Universidade Federal Fluminense

Ciro Bastos Barbosa

Explorando Técnicas de Redução de Base de Dados na Mineração de Padrões Seqüenciais

NITERÓI

2006

Ciro Bastos Barbosa

Explorando Técnicas de Redução de Base de Dados na Mineração de Padrões Seqüenciais

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Otimização Combinatória e Inteligência Artificial.

Orientador:
Prof. Alexandre Plastino de Carvalho. DSc.

UNIVERSIDADE FEDERAL FLUMINENSE

NITERÓI

2006

Explorando Técnicas de Redução de Base de Dados
na Mineração de Padrões Seqüenciais

Ciro Bastos Barbosa

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Otimização Combinatória e Inteligência Artificