp>, <\perp, \perp, \perp, \perp, d >\}$

Figura 4.3: Banco de Dados Transformado: tabela de seqüências múltiplas

Cada seqüência múltipla dessa tabela é obtida segundo as transações dos clientes de um *mesmo* grupo do banco de dados de transações. Cada seqüência simples que compõe a seqüência múltipla está associada a um cliente do grupo. A seqüência múltipla $S(1) = \{< a, \perp, b, \perp, a >, <\perp, b, c, \perp, \perp>, <\perp, \perp, \perp, a, \perp>\}$, por exemplo, é obtida através das transações dos clientes relativos ao grupo 1. A seqüência $< a, \perp, b, \perp, a >$ está vinculada ao cliente c_1. As seqüências $<\perp, b, c, \perp, \perp>$ e $<\perp, \perp, \perp, a, \perp>$ dizem respeito aos clientes c_2 e c_3 respectivamente. Note que os clientes pertencentes ao grupo 1 efetuam suas transações em 5 instantes diferentes (instantes 1,2,3,4 e 5). Por esse motivo as seqüências que compõem $S(1)$ possuem comprimento 5. Observe ainda que o cliente c_1 efetua suas transações nos instantes 1, 3 e 5 e por não realizar nenhuma transação nos instantes 2 e 4, a seqüência $< a, \perp, b, \perp, a >$ associada a c_1 possui o símbolo $\perp$ nas posições 2 e 4.

4.1.3 O Padrão Seqüencial Múltiplo

Uma vez apresentado o banco de dados de seqüências múltiplas, definimos o padrão seqüencial que estamos interessados em minerar.

Definição 4.1.1 (padrão seqüencial múltiplo). Seja $\sigma = \{s^1, \ldots, s^n\}$ uma seqüência múltipla de comprimento k. Dizemos que σ é um *padrão seqüencial múltiplo* (ou *psm*) se σ satisfaz as seguintes condições: (1) para cada $j \in \{1, ..., k\}$ existe um $i \in \{1, ..., n\}$ tal que $s^i_j \in \mathcal{I}$ e para todo $l \neq i$ temos $s^l_j = \perp$, (2) para cada $i \in \{1, \ldots, n\}$ existe um $j \in \{1, \ldots, k\}$ tal que $s^i_j \neq \perp$.

As condições (1) e (2) da definição anterior são interpretadas na matriz representando o *psm* como segue: (1) em cada linha da matriz existe uma única posição contendo um item e todas as outras posições contêm o elemento $\perp$. Intuitivamente, essa condição significa que em cada instante estamos interessados na compra realizada por um único cliente. (2) em cada coluna da matriz existe pelo menos uma posição contendo um item. Isso significa que um *psm* não conterá uma coluna com o elemento $\perp$ em todas as posições. O seguinte exemplo ilustra a definição de padrão seqüencial múltiplo.

Exemplo 4.1.2 Considere as cinco matrizes dadas abaixo. A matriz (a) representa o padrão seqüêncial múltiplo $\sigma = \{< a, \perp >, <\perp, c >\}$ cujo *rank* e comprimento são ambos 2. Neste caso, as duas condições acima são satisfeitas. O mesmo acontece para a matriz (e) cujo *rank* é 3 e o comprimento é 4. As matrizes (b), (c) e (d) não representam padrões seqüenciais múltiplos. Na matriz (b), a coluna 2 não contém itens (a condição (2) não é satisfeita); na matriz (c), a primeira linha contém 2 itens (a condição (1) não é satisfeita) e na matriz (d), a linha 2 não contém itens (a condição (1) não é satisfeita).

$$\begin{pmatrix} \perp & c \\ a & \perp \end{pmatrix} \qquad \begin{pmatrix} c & \perp \\ b & \perp \\ a & \perp \end{pmatrix} \qquad \begin{pmatrix} \perp & \perp & a \\ b & c & \perp \end{pmatrix} \qquad \begin{pmatrix} b \\ \perp \\ a \end{pmatrix} \qquad \begin{pmatrix} \perp & \perp & d \\ \perp & c & \perp \\ b & \perp & \perp \\ a & \perp & \perp \end{pmatrix}$$

(a) (b) (c) (d) (e)

Intuitivamente, a matriz (b) é eliminada porque toda a informação que ela contém (que nos interessa) pode ser expressa pela seqüência múltipla (simples) {<a,b,c>}. A

matriz (d) também não constitui um *psm* porque toda a informação que ela contém pode ser expressa pela seqüência múltipla {<a,b>}. Finalmente, a seqüência múltipla ilustrada em (c) não é considerada um *padrão* seqüencial múltiplo porque ela contém informações que saem do escopo de nosso trabalho (dois itens associados a diferentes clientes em um mesmo instante: os itens b e c aparecem na primeira linha).

Dada uma seqüência múltipla σ, se σ é um *psm*, então $rank(\sigma) \leq comp(\sigma)$. De fato, pois a condição (1) acima implica que existe uma função f_σ: {1,...,$comp(\sigma)$} $\longrightarrow$ {1,...,$rank(\sigma)$} e a condição (2) implica que f_σ é sobrejetora. Logo, $rank(\sigma) \leq comp(\sigma)$.

Observe que um padrão múltiplo, sendo formado por um conjunto de seqüências, pode ser representado por diversas matrizes, pois não existe uma ordem entre as seqüências desse conjunto. Entretanto, consideraremos apenas uma dessas matrizes para representar o padrão, onde as colunas estão ordenadas de acordo com os instantes em que as transações ocorrem. Por exemplo, a figura 4.4 a seguir ilustra um *psm* e sua versão ordenada (à direita). Um procedimento para ordenar as colunas de um padrão seqüencial múltiplo é dado na figura 4.5.

$$\begin{pmatrix} \perp & b & \perp \\ d & \perp & \perp \\ \perp & \perp & a \\ \perp & \perp & b \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} \perp & \perp & b \\ \perp & d & \perp \\ a & \perp & \perp \\ b & \perp & \perp \end{pmatrix}$$

Figura 4.4: Um *psm* e sua versão ordenada (à direita)

A partir deste ponto e também para os demais capítulos, consideramos que todos os *psm's* referenciados estejam assim ordenados.

4.1.4 Suporte de um Padrão Seqüencial Múltiplo

A seguir, definimos o conceito de *inclusão* entre seqüências múltiplas. Antes disso, definimos uma relação de ordem entre os elementos $x, y \in \mathcal{I} \cup \{\perp\}$. Dizemos que $x \preceq y$ se $x, y \in \mathcal{I} \cup \{\perp\}$ e ($x = y$ ou $x = \perp$). Por exemplo, considere $\mathcal{I}$ = {a,b,c}. Temos que $a \preceq a$, $b \preceq b$, $\perp \preceq a$, $\perp \preceq b$, etc.

```
Procedure OrdenaColunas(σ: psm para ordenar)
Begin
    σ_ord = ( );
    For(i = 1; comp(σ); i++)
    Begin
        Encontre a coluna s ∈ σ tal que o elemento na linha i ∈ I;
        If (s ∉ σ_ord) then σ_ord = append(σ_ord,s);
    End
    return σ_ord;
End
```

Figura 4.5: Procedimento *OrdenaColunas*

Definição 4.1.2 (inclusão de seqüências múltiplas). Seja $\sigma = \{s^1, ..., s^m\}$ e $\tau = \{t^1, \ldots, t^n\}$ duas seqüências múltiplas. Dizemos que σ é uma *sub-seqüência múltipla* de τ (ou que σ *está contida* em τ, denotado por $\sigma \subseteq \tau$) se existem $j_1, ..., j_m \in \{1, ..., n\}$, $j_p \neq j_q$ para $p \neq q$, e existem $i_1 < ... < i_k$, onde $k = comp(\sigma)$ tal que: $s_q^p \preceq t_{i_q}^{j_p}$ para todo $p \in \{1, ..., m\}$ e $q \in \{1, ..., k\}$. Se consideramos a representação em matriz de σ e τ, isto significa que σ pode ser obtida de τ considerando as colunas $j_1, ..., j_m$ e as linhas $i_1 < ... < i_k$ em τ. Assim, se $\sigma \subseteq \tau$ então $comp(\sigma) \leq comp(\tau)$ e $rank(\sigma) \leq rank(\tau)$.

Exemplo 4.1.3 Considere as seqüências múltiplas σ e τ ilustradas abaixo: [2]

$$\sigma = \begin{pmatrix} \perp & c \\ a & \perp \end{pmatrix} \qquad \tau = \begin{pmatrix} f & e & c \\ \perp & b & \perp \\ a & d & d \end{pmatrix}$$

Observe que σ é uma sub-seqüência múltipla de τ, ou seja, $\sigma \subseteq \tau$. Neste caso, $k = comp(\sigma) = 2$, $n = rank(\sigma) = 2$ e $m = rank(\tau) = 3$. Devemos considerar as linhas 1 e 3 e as colunas 1 e 3 em τ, ou seja, $j_1 = 1$, $j_2 = 3$, $i_1 = 1$ e $i_2 = 3$ em $\sigma_q^p \preceq \tau_{i_q}^{j_p}$ para cada $p =$ 1,2 e $j =$ 1,2.

Definição 4.1.3 (suporte de um *psm*) Seja D um banco de dados em sua versão transformada, isto é, contendo os campos IdG e $MSeq$ (como na figura 4.3) e a tupla $(g, S(g)) \in D$. Dizemos que g *suporta* um *psm* σ se $\sigma \subseteq S(g)$. O *suporte* de um *psm* σ é definido por: $sup(\sigma) = \frac{|\{g \mid g \in \Pi_{IdG} D \text{ e } g \text{ suporta } \sigma\}|}{|D|}$.

[2] Neste exemplo, σ é um *psm*, mas τ não é. A relação de inclusão $\subseteq$ é definida sobre seqüências múltiplas em geral e não somente sobre *psm's*

Um *psm* σ é dito *freqüente* se $\sup(\sigma) \geq sup\_min$, onde sup_min corresponde a um dado nível mínimo de suporte. Por exemplo, se consideramos o banco de dados D da figura 4.3, o *psm* do exemplo 4.1.2(a) é suportado somente pelo grupo $g = 1$, pois $\{< a, \perp >, <\perp, c >\}$ está contida em $S(1) = \{< a, \perp, b, \perp, a >, <\perp, b, c, \perp, \perp>, <\perp, \perp, \perp, a, \perp>\}$ (o item a da seqüência $< a, \perp>$ aparece na primeira posição de $< a, \perp, b, \perp, a >$ e o item c de $<\perp, c >$ aparece em $<\perp, b, c, \perp, \perp>$ em uma posição após a primeira (terceira posição). Assim, o suporte desse *psm* é 0.25 (suportado por um grupo dentre os quatro). Se $sup\_min = 0.2$ então tal *psm* será freqüente em D.

A freqüência de um *psm* é uma propriedade antimonotônica:

Proposição 4.1.1 Sejam os *psm's* σ e τ e um nível mínimo de suporte sup_min. Se $\sigma \subseteq \tau$ e $sup(\tau) \geq sup\_min$ então $sup(\sigma) \geq sup\_min$.

Prova: Se um grupo g do banco de dados suporta τ então $\tau \subseteq S(g)$. Logo, $\sigma \subseteq S(g)$. Conseqüentemente, g suporta σ e assim, $sup(\sigma) \geq sup(\tau) \geq sup\_min$.

Formulação do Problema. O problema de mineração em que estamos interessados pode ser descrito como segue:

- **Dado**: um banco de dados de seqüências múltiplas D e um nível mínimo de suporte sup_min;
- **Encontrar**: todos os *psm's* freqüentes com relação a D e sup_min.

Nos próximos dois capítulos, apresentaremos dois algoritmos para mineração de padrões seqüenciais múltiplos, de acordo com a formulação acima. São os algoritmos PM (*Projection Miner*) e SM (*Simultaneous Miner*).

Capítulo 5

O Algoritmo PM

Neste capítulo apresentamos o algoritmo **PM** (***P**rojection **M**iner*) para mineração de padrões seqüenciais múltiplos. PM encontra todos os *psm's* freqüentes em um banco de dados D com relação a um suporte mínimo α.

Gostaríamos de enfatizar aqui que o algoritmo PM foi desenvolvido com o intuito de utilizar métodos e algoritmos já existentes (de mineração de seqüências simples), para a mineração de seqüências múltiplas. Nosso objetivo foi tentar adaptar uma técnica de mineração bem conhecida, o algoritmo GSP (que minera seqüências simples), visando realizar a mineração de seqüências múltiplas. Seguimos por esta direção quando percebemos que um padrão seqüencial múltiplo pode ser representado por duas seqüências simples, como será descrito mais adiante. Assim, consideramos a possibilidade de desenvolver um algoritmo que trabalhasse com essas seqüências simples, ao invés de considerar o padrão múltiplo em sua forma original. Propomos então o algoritmo *PM*, que é descrito em detalhes neste capítulo.

A partir deste ponto, vamos utilizar a notação Σ_k^n para denotar o conjunto de todos os *psm's* de *rank* n e comprimento k. As notações C_k^n e L_k^n serão utilizadas para denotar o conjunto de *psm's candidatos* e *freqüentes* de rank n e comprimento k ($n \leq k$) respectivamente.

5.1 Decompondo um *PSM*

Primeiramente, mostraremos como um *psm* pode ser caracterizado por meio de duas seqüências simples (proposicionais).

$$\sigma = \begin{pmatrix} \perp & \perp & d \\ \perp & c & \perp \\ b & \perp & \perp \\ a & \perp & \perp \end{pmatrix}$$

Figura 5.1: Um *psm*

Seja σ um *psm* ordenado tal que $comp(\sigma) = k$ e $rank(\sigma) = n$. A *função característica* de σ é a função $f_\sigma : \{1, ..., k\} \rightarrow \{1, ..., n\}$ tal que $f_\sigma(i)$ é o número correspondente a (única) coluna tendo um elemento de $\mathcal{I}$ na linha i. Agora, associamos ao *psm* σ duas seqüências de comprimento k, denotadas por $\Pi_i(\sigma)$ (a *seqüência de itens* de σ) e $\Pi_s(\sigma)$ (a *seqüência de formato de* σ ou simplesmente *formato* de σ), como segue: $\Pi_i(\sigma)$ é a projeção de σ sobre o eixo do tempo e $\Pi_s(\sigma) = < f_\sigma(1), ..., f_\sigma(k) >$. Por exemplo, se σ é o *psm* ilustrado na figura 5.1, então $\Pi_i(\sigma) =< a, b, c, d >$ e $\Pi_s(\sigma) =< 1, 1, 2, 3 >$. É fácil verificar que um *psm* é completamente caracterizado por sua seqüência de itens e sua seqüência de formato. Os números n e k são chamados de *rank* e *comprimento* da seqüência de formato $\Pi_s(\sigma)$ respectivamente. Denotamos por $\mathbf{proj}(\sigma)$ o par ($\Pi_i(\sigma)$,$\Pi_s(\sigma)$). A função inversa é denotada por $\boxtimes$, isto é: $\Pi_i(\sigma) \boxtimes \Pi_s(\sigma) = \sigma$.

Note que, devido ao fato das seqüências (colunas nas representações em matriz) dos *psm's* estarem dispostas em uma ordem específica (veja procedimento *OrdenaColunas* no capítulo 4, figura 4.4), suas seqüências de formato apresentam uma forma especial: se $j \in \mathbb{N}$ aparece pela primeira vez na posição i do formato, então nenhum $k > j$ pode aparecer antes de i. Por exemplo, <1,2,1,3,1> é uma seqüência de formato, mas <1,**3**,2,1,1> não é.

5.2 O Algoritmo

PM é um algoritmo baseado na técnica *Apriori* que a cada iteração (n,k) produz o conjunto L_k^n de *psm's* freqüentes de rank n e comprimento k. O algoritmo executa iterativamente as operações de *geração de candidatos*, *poda* e *contagem de suporte*.

A estrutura geral do algoritmo PM é descrita a seguir. PM inicialmente gera os *psm's* freqüentes de rank 1 (L_1^1, L_2^1, L_3^1, etc) usando para esta tarefa um algoritmo de mineração de seqüências (simples) (por exemplo, o algoritmo GSP que apresentamos no

capítulo 3, seção 3.1.2 [2]). Em seguida, para cada $n > 1$ (um dado rank) ele gera iterativamente os conjuntos C_k^n de *psm's candidatos* de comprimento $k \geq n$. Para o caso inicial onde $k = n$, C_n^n é gerado a partir do conjunto L_{n-1}^{n-1}, que já foi gerado no passo anterior, correspondente ao rank $n-1$. Para o caso $k > n$, o conjunto C_k^n (contendo os *psm's potencialmente* freqüentes de rank n e comprimento k) é gerado a partir de L_{k-1}^{n-1} e L_{k-1}^n que foram gerados em passos anteriores. PM calcula o suporte desses candidatos percorrendo o banco de dados uma única vez e no final da iteração ele gera o conjunto L_k^n, dos *psm's* candidatos que são realmente freqüentes. A estrutura geral do algoritmo, com relação à ordem em que os padrões são obtidos é dada na figura 5.2.

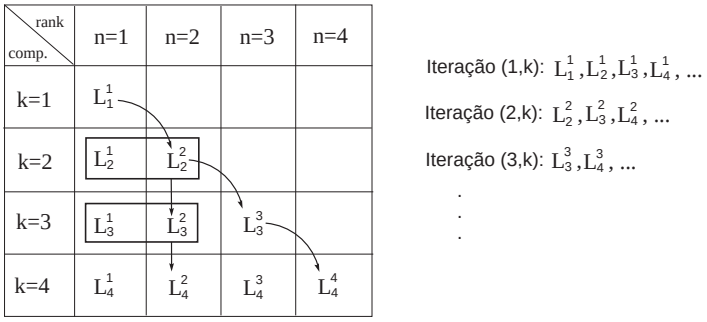


Figura 5.2: Estrutura em que PM obtém os conjuntos de *psm's* freqüentes

Agora, descreveremos em mais detalhes como PM obtém o conjunto L_k^n a partir de L_{k-1}^n e L_{k-1}^{n-1} (obtidos em iterações anteriores) para o caso $k > n$. O processo é descrito a seguir:

1. os *psm's* em L_{k-1}^n e L_{k-1}^{n-1} são projetados para obtenção de suas seqüências de itens e suas seqüências de formato. Projetando cada *psm* em L_{k-1}^n, obtemos um conjunto de seqüências de itens (denotado por LI_{k-1}^n) e um conjunto de seqüências de formato (denotado por LS_{k-1}^n). Analogamente obtemos LI_{k-1}^{n-1} e LS_{k-1}^{n-1} projetando os padrões em L_{k-1}^{n-1};
2. os conjuntos LI_{k-1}^n e LI_{k-1}^{n-1} de seqüências de itens de comprimento $k-1$ são utilizados para geração do conjunto de seqüências de itens de comprimento k, denotado por CI_k^n. Da mesma forma, os conjuntos de formatos LS_{k-1}^n e LS_{k-1}^{n-1} são utilizados para obtenção dos formatos candidatos de comprimento k, CS_k^n;

3. as seqüências de itens de CI_k^n que possuem uma sub-seqüência não pertencente a $LI_{k-1}^n \cup LI_{k-1}^{n-1}$ são podadas. Da mesma forma, são podados os formatos de CS_k^n possuindo um sub-formato não pertencente a $LS_{k-1}^n \cup LS_{k-1}^{n-1}$;
4. as seqüências de itens de CI_k^n são combinadas com os formatos de CS_k^n (através da função $\boxtimes$) para formação do conjunto C_k^n de *psm's* candidatos completos;
5. o banco de dados é então percorrido para contagem do suporte dos *psm's* candidatos de C_k^n. O conjunto L_k^n é obtido eliminando-se de C_k^n aqueles *psm's* cujo suporte não atingiu o suporte mínimo.

A figura 5.3 a seguir ilustra o processo descrito anteriormente, em que L_k^n é obtido a partir de L_{k-1}^n e L_{k-1}^{n-1}.

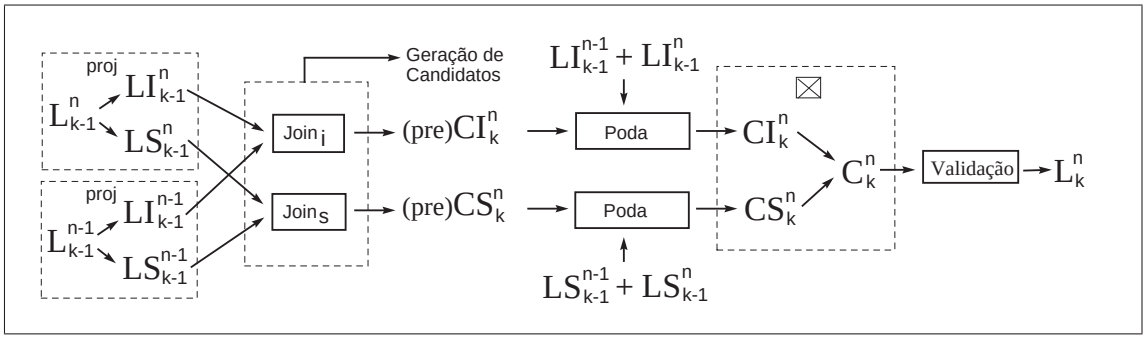


Figura 5.3: L_k^n é obtido a partir de L_{k-1}^n e L_{k-1}^{n-1}

Para o caso inicial em que $n = k$, L_n^n é obtido apenas de L_{n-1}^{n-1}. Repare que, neste caso, o único formato possível para os *psm's* de L_n^n é o formato $< 1, 2, ..., n >$. Logo, apenas o conjunto de seqüências de itens CI_n^n é gerado de LI_{n-1}^{n-1}. Em seguida CI_n^n é combinado com o formato $< 1, 2, ..., n >$, produzindo C_n^n.

O algoritmo PM é dado na figura 5.4. Nos passos **4.5** e **4.9.6** usamos o operador $\boxtimes$ para combinar as seqüências de itens com as seqüências de formato. Nesse processo, cada seqüência de formato em CS_k^n é combinada com todas as seqüências de itens de CI_k^n. O conjunto resultante de *psm's* é C_k^n.

O operação $Join_I$ realiza a junção das seqüências de itens em $LI_{k-1}^{n-1} \cup LI_{k-1}^n$ e retorna o conjunto CI_k^n de seqüências de itens candidatas. Analogamente, a operação $Join_S$ obtém o conjunto de seqüências de formato candidatas CS_k^n fazendo a junção dos formatos em $LS_{k-1}^{n-1} \cup LS_{k-1}^n$. Os detalhes das operações *join* são dados na seção 5.2.1 a seguir. A

Algoritmo PM(α - suporte mínimo, D - banco de dados)

1. N = número de seqüências múltiplas de D;

2. $k = 0$; $\quad n = 1$;

3. Repeat

3.1 $k = k + 1$;

3.2 L_k^1 = seqüências freqüentes de rank 1 e comprimento k (use GSP);

3.3 if $L_k^1 \neq \emptyset$ **then** $\{\ LI_k^1 = L_k^1; \quad LS_k^1 = \{< 1, 1, \ldots, 1 >\}\ \}$;

Until $L_k^1 = \emptyset$;

4. Repeat

4.1 $n = n + 1$; $\quad k = n$;

4.2 $CI_n^n = Join_I(LI_{n-1}^{n-1}, LI_{n-1}^{n-1})$; (seqüências de itens candidatas de LI_{n-1}^{n-1})

4.3 $CS_n^n = \{< 1, 2, ..., n >\}$

4.4 remova todas seq. candidatas $s \in CI_n^n$ tal que $\exists s' \subseteq s$, $l(s') = n - 1$ e $s' \notin LI_{n-1}^{n-1}$ (poda);

4.5 $C_n^n = CI_n^n \boxtimes CS_n^n$ (constrói os *psm's* candidatos)

4.6 Foreach grupo g em D **do**

Incremente o contador (cont) de todos os *psm's* candidatos em C_n^n contidos em $S(g)$;

4.7 L_n^n = candidatos em C_n^n com *cont* $\geq \alpha N$;

4.8 $(LI_n^n, LS_n^n) =$ **proj**(L_n^n); (faz a projeção e obtém as seqs. de itens e as seqs. de formato de L_n^n;

4.9 Repeat

4.9.1 $k = k + 1$;

4.9.2 $CI_k^n = Join_I(LI_{k-1}^{n-1} \cup LI_{k-1}^n, LI_{k-1}^{n-1} \cup LI_{k-1}^n)$;

(as seqüências de itens candidatas são geradas de LI_{k-1}^{n-1} e LI_{k-1}^n)

4.9.3 remova todas as candidatas $s \in CI_k^n$ tal que $\exists s' \subseteq s$, $l(s') = k-1$ e $s' \notin LI_{k-1}^{n-1} \cup LI_{k-1}^n$;

(poda seqüências de itens)

4.9.4 $CS_k^n = Join_S(LS_{k-1}^{n-1} \cup LS_{k-1}^n, LS_{k-1}^{n-1} \cup LS_{k-1}^n)$;

(as seqüências de formato candidatas são geradas de $LS_{k-1}^{n-1} \cup LS_{k-1}^n$)

4.9.5 remova todas as seq. candidatas $s \in CS_k^n$ tal que $\exists s' \subseteq s$ e $s' \notin LS_{k-1}^{n-1} \cup LS_{k-1}^n$;

(poda seqüências de formato)

4.9.6 $C_k^n = CI_k^n \boxtimes CS_k^n$

4.9.7 Foreach grupo g em D **do**

Incremente o contador (cont) de todos os *psm's* candidatos em C_k^n contidos em $S(g)$;

4.9.8 L_k^n = candidatos em C_k^n com *cont* $\geq \alpha N$;

4.9.9 $(LI_k^n, LS_k^n) =$**proj**(L_k^n); (obtém as seqs. de itens e as seqüências de formato de L_k^n)

Until $L_k^n = \emptyset$;

Until $L_n^n = \emptyset$;

Figura 5.4: Algoritmo PM

função **proj** realiza as projeções Π_i e Π_s para todos *psm's* em L_k^n e retorna os conjuntos LI_k^n e LS_k^n.

5.2.1 Geração de Candidatos

Como já mencionado anteriormente, no processo de produção dos candidatos C_k^n, PM trabalha com as seqüências de itens e as seqüências de formato dos *psm's* de forma independente. As operações $Join_I$ e $Join_S$ utilizadas são descritas a seguir.

Join$_I$. Esta operação é realizada entre seqüências de itens candidatas e é executada como no algoritmo GSP (veja capítulo 3, seção 3.1.2). A junção de duas seqüências ocorre quando eliminando-se o primeiro item da primeira seqüência e o último item da segunda obtemos seqüências idênticas. A figura 5.5 abaixo ilustra a operação.

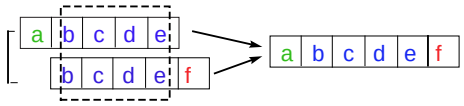
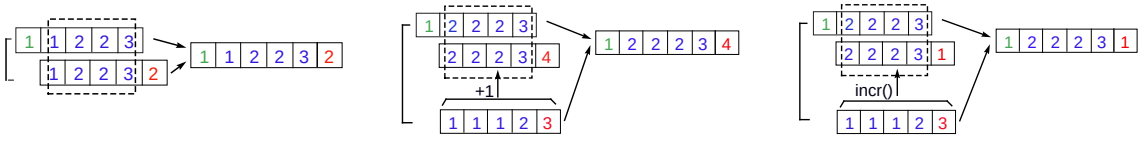


Figura 5.5: Fazendo a junção de *seqüências de itens* - $Join_I$

Join$_S$. Utilizada na geração de seqüências de formato candidatas. Há três possibilidades para obtenção de uma nova seqüência de formato. Os três casos são ilustrados na figura 5.6 e explicados a seguir.



Caso 1 Caso 2a Caso 2b

Figura 5.6: Fazendo a junção de *seqüências de formato* - $Join_S$

Seja $\sigma = < \sigma_1, \sigma_2, ..., \sigma_k >$ e $\tau = < \tau_1, \tau_2, ..., \tau_k >$ duas seqüências de formato de ranks n_σ e n_τ respectivamente. Denotamos por σ^{first} e τ^{last} as seqüências obtidas eliminando-se o primeiro elemento de σ e o último elemento de τ respectivamente. Se $\sigma^{first} = \tau^{last}$ então σ e τ podem ser ligadas. Neste caso (Caso 1), a seqüência resultante é $< \sigma_1, \sigma_2, ..., \sigma_k, \tau_k >$. Caso contrário, testamos se $\sigma_i^{first} = \tau_i^{last} + 1$ para todo $i = 1, ..., k$. Se esta condição é verificada (Caso 2a) então σ e τ podem ser ligadas e a seqüência resultante é $< \sigma_1, \ldots, \sigma_k, \tau_k + 1 >$. Agora testamos (Caso 2b) se $\sigma_i^{first} = \text{Inc}(\tau_i^{last})$ para todo $i = 1, ..., k$, onde Inc(x) $= x + 1$ se $x + 1 \leq max(n_\sigma, n_\tau)$ e Inc(x) $= 1$ caso contrário. Se a condição é verificada então uma segunda seqüência resultante também é obtida,

definada como $< \sigma_1, \ldots, \sigma_k, \mathrm{Inc}(\tau_k) >$.

O exemplo a seguir ilustra o processo de geração de candidatos do algoritmo PM.

Exemplo 5.2.1 Suponha que os conjuntos L_3^2 e L_3^1 contenham os *psm's* σ e τ dados abaixo:

$$\sigma = \begin{pmatrix} \perp & c \\ \perp & b \\ a & \perp \end{pmatrix} \text{ e } \tau = \begin{pmatrix} d \\ c \\ b \end{pmatrix} \qquad \gamma = \begin{pmatrix} \perp & d \\ \perp & c \\ \perp & b \\ a & \perp \end{pmatrix}$$

Realizamos a junção de σ e τ para obter o *psm* candidato γ de rank 2 e comprimento 4. Fazendo as projeções em σ e τ obtemos: $\Pi_i(\sigma) = < a, b, c > \in LI_3^2$, $\Pi_s(\sigma) =< 1, 2, 2 > \in LS_3^2$, $\Pi_i(\tau) =< b, c, d > \in LI_3^1$ e $\Pi_s(\tau) =< 1, 1, 1 > \in LS_3^1$. Juntando a seqüências de itens $< a, b, c >$ e $< b, c, d >$ obtemos a seqüência de itens $< a, b, c, d > \in LI_4^2$. Fazendo-se a junção das seqüências de formato $< 1, 2, 2 >$ e $< 1, 1, 1 >$ (Caso 2a), obtemos a seqüência $< 1, 2, 2, 2 > \in LS_4^2$. Observe que o Caso 2b não se aplica aqui. As seqüências $< a, b, c, d >$ e $< 1, 2, 2, 2 >$ unicamente caracterizam o *psm* $\gamma \in C_4^2$.

5.2.2 Cálculo do Suporte

Uma vez produzido o conjunto C_k^n de *psm's* candidatos de rank n e comprimento k, o algoritmo percorrerá o banco de dados para a contagem do suporte desses candidatos. A técnica usada nesta fase é a mesma usada pelo algoritmo SM e será apresentada no final do capítulo 6.

Capítulo 6

O Algoritmo SM

Neste capítulo apresentamos o algoritmo **SM** (***S**imultaneous **M**iner*) para mineração de padrões seqüenciais múltiplos. Diferentemente do algoritmo PM, apresentado no capítulo anterior, o algoritmo SM realiza a mineração dos padrões seqüenciais múltiplos sem decompô-los em componentes proposicionais, ou seja, SM trabalha com o padrão inteiro e não com suas componentes (seqüências de itens e seqüências de formato) de forma separada.

Gostaríamos de lembrar que as notações C_k^n e L_k^n denotam respectivamente os conjuntos dos *psm's candidatos* e *psm's freqüentes* de rank n e comprimento k, onde $n \leq k$.

6.1 Idéia Geral

A estrutura geral do algoritmo SM com relação a ordem em que são gerados os conjuntos L_k^n é a mesma do algoritmo PM, ou seja, SM inicialmente gera os *psm's* freqüentes de rank 1 (L_1^1, L_2^1, L_3^1, etc), em seguida, para cada $n > 1$ (um dado rank) ele gera iterativamente os conjuntos C_k^n de *psm's candidatos* de comprimento $k \geq n$. Para o caso inicial onde $k = n$, C_n^n é gerado do conjunto L_{n-1}^{n-1}, que já foi gerado no passo anterior e para o caso $k > n$, o conjunto C_k^n é gerado a partir de L_{k-1}^{n-1} e L_{k-1}^n que foram gerados em passos anteriores.

6.2 O Algoritmo

Na figura 6.1(a), mostramos quais conjuntos anteriores são utilizados na obtenção do conjunto L_k^n. Observe, por exemplo, que o conjunto L_3^2 é obtido a partir de L_2^2 e L_2^1 e o conjunto L_3^3 é obtido de L_2^2. Na figura 6.1(b), ilustramos o processo de mineração por inteiro em que L_k^n $(k > n)$ é produzido a partir de L_{k-1}^n e L_{k-1}^{n-1}.

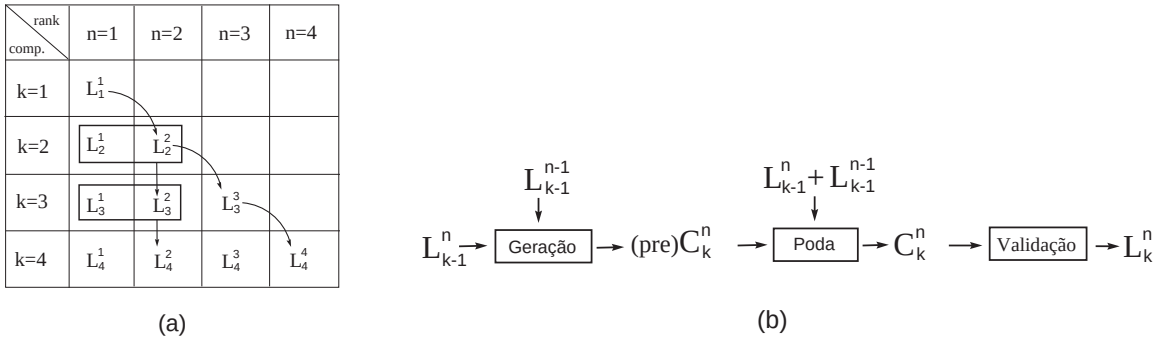


Figura 6.1: L_k^n é obtido a partir de L_{k-1}^{n-1} e L_{k-1}^n (para k > n)

O algoritmo SM é descrito na figura 6.2. Na iteração em que *psm's* freqüentes de rank n e comprimento k são gerados, o algoritmo inicialmente gera o conjunto C_k^n de *psm's* candidatos (passos **4.2** e **4.6.2**)[1] e em seguida poda de C_k^n os *psm's* contendo *sub-psm's* não pertencentes a L_{k-1}^{n-1} e L_{k-1}^n (de acordo com a proposição 4.1.1, esses *psm's* não têm chances de serem freqüentes - passos **4.3**, **4.6.3** e **4.6.4**). Após a podagem, o banco de dados é percorrido e o suporte dos candidatos é então calculado (passos **4.4** e **4.6.5**). Na seção 6.2.1, mostramos em detalhes como os *psm's* candidatos de C_k^n são obtidos a partir dos conjuntos L_{k-1}^{n-1} e L_{k-1}^n, que já foram calculados em passos anteriores.

6.2.1 Geração de Candidatos

Mostraremos nesta seção como são gerados os *psm's* em Σ_k^n (Σ_k^n denota o conjunto de *psm's* de rank n e comprimento k). Consideramos que as colunas de cada *psm* $\sigma \in \Sigma_k^n$ estejam ordenadas (segundo a ordem mencionada no capítulo 4, seção 4.1.3).

[1]Note que operador $\bowtie$ utilizado na descrição do algoritmo SM não possui a mesma função do operador $\boxtimes$ utilizado no capítulo 5 para associar seqüências de itens a seqüências de formato.

Algoritmo SM(α - suporte mínimo, D - banco de dados)

1. N = número de seqüências múltiplas em D;
2. $n = 1; k = 0$;
3. Repeat
  3.1 $k = k + 1$;
  3.2 L_k^1 = seqüências freqüentes de rank 1 e comprimento k; (utilize GSP)
Until $L_k^1 = \emptyset$;
4. Repeat
  4.1 $n = n + 1$; $k = n$;
  4.2 $C_n^n = L_{n-1}^{n-1} \bowtie L_{n-1}^{n-1}$; *(Novos psm's candidatos de rank n e comp. n são gerados de* L_{n-1}^{n-1}*)*
  (*Para detalhes sobre o operador* $\bowtie$*, veja seção 6.2.1*)
  4.3 delete todos os candidatos $\sigma \in C_n^n$ tal que $\exists \tau \subseteq \sigma$, $\tau \in \Sigma_{n-1}^{n-1}$ e $\tau \notin L_{n-1}^{n-1}$ (poda);
  4.4 Foreach grupo g em D **do**
    Incremente o contador (cont) de todos os *psm's* candidatos em C_n^n contidos em $S(g)$;
  4.5 L_n^n = candidatos em C_n^n com *cont* $\geq \alpha N$;
  4.6 Repeat
    4.6.1 $k = k + 1$;
    (Novos psm's candidatos de rank n e comprimento k são gerados de L_{k-1}^{n-1} *e* L_{k-1}^{n} *- Veja seção 6.2.1 para detalhes)*
    4.6.2 $C_k^n = (L_{k-1}^{n-1} \bowtie L_{k-1}^{n-1}) \cup (L_{k-1}^{n} \bowtie L_{k-1}^{n}) \cup (L_{k-1}^{n-1} \bowtie L_{k-1}^{n}) \cup (L_{k-1}^{n} \bowtie L_{k-1}^{n-1})$;
    4.6.3 delete todos os candidatos $\sigma \in C_k^n$ tal que $\exists \tau \subseteq \sigma$, $\tau \in \Sigma_{k-1}^{n-1}$ e $\tau \notin L_{k-1}^{n-1}$; (poda 1)
    4.6.4 delete todos os candidatos $\sigma \in C_k^n$ tal que $\exists \tau \subseteq \sigma$, $\tau \in \Sigma_{k-1}^{n}$ e $\tau \notin L_{k-1}^{n}$; (poda 2)
    4.6.5 Foreach grupo g em D **do**
      Incremente o contador (cont) de todos os *psm's* candidatos em C_k^n contidos em $S(g)$;
    4.6.6 L_n^n = candidatos em C_k^n com *cont* $\geq \alpha N$;
  Until $L_k^n = \emptyset$;
Until $L_n^n = \emptyset$

Figura 6.2: Algoritmo SM

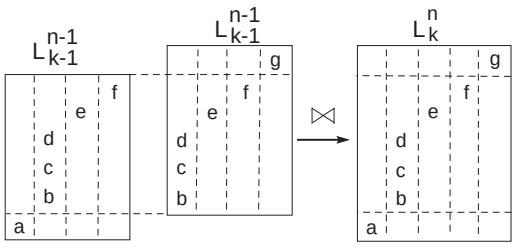
A seguinte notação será utilizada a seguir: se $\sigma \in \Sigma_k^n$ então $\overline{\sigma}$ e $\underline{\sigma}$ denotam as seqüências múltiplas (não necessariamente *psm's*) obtidas removendo-se, respectivamente, a linha n e a linha 1 de σ, como ilustrado na figura 6.3.

$$\sigma = \begin{pmatrix} \perp & \perp & d \\ \perp & c & \perp \\ b & \perp & \perp \\ a & \perp & \perp \end{pmatrix} \qquad \overline{\sigma} = \begin{pmatrix} \perp & c & \perp \\ b & \perp & \perp \\ a & \perp & \perp \end{pmatrix} \qquad \underline{\sigma} = \begin{pmatrix} \perp & \perp & d \\ \perp & c & \perp \\ b & \perp & \perp \end{pmatrix}$$

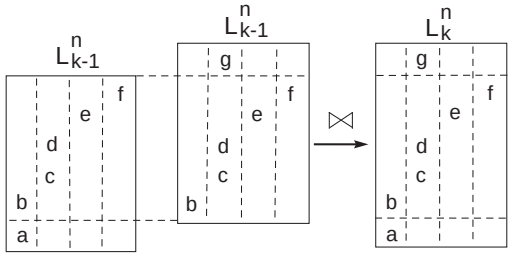
Figura 6.3: Seqüências múltiplas $\overline{\sigma}$ e $\underline{\sigma}$

Observando o passo **4.6.2** do algoritmo SM, podemos ver que existem quatro possibilidades para obter um candidado em C_k^n: (1) combinando dois *psm's* de L_{k-1}^{n-1}, (2)

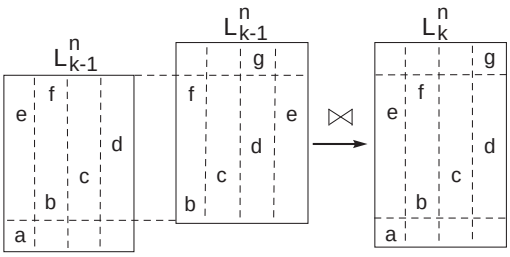
combinando dois *psm's* de L^n_{k-1}, (3) combinando um *psm* de L^{n-1}_{k-1} com um *psm* de L^n_{k-1} e (4) combinando um *psm* de L^n_{k-1} com um de L^{n-1}_{k-1}. A figura 6.4 de (a) até (e) ilustra as condições para que seja realizada a junção entre dois *psm's* (abaixo de cada figura) assim como o resultado da operação de junção, nos quatro casos.



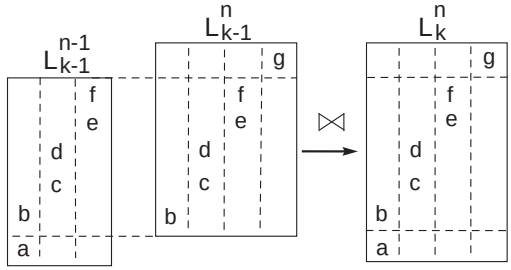
Caso 1: se eliminamos a primeira coluna de $\underline{\sigma}$ e a última coluna de $\overline{\tau}$ então os *psm's* resultantes são idênticos.



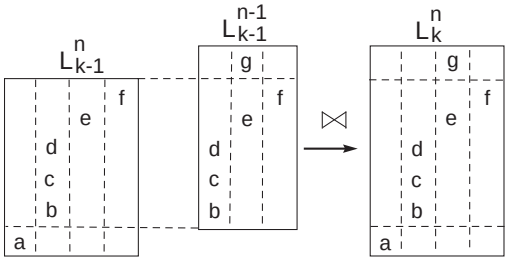
Caso 2a: $\underline{\sigma}$ e $\overline{\tau}$ são idênticos.



Caso 2b: $\underline{\sigma}$ e $\overline{\tau}$ são idênticos após "girar" as colunas de $\overline{\tau}$ uma posição à direita.



Caso 3: $\underline{\sigma}$ e $\overline{\tau}$ são idênticos e a última coluna de $\overline{\tau}$ contém somente o símbolo $\perp$.



Caso 4: A primeira coluna de $\underline{\sigma}$ contém apenas o elemento $\perp$ e $\underline{\sigma}$ é idêntico a $\overline{\tau}$ após "girar" $\underline{\sigma}$ uma posição à esquerda.

Figura 6.4: Realizando a junção de *psm's*

O seguinte teorema garante que: (1) todos os *psm's* freqüentes de Σ^n_k estão contidos em C^n_k e (2) C^n_k é mínimo, no sentido de que contém somente os *psm's* potencialmente freqüentes de Σ^n_k.

Teorema 6.2.1 Para todo n, k, $1 \leq n \leq k$, temos: (1) $L_k^n \subseteq C_k^n$; (2) se $\sigma \in \Sigma_k^n$ e existe $\tau \subseteq \sigma$, $\tau \neq \sigma$, tal que τ não é freqüente, então $\sigma \notin C_k^n$.

A prova deste teorema é dada no Apêndice desta dissertação.

A proposição dada a seguir explica as condições de parada do algoritmo.

Proposição 6.2.1 Seja $k \geq n$. Se $L_k^n = \emptyset$ then $L_p^n = \emptyset$ para todo $p \geq k$.

Prova: Suponha que exista $\sigma \in L_{k+1}^n$. Como $k + 1 > n$, podemos garantir que existe uma coluna i em σ onde pelo menos duas linhas contêm elementos diferentes de $\perp$. Se eliminamos uma das duas linhas contendo elementos diferentes de $\perp$, obtemos um *psm* em Σ_k^n. Pela proposição 4.1.1, este *psm* deve estar em L_k^n. Mas $L_k^n = \emptyset$ e temos então uma contradição.

6.2.2 Cálculo do Suporte

Para reduzir o número de operações de E/S e calcular o suporte de cada *psm* em C_k^n percorrendo o banco de dados apenas uma vez a cada iteração, o algoritmo SM encontra, para cada seqüência múltipla $S(g)$ do banco de dados, todos os candidatos σ que estão contidos em $S(g)$. Afim de reduzir o número de candidatos a serem verificados no teste de inclusão para cada seqüência múltipla do banco de dados, usamos uma técnica similar àquela descrita em [1] para a contagem do suporte de seqüências. Os *psm's* candidatos são armazenados em um **conjunto** de *árvores-hash*.

Mostramos no capítulo 5 que um *psm* σ pode ser completamente caracterizado por duas seqüências simples: a *seqüência de itens* de σ ($\Pi_i(\sigma)$) e a *seqüência de formato* ou simplesmente *formato* de σ ($\Pi_s(\sigma)$). Assim, podemos particionar o conjunto C_k^n em m subconjuntos disjuntos de *psm's*, onde cada subconjunto corresponde a um formato diferente dos *psm's* em C_k^n. Para cada um desses subconjuntos, podemos então tratar seus elementos como seqüências simples. Por exemplo, se consideramos o conjunto de candidatos C_3^2 ilustrados na tabela 6.1, temos uma partição com três elementos, correspondendo respectivamente aos formatos $< 1, 2, 2 >, < 1, 2, 1 >$ e $< 1, 1, 2 >$.

Dessa forma, podemos construir uma *árvore-hash* para cada subconjunto da partição, usando a técnica descrita em [1]. Nessa técnica, as seqüências são armazenadas nas folhas

C_3^2

$\begin{pmatrix} \perp & b \\ \perp & b \\ a & \perp \end{pmatrix}$	$\begin{pmatrix} d & \perp \\ \perp & c \\ a & \perp \end{pmatrix}, \begin{pmatrix} d & \perp \\ \perp & c \\ b & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & e \\ d & \perp \\ b & \perp \end{pmatrix} \begin{pmatrix} \perp & c \\ b & \perp \\ a & \perp \end{pmatrix}$

Tabela 6.1: Conjunto C_3^2

da árvore e os nós interiores contêm uma tabela *hash*, onde cada campo aponta para outro nó. A raiz da *árvore hash* possui profundidade 1. Um nó interior com profundidade d aponta para nós com profundidade $d+1$. Para adicionar uma seqüência s, descemos pela árvore (começando pela raiz) até chegarmos em uma folha. Em um nó interior com profundidade d, decidimos em qual ramo seguir aplicando a função *hash* para o d-ésimo elemento de s. Todos os nós são inicialmente criados como nós *folha*. Quando o número de seqüências em um nó *folha* excede um máximo especificado, o nó *folha* é convertido em um nó interior.

Como encontrar candidatos suportados por uma seqüência múltipla. Dada uma seqüência múltipla σ do banco de dados, para cada formato $\rho = < \rho_1, \rho_2, ..., \rho_k >$ de C_k^n, e para cada escolha de n colunas ordenadas em σ, procedemos exatamente como descrito em [1] para encontrar os candidatos C_k^n contidos nas n colunas escolhidas. A função *hash* vai sendo aplicada aos elementos de σ diferentes de $\perp$, na ordem natural definida pelo eixo dos tempos e de acordo com o formato ρ. Por exemplo, considere o formato $\rho =< 1, 1, 2 >$ e a seqüência múltipla do banco de dados $\sigma = \{< a, b, \perp, c, e, c > , < b, d, a, \perp, d, f >, < c, \perp, c, b, a, \perp >\}$, cuja representação em matriz é:

$$\begin{pmatrix} c & f & \perp \\ e & d & a \\ c & \perp & b \\ \perp & a & c \\ b & d & \perp \\ a & b & c \end{pmatrix}$$

Se estamos no caso onde escolhemos as duas primeiras colunas de σ então as sub-seqüências de σ que serão testadas seguindo a técnica de [1] terão o formato <1,1,2> e as seguintes seqüências de itens: $< a, b, a >, < a, b, d >, < a, b, f >, < a, c, d >, < a, c, f >, < a, e, f > , < b, c, d >, < b, c, f >, < b, e, f >, < c, e, f >$.

Assim, SM varre o banco de dados uma única vez a cada iteração na contagem do suporte das seqüências candidatas de C_k^n.

6.2.3 Um Exemplo

Considere o banco de dados de seqüências múltiplas ilustrado na figura 4.3 e um nível mínimo de suporte $sup\_min = 0{,}5$. Como tal banco de dados apresenta 4 grupos, um *psm* deve ser suportado por pelo menos 2 grupos para ser interessante. Os *psm's* interessantes de *rank* 1, obtidos aplicando-se o algoritmo GSP são:

- $L_1^1 = \{\{< a >\}, \{< b >\}, \{< c >\}\}$,
- $L_2^1 = \{\{< a, b >\}, \{< b, a >\}\}$
- $L_3^1 = \emptyset$.

Agora, apresentamos os conjuntos de *psm's* candidatos e freqüentes obtidos para o *rank* $n > 1$. Vale lembrar que para obter os candidatos de C_k^n, devemos gerar quatro subconjunto de candidatos: (1) $L_{k-1}^{n-1} \bowtie L_{k-1}^{n-1}$, (2) $L_{k-1}^{n} \bowtie L_{k-1}^{n}$, (3) $L_{k-1}^{n-1} \bowtie L_{k-1}^{n}$ e (4) $L_{k-1}^{n} \bowtie L_{k-1}^{n-1}$. Neste exemplo, mostramos apenas os subconjuntos de candidatos que não são vazios. Os *psm's* em C_k^n que são eliminados na fase da poda de SM são representados entre colchetes. Por exemplo, o *psm* $\{< c, \perp, \perp >, <\perp, a, b >\}$ é podado de C_3^2, pois contém o *psm* $\{< c, \perp >, <\perp, b >\}$ que não está em L_2^2.

Rank 2, Comprimento 2:

$C_2^2 = L_1^1 \bowtie L_1^1$	L_2^2
$\begin{pmatrix} \perp & a \\ a & \perp \end{pmatrix}, \begin{pmatrix} \perp & b \\ a & \perp \end{pmatrix},$	$\begin{pmatrix} \perp & a \\ a & \perp \end{pmatrix}, \begin{pmatrix} \perp & b \\ a & \perp \end{pmatrix},$
$\begin{pmatrix} \perp & c \\ a & \perp \end{pmatrix}, \begin{pmatrix} \perp & a \\ b & \perp \end{pmatrix},$	$\begin{pmatrix} \perp & a \\ b & \perp \end{pmatrix}, \begin{pmatrix} \perp & b \\ b & \perp \end{pmatrix},$
$\begin{pmatrix} \perp & b \\ b & \perp \end{pmatrix}, \begin{pmatrix} \perp & c \\ b & \perp \end{pmatrix},$	$\begin{pmatrix} \perp & a \\ c & \perp \end{pmatrix}$
$\begin{pmatrix} \perp & a \\ c & \perp \end{pmatrix}, \begin{pmatrix} \perp & b \\ c & \perp \end{pmatrix},$	
$\begin{pmatrix} \perp & c \\ c & \perp \end{pmatrix}$	

Rank 2, Comprimento 3:

C_3^2			L_3^2
$L_2^2 \bowtie L_2^2$	$L_2^1 \bowtie L_2^2$	$L_2^2 \bowtie L_2^1$	
$\begin{bmatrix} \boldsymbol{a} & \boldsymbol{\perp} \\ \boldsymbol{\perp} & \boldsymbol{a} \\ \boldsymbol{a} & \boldsymbol{\perp} \end{bmatrix}, \begin{pmatrix} b & \perp \\ \perp & a \\ a & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & a \\ b & \perp \\ a & \perp \end{pmatrix}, \begin{pmatrix} \perp & b \\ b & \perp \\ a & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & b \\ \perp & a \\ a & \perp \end{pmatrix}, \begin{pmatrix} \perp & a \\ \perp & b \\ a & \perp \end{pmatrix}$	$\begin{pmatrix} b & \perp \\ \perp & b \\ a & \perp \end{pmatrix}, \begin{pmatrix} a & \perp \\ \perp & b \\ b & \perp \end{pmatrix}$
$\begin{bmatrix} \boldsymbol{a} & \boldsymbol{\perp} \\ \boldsymbol{\perp} & \boldsymbol{b} \\ \boldsymbol{a} & \boldsymbol{\perp} \end{bmatrix}, \begin{pmatrix} b & \perp \\ \perp & b \\ a & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & a \\ a & \perp \\ b & \perp \end{pmatrix}, \begin{pmatrix} \perp & b \\ a & \perp \\ b & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & b \\ \perp & a \\ b & \perp \end{pmatrix}, \begin{pmatrix} \perp & a \\ \perp & b \\ b & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & a \\ b & \perp \\ a & \perp \end{pmatrix}, \begin{pmatrix} \perp & a \\ \perp & b \\ a & \perp \end{pmatrix}$
$\begin{pmatrix} a & \perp \\ \perp & a \\ b & \perp \end{pmatrix}, \begin{bmatrix} \boldsymbol{b} & \boldsymbol{\perp} \\ \boldsymbol{\perp} & \boldsymbol{a} \\ \boldsymbol{b} & \boldsymbol{\perp} \end{bmatrix}$		$\begin{bmatrix} \boldsymbol{\perp} & \boldsymbol{b} \\ \boldsymbol{\perp} & \boldsymbol{a} \\ \boldsymbol{c} & \boldsymbol{\perp} \end{bmatrix}$	
$\begin{pmatrix} a & \perp \\ \perp & b \\ b & \perp \end{pmatrix}, \begin{bmatrix} \boldsymbol{b} & \boldsymbol{\perp} \\ \boldsymbol{\perp} & \boldsymbol{b} \\ \boldsymbol{b} & \boldsymbol{\perp} \end{bmatrix}$			
$\begin{bmatrix} \boldsymbol{a} & \boldsymbol{\perp} \\ \boldsymbol{\perp} & \boldsymbol{a} \\ \boldsymbol{c} & \boldsymbol{\perp} \end{bmatrix}, \begin{bmatrix} \boldsymbol{b} & \boldsymbol{\perp} \\ \boldsymbol{\perp} & \boldsymbol{a} \\ \boldsymbol{c} & \boldsymbol{\perp} \end{bmatrix}$			

Rank 2, Comprimento 4:

$C_4^2 = L_3^2 \bowtie L_3^2$	L_4^2
$\begin{pmatrix} \perp & a \\ b & \perp \\ \perp & b \\ a & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & a \\ b & \perp \\ \perp & b \\ a & \perp \end{pmatrix}$

Rank 2, Comprimento 5: nenhuma seqüência múltipla pode ser gerada para C_5^2. Logo, $L_5^2 = \emptyset$ e o algoritmo não procede para o rank 2 (de acordo com a proposição 6.2.1).

$C_5^2 = L_4^2 \bowtie L_4^2$	L_5^2
$\emptyset$	$\emptyset$

Rank 3, Comprimento 3:

$C_3^3 = L_2^2 \bowtie L_2^2$	L_3^3
$\left(\begin{array}{ccc} \perp & \perp & a \\ \perp & a & \perp \\ a & \perp & \perp \end{array}\right), \left(\begin{array}{ccc} \perp & \perp & b \\ \perp & a & \perp \\ a & \perp & \perp \end{array}\right)$	$\left(\begin{array}{ccc} \perp & \perp & a \\ \perp & b & \perp \\ a & \perp & \perp \end{array}\right)$
$\left(\begin{array}{ccc} \perp & \perp & a \\ \perp & b & \perp \\ a & \perp & \perp \end{array}\right), \left(\begin{array}{ccc} \perp & \perp & b \\ \perp & b & \perp \\ a & \perp & \perp \end{array}\right)$	
$\left(\begin{array}{ccc} \perp & \perp & a \\ \perp & a & \perp \\ b & \perp & \perp \end{array}\right), \left(\begin{array}{ccc} \perp & \perp & b \\ \perp & a & \perp \\ b & \perp & \perp \end{array}\right)$	
$\left(\begin{array}{ccc} \perp & \perp & a \\ \perp & b & \perp \\ b & \perp & \perp \end{array}\right), \left(\begin{array}{ccc} \perp & \perp & b \\ \perp & b & \perp \\ b & \perp & \perp \end{array}\right)$	
$\left(\begin{array}{ccc} \perp & \perp & a \\ \perp & a & \perp \\ c & \perp & \perp \end{array}\right), \left[\begin{array}{ccc} \perp & \perp & \boldsymbol{b} \\ \perp & \boldsymbol{a} & \perp \\ \boldsymbol{c} & \perp & \perp \end{array}\right]$	

Rank 3, Comprimento 4:

C_4^3				L_4^3
$L_3^2 \bowtie L_3^2$	$L_3^2 \bowtie L_3^3$	$L_3^3 \bowtie L_3^2$	$L_3^3 \bowtie L_3^3$	
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Rank 4, Comprimento 4: nenhuma seqüência múltipla é gerada. Logo, $L_4^4 = \emptyset$ e o algoritmo pára, pois todos os conjuntos de seqüências múltiplas de comprimento e/ou rank maior também serão vazios (veja proposição 6.2.1).

$C_4^4 = L_3^3 \bowtie L_3^3$	L_4^4
$\emptyset$	$\emptyset$

O algoritmo SM foi testado em vários bancos de dados sintéticos, assim como o algoritmo PM apresentado no capítulo anterior. No capítulo seguinte, apresentaremos os resultados obtidos com tais experimentos, realizando uma análise comparativa entre os dois algoritmos.

Capítulo 7

Resultados Experimentais

Para avaliar a performance dos algoritmos PM e SM, realizamos vários experimentos usando banco de dados sintéticos. Os experimentos foram realizados em um Pentium 4 de 2.4 GHz com 1 GB de memória RAM executando o sistema operacional Windows XP Professional.

7.1 Dados Sintéticos

Desenvolvemos um gerador de dados sintéticos baseado na idéia descrita em [1] para seqüências simples. Nosso gerador produz bancos de dados de seqüências múltiplas de acordo com os parâmetros de entrada, que são mostrados na tabela 7.1.

Na geração dos bancos de dados sintéticos, configuramos o número de itens (N) para 3000 e o número de padrões maximais potencialmente freqüentes (N_m) para 1000. O rank médio dos padrões potencialmente freqüentes, $|R|$, é ajustado para 3 e o comprimento médios desses padrões, $|S|$, é ajustado para 4. Os valores de todos os parâmetros utilizados na geração dos bancos de dados sintéticos são dados na tabela 7.1. O banco de dados $D2-G4-C6-N3$, por exemplo, mantém 2000 grupos com uma média de 4 clientes por grupo e 6 transações por cliente e contém 3000 itens. As quatro últimas linhas da tabela 7.2 informam os valores dos parâmetros para os bancos de dados utilizados nos testes de escalabilidade. Na última linha, por exemplo, a configuração D4-G4-C3-Nx representa vários bancos de dados onde apenas o parâmetro *número de itens* (Nx) varia de um banco

para outro.

$\|D\|$	Número de grupos (tamanho do banco de dados) - em milhares
$\|G\|$	Número médio de clientes por grupo
$\|C\|$	Número médio de transações por cliente
$\|R\|$	Rank médio dos *psm's* potencialmente freqüentes
$\|S\|$	Comprimento médio dos *psm's* potencialmente freqüentes
N	Número de itens
N_m	Número de *psm's* maximais potencialmente freqüentes

Tabela 7.1: Parâmetros usados no Gerador de Dados Sintéticos

Nome	$\|D\|$	$\|G\|$	$\|C\|$	$\|R\|$	$\|S\|$	N	N_m	**Tamanho (MB)**
D2-G4-C3-N3	2	4	3	3	4	3000	1000	1,06
D2-G4-C6-N3	2	4	6	3	4	3000	1000	2,09
D4-G4-C6-N3	4	4	6	3	4	3000	1000	4,19
D2-G6-C3-N3	2	6	3	3	4	3000	1000	1,57
Dx-G3-C3-N3	1-5	3	3	3	4	3000	1000	0,5 - 5,4
D2-Gx-C3-N3	2	3-8	3	3	4	3000	1000	0,8 - 2,1
D4-G3-Cx-N3	4	3	2-10	3	4	3000	1000	1,2 - 5,3
D4-G4-C3-Nx	4	4	3	3	4	250-7500	1000	2,1

Tabela 7.2: Bancos de Dados Sintéticos - Parâmetros Utilizados

7.2 Análise Comparativa de Performance

A figura 7.1 ilustra os tempos de execução dos algoritmos PM e SM para os quatro primeiros bancos da tabela 7.2 à medida que o suporte mínimo é decrementado de 1% a 0.25%. Como esperado, os tempos de execução de ambos algoritmos aumentam. Observe que SM executa consideravelmente melhor que PM nos casos em que o suporte mínimo é baixo, principalmente para o bancos de dados D4-G4-C6-N3, que é o maior. Isto se explica pelo fato de que PM gera a cada iteração um número maior de padrões candidatos do que são gerados em SM. PM primeiramente gera as seqüências de itens e os formatos dos *psm's* e então os combina para a formação dos padrões completos. Nesse processo, o número de padrões candidatos obtidos é maior do que em SM, onde a geração e a poda dos padrões é realizada sem que o padrão seja decomposto. Os dois algoritmos apresentam uma performance similar para valores mais altos do suporte mínimo. O motivo é que,

neste caso, a maior parte dos padrões minerados possui comprimento e rank pequenos (o algoritmo raramente passa da iteração (3,3)) e a maior parte do tempo de execução é gasta na primeira iteração, que é realizada da mesma forma para os dois algoritmos.

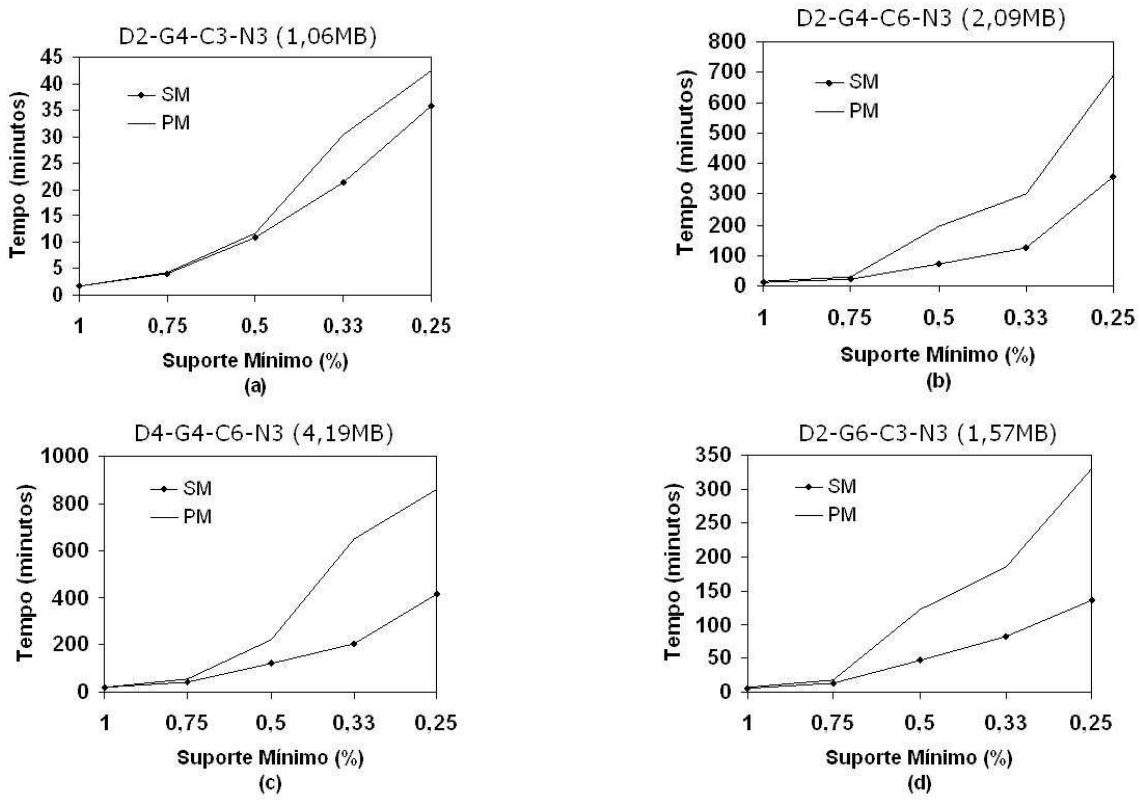


Figura 7.1: Tempos de Execução para Variação do Suporte - SM x PM

7.3 Análise Comparativa de Escalabilidade

Nesta seção apresentamos os resultados dos principais experimentos realizados para análise da escalabilidade dos algoritmos PM e SM. Também executamos outros testes e resultados similares foram obtidos.

A figura 7.2(a) mostra como os tempos de execução de SM e PM aumentam à medida que o número de seqüências múltiplas do banco de dados ($|D|$) é incrementado de 1000 a 5000. Os resultados foram obtidos para os bancos de dados Dx-G3-C3-N3 ($x = 1, \ldots, 5$) com o suporte mínimo ajustado para 0,15%. O gráfico mostra que o tempo de execução de SM aumenta de forma mais linear que o tempo de execução de PM à medida que

aumentamos o número de grupos (ou tamanho) do banco de dados.

A figura 7.2(b) ilustra como o tempo de execução dos algoritmos aumenta a medida em que o número de clientes por grupo ($|G|$) é incrementado de 3 a 8. Os bancos de dados utilizados foram D2-Gx-C3-N3 ($x = 3, \ldots, 8$) com o suporte mínimo ajustado para 0,25%. Podemos ver que o tempo de execução de PM aumenta de forma marcante à medida que o número de clientes por grupo é incrementado, ao contrário do algoritmo SM, cujo tempo de execução aumenta suavemente. Esse comportamento também é explicado pelo fato de que PM gera mais candidatos do que SM a cada iteração e o tempo de execução do cálculo do suporte aumenta com relação ao número de clientes por grupo. Dessa forma, é de se esperar que a performance de PM seja inferior, pois o número de candidatos testados por PM a cada iteração é muito maior que o número de candidatos testados por SM.

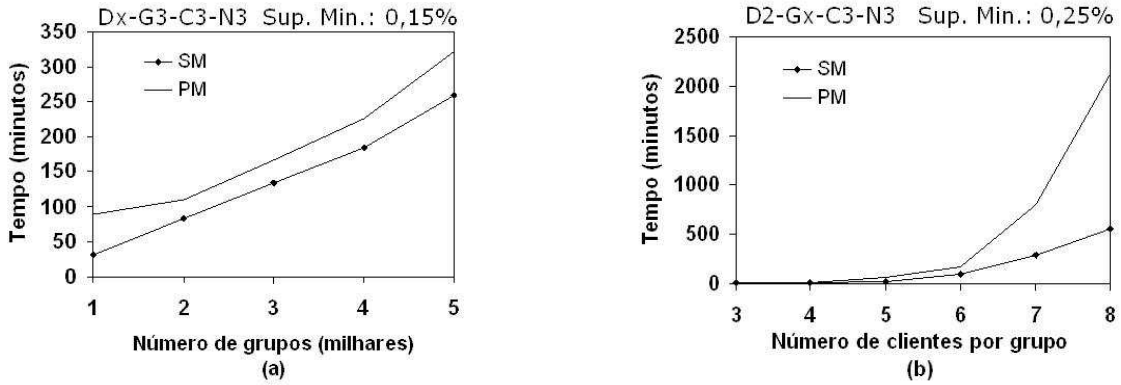


Figura 7.2: Testes de Escalabilidade: Nº de Grupos e Clientes por Grupo

Também realizamos experimentos variando o número de transações por cliente e o número de itens dos bancos de dados. A figura 7.3(a) ilustra os tempos de execução dos algoritmos a medida que aumentamos o número de transações por cliente de 2 para 10. Os bancos de dados utilizados foram D4-G3-Cx-N3 ($x = 2, 4, \ldots, 10$) com o suporte mínimo ajustado para 0.25%. Também neste caso, a curva do tempo de execução do algoritmo PM acentua-se mais rapidamente.

A figura 7.3(b) ilustra a performance de SM e PM quando o número de itens é incrementado de 250 a 7500. O suporte mínimo é ajustado para 0,5%. A performance dos dois algoritmos é praticamente similar, sendo que SM executa ligeiramente mais rápido que PM.

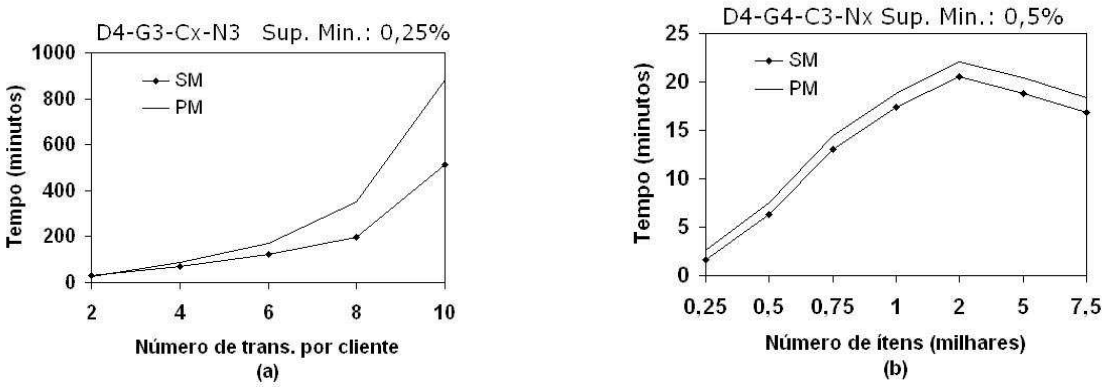


Figura 7.3: Testes de Escalabilidade: Transações por Cliente e Número de Itens

Parte III

Mineração de Padrões Seqüenciais Múltiplos com Restrições

Capítulo 8

O Problema de Mineração de *PSM's* com Restrições

Os algoritmos propostos nos capítulos anteriores foram desenvolvidos para encontrar **todos** os padrões seqüenciais múltiplos freqüentes em um banco de dados de transações. Dependendo do valor definido para o suporte mínimo, a quantidade de padrões minerados pode ser imensa e muitos deles podem ser completamente irrelevantes para o usuário. Neste capítulo, especificamos um formalismo para a mineração de *psm's* que fornece ao usuário um controle no processo de mineração afim de evitar a geração de padrões não interessantes.

8.1 Restrições Expressas por Expressões Regulares

Nos últimos anos, algumas pesquisas foram realizadas sobre o problema de introduzir restrições especificadas pelo usuário no processo de mineração. Tais restrições normalmente são consideradas ou durante uma fase de *pós processamento* [44, 45], ou são incorporadas no *processo de mineração* propriamente dito. A última forma tem sido bastante explorada no contexto da mineração de regras de associação [46, 47], assim como na mineração de padrões seqüenciais [48, 33]. Nesta dissertação, vamos explorar o problema de mineração de *psm's* incorporando uma restrição especificada pelo usuário dentro do processo de mineração, em vez de considerá-la em uma etapa de pós processamento.

A técnica que utilizamos segue a idéia introduzida em [33] (descrita no capítulo 3, seção 3.2.2 desta dissertação), onde é proposta uma especificação para a consideração de restrições baseada em expressões regulares no contexto da mineração de padrões seqüenciais (simples). Como mencionado em [33], as expressões regulares fornecem uma sintaxe simples e natural para especificação de classes especiais de padrões seqüenciais. Aqui, estendemos o formalismo para consideração de restrições expressas por expressões regulares no contexto da mineração de seqüências múltiplas.

8.2 Expressões Regulares sobre *PSM's*

No capítulo 5, mostramos que um *psm* σ pode ser completamente caracterizado por duas seqüências simples: a *seqüência de itens* de σ ($\Pi_i(\sigma)$) e a *seqüência de formato* ou simplesmente *formato* de σ ($\Pi_s(\sigma)$). Dessa forma, para possibilitar a seleção de classes especiais de *psm's*, decidimos considerar um par de restrições expressas por expressões regulares, uma para a seqüência de itens e outra para a seqüência de formato. Assim, a expressão regular considerada nas seqüências de formato dos *psm's* terá a função de restringir os "formatos" dos *psm's* que serão minerados, enquanto a expressão regular considerada na seqüência de itens exercerá influência na seleção dos itens e na ordem em que aparecem nos *psm's*. O exemplo a seguir ilustra essa idéia.

Exemplo 8.2.1 Considere novamente o exemplo da *cotação de ações* introduzido no capítulo 1 (exemplo 1.1.1) e suponha que estejamos interessados somente em padrões seqüenciais múltiplos da forma: *um repetido aumento de x é seguido por um aumento de y*. Esta condição pode ser especificada pelo seguinte par de expressões regulares: (*sobe sobe sobe* sobe, 111*2*).

A seguir, daremos a definição de uma expressão regular especial, que será aquela utilizada nas seqüências de formato dos *psm's*. Antes disso, gostaríamos de relembrar a forma especial das seqüências de formato, existente devido a ordem das múltiplas seqüências no *psm*: se $j \in \mathbb{N}$ aparece pela primeira vez na posição i do formato, então nenhum $k > j$ pode aparecer antes de i. Por exemplo, <1,2,1,3,1> é uma seqüência de formato, mas <1,**3**,2,1,1> não é.

Definição 8.2.1 Uma expressão regular e sobre o alfabeto $\mathbb{N}$ é chamada de *expressão regular de formato* se $e = 1e_1 2e_2 ... ne_n$, para algum $n \in \mathbb{N}$, e para cada $i = 1, ..., n$, e_i é uma expressão regular sobre $\{1, ..., i\}$. O número n é chamado de *rank* de e e é denotado por $r(e)$. A m-subexpressão de e, para $m \leq n$, é a expressão regular $e' = 1e_1 2e_2 ... me_m$.

Por exemplo, $\mathbf{12}1^*\mathbf{3}(1+2)^*\mathbf{4}$ é uma expressão regular de formato (rank 4), onde $e_1 = \epsilon$ (representamos uma expressão vazia por ϵ), $e_2 = 1^*$, $e_3 = (1+2)^*$ e $e_4 = \epsilon$. Entretando, 131^*2 não é uma expressão regular de formato, pois o elemento 3 aparece sem que o elemento 2 tenha aparecido antes. A 2-subexpressão de $121^*3(1+2)^*4$ é 121^*.

Definição 8.2.2 Uma *restrição-PSM* é um par $[R_i, R_s]$ onde R_i é uma expressão regular sobre o alfabeto $\mathcal{I}$ (o conjunto de itens) e R_s é uma expressão regular de formato de rank n. Um *psm* σ com $r(\sigma) \leq r(R_s)$ *satisfaz* uma restrição-PSM $[R_i, R_s]$ se $\Pi_i(\sigma)$ satisfaz R_i e $\Pi_s(\sigma)$ satisfaz a m-subexpressão de R_s, onde $m = r(\sigma)$.

Por exemplo, $[ab^*, 122^*3(1+2)^*4]$ é uma restrição-PSM, onde $R_i = ab^*$ e $R_s = 122^*3(1+2)^*4$. Veja que σ = (<a,b,b>,<1,2,2>) satisfaz $[ab^*, 122^*3(1+2)^*4]$, pois <a,b,b> satisfaz ab^* e <1,2,2> satisfaz 122^* (2-subexpressão de $122^*3(1+2)^*4$).

Dizemos que um *psm* σ é *interessante* com relação a um banco de dados $\mathcal{D}$, um suporte mínimo α e uma restrição-PSM $[R_i, R_s]$ se σ é freqüente com relação a $\mathcal{D}$ e α e σ satisfaz $[R_i, R_s]$.

Agora que já definimos uma restrição de expressões regulares para seqüências múltiplas, formalizamos o problema de mineração de *psm's* com restrições.

Formulação do Problema. Dado um banco de dados $\mathcal{D}$, um nível mínimo de suporte α e uma restrição-PSM $[R_i, R_s]$, encontrar todos os *psm's interessantes* com relação a $\mathcal{D}$, α e $[R_i, R_s]$.

No capítulo seguinte, apresentamos o algoritmo *MSP-Miner* para solução do problema formulado acima. *MSP-Miner* foi desenvolvido com base na idéia introduzida em [33] (ver capítulo 3, seção 3.2.2 desta dissertação) para minerar todos os *psm's* interessantes em um banco de dados $\mathcal{D}$, com relação a um suporte mínimo α e uma restrição $[R_i, R_s]$.

Capítulo 9

O Algoritmo *MSP-Miner*

Neste capítulo apresentamos o algoritmo *MSP-Miner* (***M**ulti-**S**equence **P**attern Miner*) para mineração de *psm's* satisfazendo uma restrição-PSM $R = [R_i, R_s]$.

9.1 Idéia Geral

Assim como os algoritmos PM e SM, *MSP-Miner* é um algoritmo que utiliza iteração dupla baseado na idéia de *Apriori*: na iteração (n, k), os *psm's* candidatos potencialmente interessantes de rank n e comprimento k são gerados, em seguida os candidatos que não podem ser freqüentes são podados e por fim, aqueles restantes são testados na fase da contagem do suporte. Em *MSP-Miner*, a fase de geração de candidatos incorpora uma restrição-PSM R', que é um "relaxamento" de R, ou seja, R' é uma restrição mais fraca que R. Na iteração (n, k), os *psm's* candidatos de rank n e comprimento k são gerados de anteriores L^n_{k-1} e L^{n-1}_{k-1} de forma a continuar satisfazendo R'. Em uma etapa de pós-processamento, *MSP-Miner* elimina os *psm's* não satisfazendo a restrição original R. A escolha por incorporar o relaxamento R' de R e não a própria restrição R no processo de mineração é justificada na seção seguinte.

De uma forma geral, *MSP-Miner* obtém o conjunto L^n_k de maneira semelhante aos algoritmos SM e PM (no sentido de que são utilizados os conjuntos já obtidos anteriormente L^n_{k-1} e L^{n-1}_{k-1}). A figura 9.1 a seguir ilustra esse processo. A região em destaque corresponde à união de todos os conjuntos gerados até a iteração (3,4) e é denotada por

$\mathcal{L}_{3,4}$. Para cada n, k, denotamos por $\mathcal{L}_{n,k}$ o conjunto formado pela união de todos os conjuntos anteriores gerados até a iteração (n,k), ou seja, $\mathcal{L}_{n,k} = \bigcup_{p=1}^{n-1} \bigcup_{q=n}^{k-1} L_q^p \cup \bigcup_{p=1}^{n-1} L_k^p$.

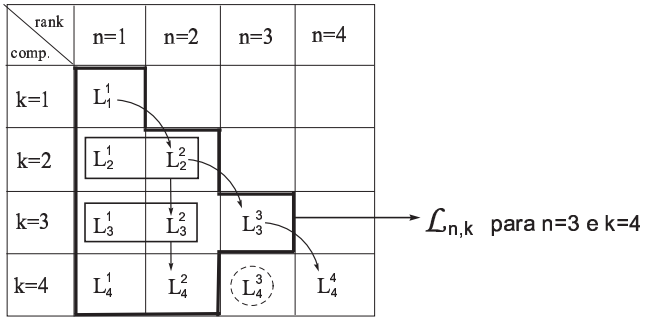


Figura 9.1: L_k^n é obtido de L_{k-1}^n e L_{k-1}^{n-1}

9.2 A Restrição R e a Fase da Poda

Considere, por um momento, que a restrição original R fosse considerada na fase de geração de candidatos do algoritmo *MSP-Miner*. Pela propriedade de *antimonotonia* de *psm's* freqüentes (proposição 4.1.1), podemos eliminar do conjunto de candidatos aqueles *psm's* que possuem pelo menos um *sub-psm* não freqüente. Ao mesmo tempo, pelo fato da restrição R não ser *antimonotônica*, não podemos eliminar *psm's* contendo um *sub-psm* que não satisfaz R (fazendo isto, poderíamos eliminar *psm's* interessantes). Conseqüentemente, para a realização da poda, devemos nos concentrar somente em *psm's* contendo *sub-psm's* não freqüentes.

Para que um *psm* candidato σ de C_k^n seja podado, devemos encontrar algum *sub-psm* $\sigma' \subset \sigma$ que não seja freqüente. Sabemos que um *sub-psm* $\sigma' \notin \mathcal{L}_{n,k}$ se e somente se: (1) σ' não é freqüente ou (2) σ' não satisfaz a restrição R. Assim, para ter certeza de que σ não é freqüente, é suficiente verificar que σ' **satisfaz** R e $\sigma' \notin \mathcal{L}_{n,k}$.

Notamos que, ao mesmo tempo que uma grande seletividade da restrição R diminui o número de candidatos gerados, ela tem um efeito negativo sobre a quantidade de *psm's* que são podados, isto é, a medida que a seletividade de R aumenta, a quantidade de *psm's* podados diminui, como ilustrado na figura 9.2 a seguir.

Seletividade de R	Candidatos Gerados	Candidatos Podados
⇑	⇓	⇓

Figura 9.2: Efeito negativo da restrição R para a fase da poda

Para encontrar um meio termo entre a seletividade de R e a quantidade de *psm's* podados, aplicamos uma idéia similar àquela introduzida em [33] e que já discutimos no capítulo 3, seção 3.2.2, ou seja, consideramos uma restrição *mais fraca* R' que é menos restritiva que R mas ao mesmo tempo suficientemente forte para eliminar muitos candidados na fase de geração de candidatos. Dessa forma, o processo de mineração produz um conjunto de *psm's* que são interessantes com relação a R'. Em uma fase de pós-processamento os *psm's* não satisfazendo R são eliminados.

No capítulo 3, descrevemos os quatro relaxamentos para uma dada restrição considerados em [33]. Eles são relembrados aqui: (1) o relaxamento *total*, onde nenhuma restrição é incorporada na fase de geração de candidatos (a restrição R é considerada somente na fase de pós-processamento); (2) o relaxamento *sufixo-válido*, onde somente seqüências *válidas* com relação a um estado do autômato (correspondente à dada expressão regular R) são consideradas (veja seção 3.2.2, tópico *Os quatro algoritmos SPIRIT*); (3) o relaxamento *legal*, onde somente seqüências *legais* com relação a um estado do autômato são consideradas; (4) nenhum relaxamento: somente as seqüências satisfazendo o autômato são consideradas, isto é, a restrição R é totalmente considerada na fase de geração de candidatos e nenhum pós-processamento é necessário. Os resultados experimentais apresentados em [33] nos permite concluir que o relaxamento sufixo-válido é responsável pela melhor performance. Baseado nesta idéia, consideramos um relaxamento similar para nossas restrições-PSM.

Definição 9.2.1 Seja $\mathcal{A}$ e s, um autômato e uma seqüência respectivamente sobre o alfabeto Σ. Dizemos que s é *prefixo-válida* com relação a $\mathcal{A}$ se existe um caminho em $\mathcal{A}$ começando no estado inicial e produzindo s (não necessariamente finalizando em um estado final).

Definição 9.2.2 Seja $R = [R_i, R_s]$ uma restrição-PSM e A_{R_i}, A_{R_s} os autômatos correspondentes a R_i e R_s respectivamente. Dizemos que um *psm* σ é *prefixo-válido* (ou sim-

plesmente *p-válido*) com relação a R se $\Pi_i(\sigma)$ e $\Pi_s(\sigma)$ são prefixo-válidas com relação a A_{R_i} e A_{R_s} respectivamente.

O algoritmo *MSP-Miner* pode ser descrito resumidamente como segue:

1. **Geração de Candidatos.** O conjunto C_k^n de *psm's* p-válidos é gerado a partir dos conjuntos de *psm's* p-válidos e freqüentes obtidos anteriormente L_{k-1}^n e L_{k-1}^{n-1} e dos autômatos correspondentes a R_i e R_s.

2. **Poda.** Os *psm's* em C_k^n contendo um *sub-psm* p-válido $\sigma \notin \mathcal{L}_{n,k}$ são podados.

3. **Cálculo do Suporte.** O suporte de cada candidato restante é avaliado fazendo-se uma varredura no banco de dados. São eliminados os *psm's* com suporte inferior a um dado suporte mínimo.

4. **Condição de Parada.** Para o rank 1, o algoritmo termina no estágio $(1, k)$, onde k é o primeiro comprimento para o qual $L_k^1 = \emptyset$. Para rank $n \geq 2$, o algoritmo finaliza no estágio (n, k) se uma das seguintes condições é verificada: (1) $L_p^{n-1} = \emptyset$ para todo $p \in \{n-1, ..., k\}$ ou (2) $L_{k-1}^{n-1} = L_{k-1}^n = \emptyset$ e $L_p^{n-1} = \emptyset$ para todo $p \geq k-1$.

5. **Pós-Processamento.** Todos os *psm's* p-válidos encontrados são testados para validação com relação à restrição original R.

O algoritmo *MSP-Miner* é descrito na figura 9.3. Agora, discutiremos os detalhes da fase de geração da iteração (n, k).

9.3 Geração de Candidatos

Os *psm's* p-válidos de rank n e comprimento k são gerados de acordo com os passos descritos a seguir:

1. Primeiramente focamos nas seqüências de formato e geramos todas aquelas de comprimento k sobre o alfabeto $\{1, ..., n\}$ que são p-válidas com respeito ao autômato A_{R_s} (passo 8.1). Seja $\mathcal{S}$ o conjunto de seqüências de formato geradas dessa forma.

Procedure *MSP-Miner*(α, D, $[R_i, R_s]$) // α: suporte mínimo, D: banco de dados
begin
1. N = número de multi-seqüências em D
2. A_{R_i}, A_{R_s} = autômatos correspondentes a R_i e R_s respectivamente
3. CI_1^1 = seqüências de itens de comprimento 1 p-válidas com relação ao autômato A_{R_i}
4. $CS_1^1 = \{< 1 >\}$ // a única seqüência de formato de rank 1 e comprimento 1
5. $C_1^1 = CI_1^1 \boxtimes CS_1^1$
6. L_1^1 = *psm's* freqüentes de C_1^1
7. $n = 1$; $k = 2$; existePSMsRankAtual = FALSE; kMaxIteração = 0
8. repeat
// fase de geração de candidatos
8.1 CS_k^n = seqüências de formato de rank n e comp. k p-válidas com relação ao autômato A_{R_s}
8.2 for each seqüência de formato s de CS_k^n **do**
8.2.1 s' = $(k-1)$-prefixo de s
8.2.2 x = rank(s') // o rank de s' pode ser n ou $n-1$
8.2.3 if $s' \in \Pi_s(L_{k-1}^x)$
8.2.3.1 $\mathcal{I}(s)$ = ExpandeSeqsItens(A_{R_i}, s, s',$x, k-1$, L_{k-1}^x)
8.2.3.2 $C_k^n = C_k^n \cup (\mathcal{I}(s) \boxtimes \{s\})$
end if
end for
// fase da poda
8.3 remova todos os cand. $\sigma \in C_k^n$ tal que $\exists \tau \subseteq \sigma$, τ é p-válido com resp. a $[R_i,R_s]$ e $\tau \notin \mathcal{L}_{n,k}$
// fase do cálculo do suporte
8.4 for each grupo g em D **do**
Incremente o contador de todos os *psm's* candidatos em C_k^n que estão contidos em $S(g)$
8.5 L_k^n = candidatos em C_k^n com contador $\geq \alpha N$
8.6 if $L_k^n \neq \emptyset$
kMaxIteração = k; existePSMsRankAtual = TRUE
$\mathcal{L}_{n,k} = \mathcal{L}_{n,k} \cup L_k^n$
8.7 $k = k + 1$
8.8 if ($L_{k-1}^{n-1} = \emptyset$ **and** $L_{k-1}^n = \emptyset$ **and** k > kMaxIteraçãoAnt)
8.8.1 $n = n + 1$; $k = n$ // incrementing the rank
8.8.2 kMaxIteraçãoAnt = kMaxIteração; kMaxIteração = 0
8.8.3 existePSMsRankAnt = existePSMsRankAtual; existePSMsRankAtual = FALSE
until not (existePSMsRankAnt **and** n = k)
// fase de pós-processamento
9. remova todos os *psm's* σ em $\mathcal{L}n, k$ que não satisfazem $[R_i,R_s]$
10. return $\mathcal{L}_{n,k}$
end

Figura 9.3: Algoritmo *MSP-Miner*

2. Para cada $s \in \mathcal{S}$, devemos gerar o conjunto das seqüências de itens $\mathcal{I}(s)$, de tal forma que os *psm's* correspondentes $\mathcal{I}(s) \boxtimes \{s\}$ sejam p-válidos e potencialmente

freqüentes (vale lembrar que a função $\boxtimes$ em $CI \boxtimes CS$ é utilizada para "montar" os *psm's*, ligando cada seqüência de formato de CS com todas as seqüências de itens de CI). Os passos necessários para a execução desta tarefa são listados a seguir:

(a) Para cada $s \in \mathcal{S}$, consideramos seu prefixo s' de comprimento $k-1$ removendo-se o último elemento de s. (passo 8.2.1). Note que $r(s') = n$ ou $r(s') = n-1$ (passo 8.2.2);

(b) No processo de geração das seqüências de itens, usamos o conjunto de *psm's* freqüentes e p-válidos gerado previamente L_{k-1}^{n} ou L_{k-1}^{n-1}, dependendo do rank de s' ser n ou $n-1$ respectivamente. (passo 8.2.3.1). O procedimento *ExpandeSeqsItens* (figura 9.4) é responsável por esta tarefa;

(c) Seleciona-se no conjunto adequado (L_{k-1}^{n} ou L_{k-1}^{n-1}) aqueles *psm's* correspondentes ao dado prefixo s' (passo 1. de *ExpandeSeqsItens*);

(d) Em seguida, projetamos esses *psm's* para obter suas seqüências de itens, que também são freqüentes e p-válidas com relação a A_{R_i}. Agora, essas seqüências de itens são *expandidas* de acordo com o autômato A_{R_i} (A "expansão" ficará mais clara no exemplo a seguir; para mais detalhes, veja o procedimento *ExpandeSeqItens* descrito na figura 9.4). Neste ponto, já obtivemos o conjunto $\mathcal{I}(s)$ de seqüências de comprimento k potencialmente freqüentes e p-válidas com relação ao autômato A_{R_i} (Passo 8.2.3.1). Assim, o conjunto de *psm's* gerados para a seqüência de formato s é $\mathcal{I}(s) \boxtimes \{s\}$ (Passo 8.2.3.2).

3. O conjunto inteiro C_k^n de *psm's* candidatos de rank n e comprimento k é $\bigcup_{s \in \mathcal{S}} (\mathcal{I}(s) \boxtimes \{s\})$ (Passo 8.2.3.2).

9.3.1 Um Exemplo

O exemplo a seguir ilustra a fase de geração de candidatos do algoritmo *MSP-Miner*.

Exemplo 9.3.1 Sejam A_{R_i} e A_{R_s} (mostrados nas figuras 9.6 e 9.5 respectivamente) os autômatos correspondentes às expressões regulares R_i e R_s respectivamente. Seja q um

Procedure ExpandeSeqsItens(A_{R_i}, seqüência de formato s, seqüência de formato s', n, k, L_k^n)
begin
1. $L_k'^n$ = *psm's* em L_k^n correspondentes à seqüência do formato s'
// $\Pi_i(L_k'^n)$ são seqüências de itens p-válidas com respeito a algum estado de A_{R_i}
3. $\mathcal{I}(s) = \emptyset$;
4. **for each** estado q de A_{R_i} **do**
4.1 V_q = conjunto de todas as transições terminando em q
4.2 **for each** transição t de V_q **do**
4.2.1 a = rótulo de t
4.2.2 **for each** seqüência de itens p-válida $u = < u_1, ..., u_k >$ de $\Pi_i(L_k'^n)$
com relação ao estado de origem de t **do**
4.2.2.1 novaSeq = $< u_1, u_2, ..., u_k, a >$
4.2.2.2 $\mathcal{I}(s) = \mathcal{I}(s) \cup \{\text{novaSeq}\}$
end para
end para
end para
4. **return** $\mathcal{I}(s)$
end

Figura 9.4: Algoritmo ExpandeSeqsItens

estado do autômato A_{Ri}. Denotamos por $L_k^n(q)$ o conjunto $\{\sigma \in L_k^n \mid \Pi_i(\sigma)$ é p-válida com respeito a $q\}$.

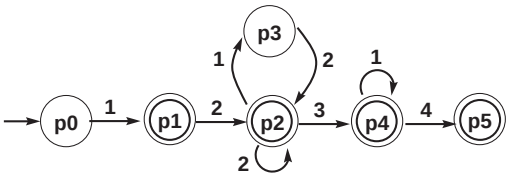


Figura 9.5: Autômato A_{R_s}

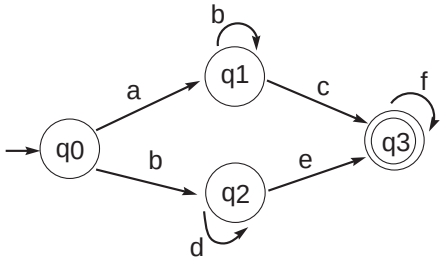


Figura 9.6: Autômato A_{R_i}

Neste exemplo, mostraremos como C_5^3 é gerado a partir de L_4^2 e L_4^3. Vamos considerar que tais conjuntos já foram obtidos nos passos (2,4) e (3,4) respectivamente e são dados através das tabelas 9.1 e 9.2.
O conjunto C_5^3 é obtido como descrito a seguir:

1. Primeiramente, são geradas todas as seqüências de formato de rank 3 e comprimento 5 p-válidas com relação ao autômato A_{R_s}. As seqüências geradas são $< 1, 2, 1, 2, 3 >$, $< 1, 2, 3, 1, 1 >$, $< 1, 2, 2, 2, 3 >$ e $< 1, 2, 2, 3, 1 >$.

L_4^2

$L_4^2(q_0)$	$L_4^2(q_1)$	$L_4^2(q_2)$	$L_4^2(q_3)$
	$\begin{pmatrix} \perp & b \\ \perp & b \\ \perp & b \\ a & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & d \\ d & \perp \\ \perp & d \\ b & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & c \\ b & \perp \\ \perp & b \\ a & \perp \end{pmatrix}, \begin{pmatrix} \perp & e \\ d & \perp \\ \perp & d \\ b & \perp \end{pmatrix}, \begin{pmatrix} \perp & e \\ \perp & d \\ d & \perp \\ b & \perp \end{pmatrix}$

Tabela 9.1: Conjunto L_4^2

L_4^3

$L_4^3(q_0)$	$L_4^3(q_1)$	$L_4^3(q_2)$	$L_4^3(q_3)$
	$\begin{pmatrix} \perp & \perp & b \\ \perp & b & \perp \\ \perp & b & \perp \\ a & \perp & \perp \end{pmatrix}$		$\begin{pmatrix} f & \perp & \perp \\ \perp & \perp & c \\ \perp & b & \perp \\ a & \perp & \perp \end{pmatrix}$

Tabela 9.2: Conjunto L_4^3

2. Para cada seqüência de formato obtida no passo anterior, devemos gerar o conjunto de seqüências de itens de comprimento 5 de maneira que os *psm's* correspondentes sejam p-válidos.

   (a) Considere a seqüência de formato $< 1, 2, 1, 2, 3 >$. O 4-prefixo (prefixo de comprimento 4) dessa seqüência é $< 1, 2, 1, 2 >$, que possui rank 2 e comprimento 4. Portanto, somente as seqüências de itens correspondentes de L_4^2 serão expandidas para obtermos as seqüências de itens de comprimento 5. Esta tarefa é realizada pelo procedimento *ExpandeSeqsItens* (figura 9.4), como descrito a seguir:

   Passo 1: Devemos selecionar os *psm's* em L_4^2 correspondentes à seqüência de formato $< 1, 2, 1, 2 >$. Na tabela abaixo, mostramos os *psm's* selecionados para cada estado.

$L_4^2(q_2)$	$L_4^2(q_3)$
$\begin{pmatrix} \perp & d \\ d & \perp \\ \perp & d \\ b & \perp \end{pmatrix}$	$\begin{pmatrix} \perp & c \\ b & \perp \\ \perp & b \\ a & \perp \end{pmatrix}, \begin{pmatrix} \perp & e \\ d & \perp \\ \perp & d \\ b & \perp \end{pmatrix}$

   Passo 2: Projetamos esses *psm's* afim de obtermos suas seqüências de itens. Os seguintes conjuntos de seqüências de itens são obtidos: $\{< b, d, d, d >\}$ (com relação a q_2) e $\{< a, b, b, c >, < b, d, d, e >\}$ (com relação a q_3).

   Passo 3: Tais seqüências de itens são expandidas de acordo com o autômato A_{R_i}. A idéia é produzir seqüências de itens p-válidas com relação a cada estado de A_{R_i}. Note que somente as seqüências de itens terminando em q_2 e q_3 serão consideradas para expansão (aquelas descritas no passo 2).

   i. Estado q_0: Não há transição finalizando em q_0.

ii. Estado q_1: As transições finalizando em q_1 são: $q_0 \xrightarrow{a} q_1$ e $q_1 \xrightarrow{b} q_1$. Não há seqüências de itens p-válidas com relação aos estados de origem q_0 ou q_1. Logo, nenhuma seqüência de itens é gerada para o estado q_1.

iii. Estado q_2: As transições finalizando em q_2 são: $q_0 \xrightarrow{b} q_2$ e $q_2 \xrightarrow{d} q_2$.

- $q_0 \xrightarrow{b} q_2$: Não há seqüências de itens p-válidas finalizando em q_0. Logo, nenhuma seqüência de itens será expandida.
- $q_2 \xrightarrow{d} q_2$: A seqüência de itens $< b, d, d, d >$ é p-válida com relação ao estado de origem q_2. Assim, geramos a seqüência de itens $< b, d, d, d, \mathbf{d} >$.

iv. Estado q_3: Transições terminando em q_3: $q_1 \xrightarrow{c} q_3$, $q_2 \xrightarrow{e} q_3$ e $q_3 \xrightarrow{f} q_3$

- $q_1 \xrightarrow{c} q_3$: Não há seqüências de itens p-válidas com relação a q_1 e por isso nenhuma seqüência é expandida.
- $q_2 \xrightarrow{e} q_3$: A seqüência de itens $< b, d, d, d >$ é p-válida com relação a q_2. Geramos então, a seqüência de itens $< b, d, d, d, \mathbf{e} >$.
- $q_3 \xrightarrow{f} q_3$: As seqüências de itens $< a, b, b, c >$ e $< b, d, d, e >$ são p-válidas com respeito a q_3. Assim, elas são expandidas para obtenção das seqüências $< a, b, b, c, \mathbf{f} >$ e $< b, d, d, e, \mathbf{f} >$ respectivamente.

Dessa forma, para a seqüência de formato $< 1, 2, 1, 2, 3 >$, o algoritmo *MSP-Miner* gera as seqüências de itens $< b, d, d, d, d >$, $< b, d, d, d, e >$, $< a, b, b, c, f >$ e $< b, d, d, e, f >$.

Passo 4: Cada seqüência de itens gerada no passo 3 é combinada com a seqüência de formato $< 1, 2, 1, 2, 3 >$. Os *psm's* obtidos dessa maneira são dados abaixo:

$$\begin{pmatrix} \perp & \perp & d \\ \perp & d & \perp \\ d & \perp & \perp \\ \perp & d & \perp \\ b & \perp & \perp \end{pmatrix}, \begin{pmatrix} \perp & \perp & e \\ \perp & d & \perp \\ d & \perp & \perp \\ \perp & d & \perp \\ b & \perp & \perp \end{pmatrix}, \begin{pmatrix} \perp & \perp & f \\ \perp & c & \perp \\ b & \perp & \perp \\ \perp & b & \perp \\ a & \perp & \perp \end{pmatrix} \text{ e } \begin{pmatrix} \perp & \perp & f \\ \perp & e & \perp \\ d & \perp & \perp \\ \perp & d & \perp \\ b & \perp & \perp \end{pmatrix}.$$

(b) Os passos 1, 2, 3 e 4 são novamente executados para as seqüências de formato $< 1, 2, 3, 1, 1 >$, $< 1, 2, 2, 2, 3 >$ e $< 1, 2, 2, 3, 1 >$.

3. C_5^3 é o conjunto contendo exatamente os *psm's* gerados em (a) e (b) acima.

9.4 Fase do Cálculo do Suporte

A etapa do cálculo do suporte do algoritmo *MSP-Miner* é exatamente igual à fase de cálculo do suporte do algoritmo SM, que foi apresentada no final do capítulo 6.

No capítulo seguinte, mostraremos os resultados experimentais dos testes realizados com o algoritmo *MSP-Miner* sobre bancos de dados sintéticos. Analisamos a performance de *MSP-Miner* e o comparamos com uma versão adaptada do algoritmo SM, em que acrescentamos uma etapa de pós-processamento para considerar nossa restrição. Baseado nos resultados obtidos, confirmamos que a incorporação da restrição *dentro* do processo de mineração produz resultados melhores do que sua incorporação somente numa etapa de pós-processamento.

Capítulo 10

Resultados Experimentais e Análise de Performance

Neste capítulo apresentamos os resultados dos vários experimentos realizados com *MSP-Miner* em bancos de dados sintéticos. Além de testar *MSP-Miner*, também comparamos sua performance com a performance do algoritmo SM (apresentado no capítulo 6), adaptado para considerar restrições-PSM. Acrescentamos a SM uma etapa de pós-processamento responsável por selecionar entre os *psm's* minerados, aqueles que satisfazem a restrição tomada. Denominamos o algoritmo assim modificado por SM-R. Nossos resultados confirmam a viabilidade de incorporar a restrição-PSM dentro do processo de mineração, ao invés de considerá-la apenas em uma etapa de pós-processamento.

No final do capítulo será apresentada uma interface gráfica desenvolvida para facilitar o uso do algoritmo *MSP-Miner*, assim como dos algoritmos PM e SM apresentados nos capítulos anteriores.

Os experimentos com *MSP-Miner* e SM-R foram realizados em um Pentium 4 de 3.0 GHz com 1 GB de memória principal executando o sistema operacional *Windows XP Professional*.

10.1 Dados Sintéticos

10.1.1 Gerador de Bancos de Dados

Para geração dos bancos de dados sintéticos, fizemos uso do mesmo programa gerador utilizado nos testes dos algoritmos PM e SM. Como já descrito no capítulo 7 (seção 7.1), nosso programa gerador produz bancos de dados de acordo com os parâmetros de entrada da tabela 7.1. Na geração dos bancos de dados, o número de itens (N) foi configurado para 5000, o número médio de clientes por grupo ($|G|$) foi configurado para 4 e o número médio de transações por cliente ($|C|$), ajustamos para 6. Realizamos nossos testes utilizando dois bancos de dados: um deles contendo 4000 grupos e o outro contendo 8000 grupos. Na tabela 10.1 abaixo são listadas as configurações de todos os parâmetros para os dois bancos de dados gerados.

Nome	$\|D\|$	$\|G\|$	$\|C\|$	$\|R\|$	$\|S\|$	N	N_m	**Tamanho (MB)**
D4-G4-C6-R3-S4	4	4	6	3	4	5000	200	4,3
D8-G4-C6-R3-S4	8	4	6	3	4	5000	400	8,6

Tabela 10.1: Bancos de Dados Sintéticos - Parâmetros

10.1.2 Gerador de Restrições

Como *MSP-Miner* incorpora uma restrição R no processo de mineração, também estamos interessados em analisar a performance e a sensibilidade do algoritmo com relação à restrição por ele considerada. O nosso objetivo é verificar como *MSP-Miner* se comporta para as mais variadas restrições. Para isso, desenvolvemos um *gerador de restrições* baseado no gerador de expressões regulares proposto em [33]. Nosso gerador é capaz de gerar restrições do tipo R = $[R_i, R_s]$. A seguir, daremos alguns detalhes de como esse gerador foi construído.

A restrição R é composta por duas expressões regulares e por esse motivo o gerador de restrições utiliza-se de um procedimento (*REgen*) para criação de expressões regulares de acordo com parâmetros de entrada. Como em [33], esse procedimento produz expressões regulares da forma $(B_1 \mid B_2 \mid \cdots \mid B_n)^*$, onde cada *bloco* B_i é composto por uma

concatenação de *termos* ($B_i = T_1T_2\cdots T_m$). Um termo T_i é uma disjunção de itens ($T_i = s_1 \mid s_2 \mid \cdots \mid s_{r_i}$). Os parâmetros requeridos pelo *gerador de restrições* são:

- I = o número máximo de itens por termo
- T = o número de termos por bloco
- B = o número de blocos
- $iMax$ = o número máximo de itens usados na expressão regular R_i
- r = *rank* de R_s

O gerador de restrições faz uso do procedimento *REgen* para criar a expressão regular R_i, que terá a forma $(B_1 \mid B_2 \mid \cdots \mid B_n)^*$ e conterá B blocos, T termos por bloco e no máximo I itens por termo, sendo que R_i conterá itens do conjunto {1,...,IMax}. O gerador cria a expressão R_s com a forma $1e_12e_2...re_r$, onde **cada** e_i é uma expressão regular sobre $\{1, ..., i\}$ (veja Definição 8.2.1), obtida por meio do procedimento *REgen*, contendo B blocos, T termos por bloco e no máximo I itens por termo. Os itens de e_i são obtidos de {1,...,i}.

A tabela 10.2 mostra todos os parâmetros utilizados na geração das restrições-PSM e também o parâmetro *suporte mínimo*. São dados os valores padrão e os intervalos de valores para os quais os experimentos foram realizados.

Parâmetro	**Valor Padrão**	**Variação**
Nº de Blocos (B)	4	2 - 8
Nº de Termos por Bloco (T)	3	2 - 8
Nº Máx. de Itens por Termo (I)	8	2 - 30
Rank de R_s (r)	10	-
Nº de itens para R_i ($iMax$)	5000	-
Suporte Mínimo	1.0	0.2 - 1.2

Tabela 10.2: Suporte Mínimo e Parâmetros da Restrição-PSM

10.2 Análise de Performance

A figura 10.1 nos mostra os tempos de execução dos algoritmos *MSP-Miner* e SM-R para o primeiro banco de dados da tabela 10.1. Podemos verificar em todos os gráficos dessa figura que *MSP-Miner* sempre executa melhor que SM-R e que a diferença entre seus tempos de execução é bastante significativa. Estes resultados confirmam a eficiência da

incorporação de restrições-PSM durante o processo de mineração ao invés de tratá-las durante uma fase de pós-processamento.

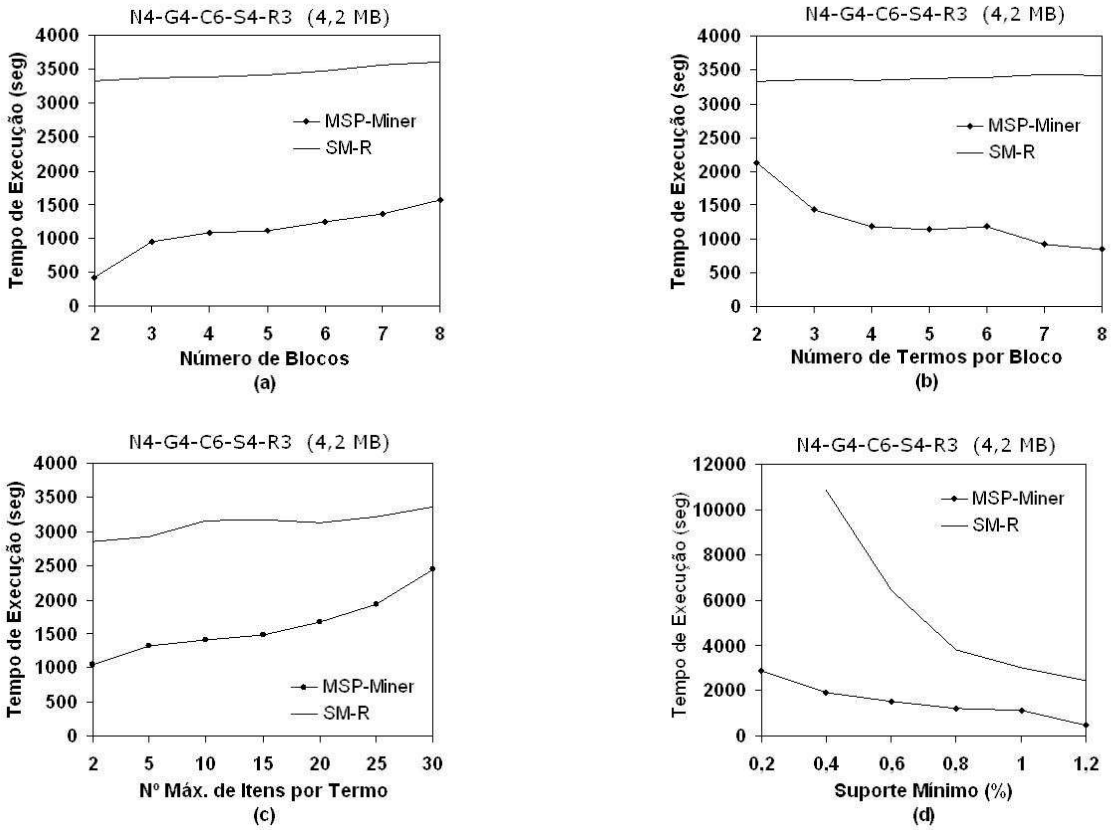


Figura 10.1: Resultados experimentais para banco D4-G4-C6-R3-S4 de 4.2 MB

Os gráficos das figuras 10.1(a), 10.1(b) e 10.1(c) nos mostram os tempos de execução dos dois algoritmos à medida que os parâmetros da restrição-PSM variam. A figura 10.1(a) ilustra a performance dos algoritmos à medida que o número de blocos nas expressões regulares é incrementado de 2 a 8. Como esperado, à medida que aumentamos o número de blocos, a seletividade da restrição-PSM diminui e conseqüentemente o número de *psm's* p-válidos aumenta. Dessa forma, também há um aumento no tempo de execução de *MSP-Miner*. No entanto, para o algoritmo SM-R, o tempo de execução aumenta muito suavemente. Note que SM-R apresenta esse comportamento não somente quando o número de blocos varia, mas também quando os outros parâmetros da restrição-PSM são incrementados (Figures 10.1(b) e 10.1(c)). Isto acontece porque SM-R sempre encontra o conjunto inteiro de *psm's* freqüentes e somente depois ele seleciona aqueles satisfazendo

a restrição. O tempo gasto no processo de mineração (antes do pós-processamento) não depende da restrição considerada. Além disso, a fase de pós-processamento é realizada em memória principal e seu tempo de execução é muito pequeno quando comparado ao tempo gasto no processo de mineração.

A figura 10.1(c) mostra a performance dos algoritmos à medida que o número máximo de itens por termo nas expressões regulares é incrementado de 2 a 30. Uma análise similar àquela realizada para a variação do número de blocos (figura 10.1(a)) pode ser considerada, pois também neste caso há uma diminuição da seletividade da restrição.

A performance dos algoritmos à medida que o suporte mínimo é incrementado de 0.2% a 1.2% é ilustrada na figura 10.1(d). Como esperado, os tempos de execução para ambos algoritmos diminuem, pois poucos candidatos têm o potencial de ser freqüente para altos valores do suporte mínimo. O gráfico mostra que para baixos níveis de suporte, *MSP-Miner* executa muito melhor que SM-R. Isto acontece porque a quantidade de padrões candidatos que chegam à fase de cálculo do suporte em SM-R é muito maior que a quantidade de candidatos em *MSP-Miner*, onde somente os candidatos selecionados permanecem após a fase de geração.

A figura 10.1(b) exibe os tempos de execução dos dois algoritmos à medida que o número de termos por bloco nas expressões regulares é incrementado de 2 a 8. Notamos que o tempo de execução do algoritmo *MSP-Miner* diminui à medida que o número de termos por bloco é incrementado. Realmente, quando as expressões regulares R_i e R_s têm poucos termos por bloco, os *psm's* curtos (tendo pequeno comprimento) podem ser p-válidos assim como os *psm's* longos (devido ao $*$ em $(B_1 \mid B_2 \mid \cdots \mid B_n)^*$). Entretanto, à medida que o número de termos por bloco aumenta, somente *psm's* longos podem ser p-válidos.

A figura 10.2 a seguir ilustra a performance dos algoritmos para o segundo banco de dados da tabela 10.1. Como esperado, há um aumento nos tempos de execução de ambos algoritmos, mas *MSP-Miner* continua executando melhor que SM-R, como no primeiro banco de dados. Quando o número de blocos varia de 2 a 6, o tempo de execução de *MSP-Miner* aumenta linearmente. Para números maiores de blocos, entretanto, a performance de *MSP-Miner* cai significativamente.

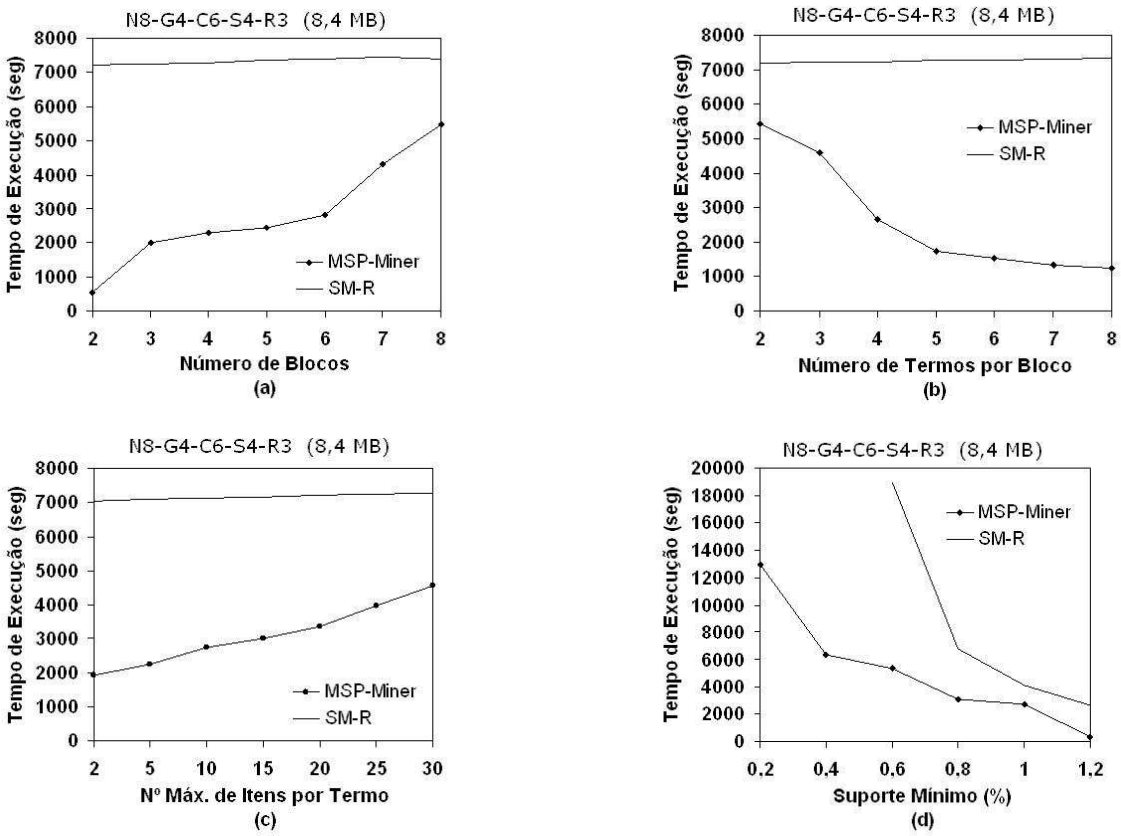


Figura 10.2: Resultados experimentais para banco D8-G4-C6-R3-S4 de 8.4 MB

10.3 Interface Gráfica

Nesta seção apresentamos a interface gráfica desenvolvida para facilitar o uso dos três algoritmos propostos para mineração de seqüências múltiplas. Através dela, o usuário pode informar, de maneira fácil, o banco de dados a ser minerado, os parâmetros necessários a mineração (suporte, expressões regulares, etc.) e escolher um algoritmo para realizar a tarefa de descoberta. Além disso, a interface também possibilita uma melhor visualização dos resultados e padrões minerados.

A figura 10.3 a seguir ilustra a janela principal de *MultiMine* (assim denominamos o conjunto interface e algoritmos). O usuário deve informar um nome para a configuração de mineração, o banco de dados a ser minerado, o suporte mínimo e escolher qual algoritmo realizará a mineração. Caso seja escolhido *MSP-Miner*, será exibida a janela da figura 10.4 para que as expressões regulares referentes à restrição sejam informadas.

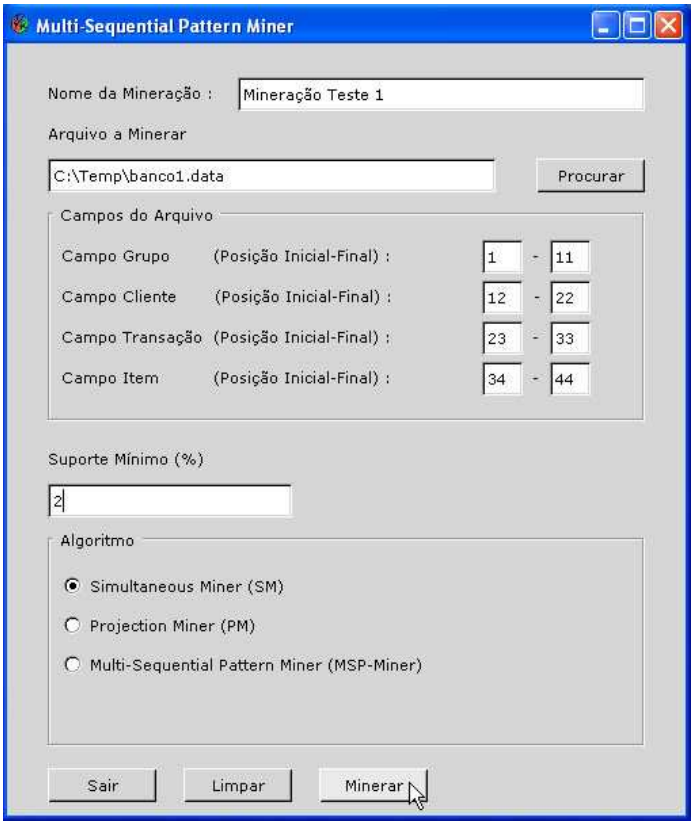


Figura 10.3: Janela Inicial do MultiMine

Figura 10.4: Janela para Entrada de Expressões Regulares

Quando o botão "Minerar" da janela principal é acionado, MultiMine coloca em execução o algoritmo escolhido. O andamento da mineração pode ser acompanhado pelo

usuário, como ilustra a janela da figura 10.5.

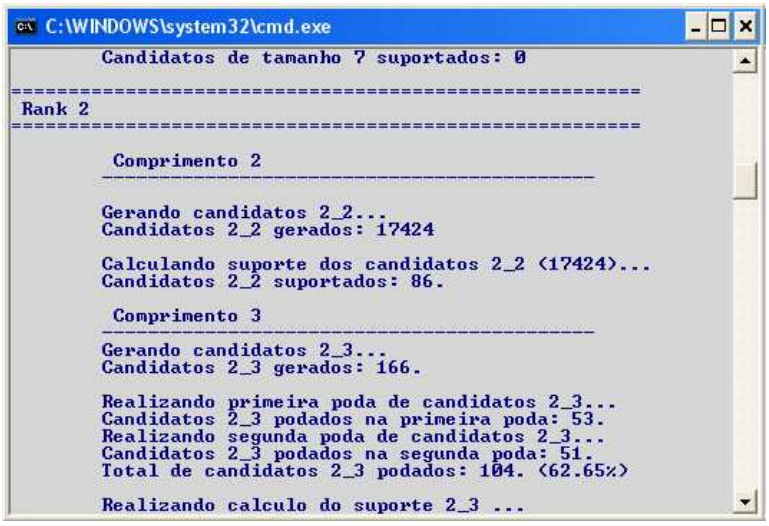


Figura 10.5: Processo de Mineração

Assim que a mineração é concluída, o programa exibe alguns resultados estatísticos obtidos da mesma, como o número de padrões múltiplos encontrados, o tempo gasto na mineração, etc. Veja a figura 10.6.

Figura 10.6: Janela de Resultados Estatisticos da Mineração

O usuário poderá visualizar os padrões minerados acionando o botão "Visualizar Padrões" da janela de estatísticas. A figura 10.7 ilustra a janela de listagem desses padrões. O usuário seleciona o rank e o comprimento dos *psm's* minerados que deseja visualizar e cada padrão é listado através das duas seqüências que o compõe.

MultiMine também fornece uma visualização gráfica (em forma de matriz) dos padrões encontrados. O usuário seleciona um padrão a partir da janela de listagem de *psm's* e

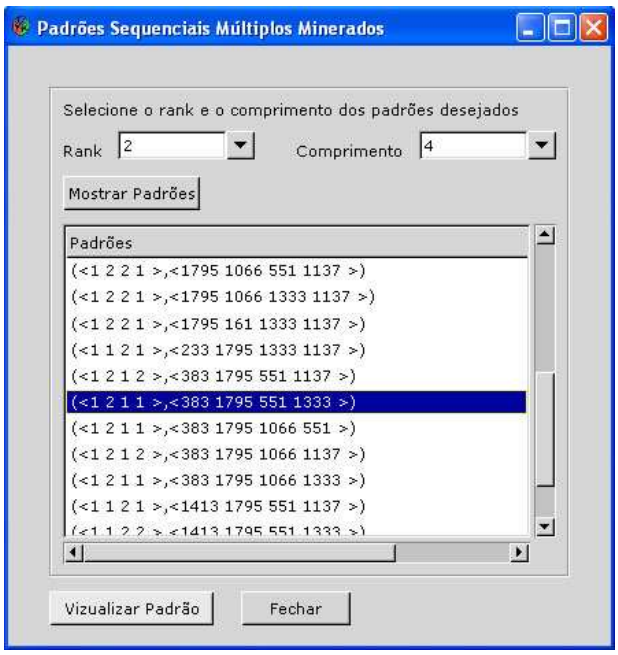


Figura 10.7: Janela de Listagem dos Padrões Minerados

aciona o botão "Visualizar Padrão". O *psm* selecionado é então mostrado graficamente. A janela da figura 10.6 ilustra o padrão múltiplo (<1,2,1,1>,<383,1795,551,1333>).

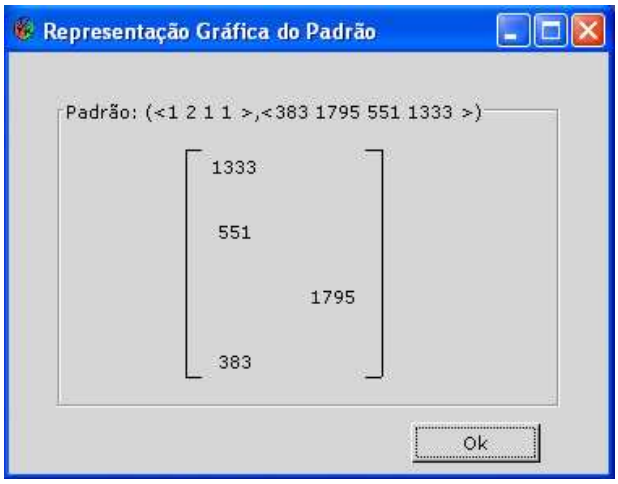


Figura 10.8: Janela para Visualização Gráfica de Padrão

Neste capítulo apresentamos os resultados dos principais testes realizados com o algoritmo *MSP-Miner*. Analisamos sua performance à medida em que variamos os vários parâmetros que definem a restrição-PSM considerada e ainda comparamos *MSP-Miner* com o algoritmo SM adaptado com uma fase de pós-processamento para consideração da restrição. Também mostramos a interface gráfica desenvolvida para facilitar o uso dos

programas mineradores, não só para *MSP-Miner*, mas também para os algoritmos SM e PM apresentados nos capítulos anteriores.

Capítulo 11

Conclusão e Trabalhos Futuros

A mineração de padrões freqüentes em grandes bancos de dados constitui um importante problema em mineração de dados. Os padrões seqüenciais simples, já bastante estudados nos últimos anos, são padrões temporais proposicionais. Nesta dissertação propusemos um novo padrão temporal, que denominamos de *padrão seqüencial múltiplo* (psm) e mostramos que, diferente dos padrões seqüenciais comumente estudados, nosso padrão não pode ser especificado através da Lógica Temporal Proposicional e necessita da expressividade da Lógica Temporal de Primeira Ordem. Nesta pesquisa, exploramos o problema de mineração desses padrões *com* e *sem* restrições.

Mineração de *psm's* sem restrições. Apresentamos os algoritmos PM e SM para mineração de todos os *psm's* freqüentes em um banco de dados D com relação a um dado suporte mínimo. PM e SM não consideram nenhum outro tipo de restrição a respeito dos padrões minerados além do suporte mínimo. Suas idéias principais são relembradas a seguir:

- **PM** (*Projection Miner*). Efetua a mineração dos padrões seqüenciais múltiplos decompondo-os em duas componentes proposicionais: a *seqüência de itens* e a *seqüência de formato*. PM trabalha com essas seqüências (simples) nas fases de geração e poda de maneira independente, ou seja, seqüências de itens de comprimento maior são obtidas através da combinação de seqüências de itens geradas na iteração anterior, da mesma forma que no algoritmo GSP (para mineração de padrões se-

qüênciais simples/proposicionais). De maneira análoga, PM obtém as seqüências de formato de maior comprimento.

- **SM** (*Simultaneous Miner*). Minera os *psm's* freqüentes sem decompô-los em seqüências simples. SM trabalha com o padrão múltiplo *completo* nas fases de geração e poda.

Através dos resultados experimentais, podemos verificar que o algoritmo PM, embora decomponha o padrão múltiplo em seqüências simples, apresenta uma performance consideravelmente inferior à performance do algoritmo SM, que realiza a mineração do padrão de primeira ordem sem decompô-lo a componentes proposicionais.

Mineração de *psm's* com restrições. Neste trabalho, também estendemos uma técnica utilizada na mineração de padrões seqüenciais proposicionais para incorporação de restrições no processo de mineração de *psm's*. O objetivo é reduzir o número de padrões minerados, fornecendo ao usuário um controle do processo de mineração afim de que sejam encontrados somente os padrões seqüenciais múltiplos de potencial interesse. Propusemos então o algoritmo *MSP-Miner*, que descobre os *psm's* satisfazendo uma restrição expressa por expressões regulares:

- **MSP-Miner** (*Multi-Sequence Pattern Miner*). Incorpora restrições durante o processo de mineração, considerando condições *não-antimonotônicas* (especificadas por expressões regulares) nas fases de geração de candidatos e poda. A restrição tomada pelo algoritmo é composta por duas expressões regulares. Uma delas é considerada na *seqüência de itens* do *psm* e a outra atua na *seqüência de formato* do mesmo, responsável por restringir os formatos dos *psm's* a serem minerados.

Realizamos vários experimentos para avaliar a performance e a escalabilidade de *MSP-Miner* e ainda o comparamos com uma versão do algoritmo SM adaptada para considerar as restrições (adicionamos a SM uma etapa de pós-processamento que seleciona os padrões satisfazendo a restrição), que chamamos de SM-R. Os resultados obtidos confirmam a viabilidade de incorporar restrições expressas por expressões regulares durante o processo de mineração também no contexto de mineração de padrões temporais de primeira ordem.

Trabalhos Futuros

Os seguintes tópicos, consistindo de possíveis otimizações e extensões do trabalho realizado nesta dissertação, constituem objeto de estudo futuro:

- **Testes com bancos de dados reais**. Os três algoritmos apresentados neste trabalho (PM, SM e MSP-Miner) foram testados apenas sobre bancos de dados sintéticos. Testes com bancos de dados reais complementariam os resultados e permitiriam uma análise mais efetiva dos algoritmos.

- **Otimização para etapa do cálculo do suporte**. Assim como a maioria dos algoritmos baseados na idéia de *Apriori*, onde a mineração se realiza em três fases: geração, poda e contagem do suporte, os algoritmo PM, SM e MSP-Miner também consomem a maior parte do tempo de execução na fase de contagem do suporte. Logo, mecanismos e/ou estruturas adicionais para otimização dessa etapa seriam interessantes.

  - **Poda no banco de dados**. Um possível ganho em desempenho para os três algoritmos poderia ser alcançado realizando-se uma poda no banco de dados a cada iteração. Se o algoritmo está na iteração k e uma seqüência múltipla do banco de dados não contribui na contagem do suporte de nenhuma seqüência candidata, então tal seqüência múltipla não precisa ser analisada nas próximas iterações, pois também não contribuirá na contagem dos padrões maiores.

- **Considerar transações contendo *itemsets***. Para simplificar a representação e facilitar a exploração da essência do problema de minerar seqüências múltiplas, temos considerado que as transações possuem um único item. Uma extensão dos algoritmos poderia ser feita para que fossem tratadas seqüências múltiplas de *itemsets* ao invés de seqüências múltiplas de itens.

APÊNDICE A

Prova do Teorema 6.2.1

(1) Seja $\sigma \in L_k^n$. Consideramos que as colunas de σ estão ordenadas de acordo com o procedimento *OrdenaColunas* apresentado no capítulo 4, seção 4.1.3. Para $n = 1$, a validade é verificada, pois o procedimento de geração de candidatos usado no algoritmo GSP é correto, isto é, o conjunto de candidatos sempre contém todas as seqüências interessantes. Para $n = k = 2$, a validade também é verificada: como $\sigma \in L_2^2$, os *psm's* $\{< \sigma_1^1 >\}$ e $\{< \sigma_2^2 >\}$ estão em L_1^1. Logo, pela forma com que C_2^2 é gerado, é claro que $L_2^2 \subseteq C_2^2$. Seja n, k, $2 \leq n \leq k$, $k \geq 3$. Precisamos considerar os seguintes casos:

- o (único) item da linha 2 é colocado na coluna 1 e
  - existe um item na coluna n colocado na linha i, com $i < k$: neste caso, $\sigma \in L_k^n \bowtie L_k^n$.
  - a coluna n contém somente um item, o qual é colocado na linha k: neste caso, $\sigma \in L_{k-1}^{n-1} \bowtie L_k^n$.
- o (único) item da linha 2 é colocado na coluna 2 e
  - existe um item na coluna n colocado na linha i, onde $i < k$: neste caso, $\sigma \in L_k^n \bowtie L_{k-1}^{n-1}$.
  - a coluna n contém somente um item, o qual é colocado na linha k: neste caso, $\sigma \in L_{k-1}^{n-1} \bowtie L_{k-1}^{n-1}$.

(2) Visto que $\tau \subseteq \sigma$ e $\tau \neq \sigma$, então $\tau \in \Sigma_p^m$ com $m < n$ ou $p < k$.

- Seja $m < n$ e $m \leq p \leq k$. Neste caso, τ é obtido eliminando-se pelo menos uma coluna de σ e uma das linhas correspondendo aos itens dessa coluna. Logo, $p < k$. Podemos supor, sem perda de generalidade, que somente uma coluna foi eliminada, isto é, $m = n - 1$. Podemos afirmar que τ está contido em um *psm* $\tau' \in \Sigma_{k-1}^{n-1}$, tal que $\tau' \subseteq \sigma$. Visto que τ não é interessante, então τ' também não é, isto é, $\tau' \notin L_{k-1}^{n-1}$. Assim, pela forma em que os *psm's* são gerados em C_k^n, $\sigma \notin C_k^n$.

- Seja $m = n$ e $n \leq p < k$. Neste caso, τ é obtido eliminando-se pelo menos uma linha de σ. Podemos supor, sem perda de generalidade, que somente uma linha foi eliminada. Conseqüentemente, $\tau \in \Sigma^n_{k-1}$. Visto que τ não é interessante, isto é $\tau \notin L^n_{k-1}$, então, pela forma com que os *psm's* são gerados em C^n_k, $\sigma \notin C^n_k$.

Referências Bibliográficas

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In Philip S. Yu and Arbee S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[2] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, editors, *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, volume 1057, pages 3–17. Springer-Verlag, 25–29 1996.

[3] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 108–118. ACM Press, 2000.

[4] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal. Multi-dimensional sequential pattern mining. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 81–88. ACM Press, 2001.

[5] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[6] C. Bettini, X. S. Wang, and S. Jajodia. Testing complex temporal relationships involving multiple granularities and its application to data mining (extended abstract). In *PODS '96: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 68–78, New York, NY, USA, 1996. ACM Press.

[7] G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Knowledge Discovery and Data Mining*, pages 16–22, 1998.

[8] H. Lu, L. Feng, and J. Han. Beyond intra-transaction association analysis: Mining multi-dimensional inter-transaction association rules. *ACM Trans. Inf. Syst.*, 18(4):423–454, 2000.

[9] B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: a temporal logic approach. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad, editors, *Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, 1996. AAAI Press.

[10] M. Joshi, G. Karypis, and V. Kumar. Universal formulation of sequential patterns. University of Minnesota, Computer Science and Engineering, 1999.

[11] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359. ACM Press, 2000.

[12] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224. IEEE Computer Society, 2001.

[13] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

[14] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the SIAM International Conference on Data Mining*, San Francisco, CA.

[15] G. Piatetsky-Shapiro, U. Fayyad, and P. Smith. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[16] C. J. Matheus, P. K. Chan, and G. Piatetsky-Shapiro. Systems for knowledge discovery in databases. *IEEE Trans. On Knowledge And Data Engineering*, 5:903–913, 1993.

[17] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216. ACM Press, 1993.

[18] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 144–155. Morgan Kaufmann Publishers Inc., 1994.

[19] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Mach. Learn.*, 36(1-2):105–139, 1999.

[20] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Mach. Learn.*, 29(2-3):131–163, 1997.

[21] A. Srivastava, E. Han, V. Kumar, and V. Singh. Parallel formulations of decision-tree classification algorithms. *Data Min. Knowl. Discov.*, 3(3):237–261, 1999.

[22] M. Ester, H-P Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD '96: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[23] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, June 1998.

[24] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[25] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data*

Bases, pages 420–431, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[26] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD)*, pages 67–73. AAAI Press, 14–17 1997.

[27] R. Srikant and R. Agrawal. Mining generalized association rules. *Future Generation Computer Systems*, 13(2–3):161–180, 1997.

[28] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*, pages 412–421, Washington, DC, USA, 1998. IEEE Computer Society.

[29] W. Lin, M. A Orgun, and G Williams. An overview of temporal data mining. In *Proceedings of the 1st Australian Data Mining Workshop*, pages 83–90. University of Technology, Sydney, 2002.

[30] C. Antunes and A. Oliveira. Temporal data mining: an overview. In *Workshop on Temporal Data Mining (KDD)*, pages 1–13, San Francisco, 2001.

[31] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, 2000.

[32] R. J. Bayardo Jr. Efficiently mining long patterns from databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 85–93, New York, NY, USA, 1998. ACM Press.

[33] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *The VLDB Journal*, pages 223–234, 1999.

[34] F. Masseglia, F. Cathala, and P. Poncelet. The PSP approach for mining sequential patterns. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 176–184. Springer-Verlag, 1998.

[35] D. E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[36] H. Pinto. Multi-dimensional sequential pattern mining. M.Sc. thesis, Computing Science, Simon Fraser University, April 2001.

[37] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg CUBE. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 359–370. ACM Press, 1999.

[38] R. O. Santos. Mineração de padrões seqüenciais com dimensões temporalmente variantes. Dissertação de Mestrado, Ciência da Computação, Universidade Federal de Uberlândia, Março 2004.

[39] A. R. Santos. Mineração de padrões seqüenciais generalizados utilizando programação genética. Dissertação de Mestrado, Ciência da Computação, Universidade Federal de Uberlândia, Março 2003.

[40] C. Masson and F. Jacquenet. Mining frequent logical sequences with SPIRIT-LoG. In S. Matwin and C. Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 166–181. SV, 2003.

[41] S. D. Lee and L. De Raedt. Constraint based mining of first-order sequences in SeqLog. pages 80–96. University of Alberta, Edmonton, Canada, July 2002.

[42] N. Jacobs and H. Blockeel. From shell logs to shell scripts. In C. Rouveirol and M. Sebag, editors, *ILP '00: Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157, pages 80–90, September 2001.

[43] S. de Amo, D. A. Furtado, A. Giacometti, and D. Laurent. An apriori-based approach for first-order temporal pattern mining. In *Proceedings of the 19th Brazilian Symposium on Databases*, pages 48–61, Brasilia, Brazil, October 2004.

[44] B. Liu, W. Hsu, and Y. Ma. Pruning and summarizing the discovered associations. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 125–134. ACM Press, 1999.

[45] D. Shah, L. Lakshmanan, K. Ramamritham, and S. Sudarshan. Interestingness and pruning of mined patterns. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, 1999.

[46] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of 1998 ACM SIGMOD International Conference Management of Data*, pages 13–24, 1998.

[47] B. Padmanabhan and A. Tuzhilin. Small is beautiful: discovering the minimal set of unexpected patterns. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 54–63. ACM Press, 2000.

[48] M. J. Zaki. Sequence mining in categorical domains: incorporating constraints. In *CIKM '00: Proceedings of the ninth international conference on Information and knowledge management*, pages 422–429. ACM Press, 2000.

Livros Grátis

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

Milhares de Livros para Download:

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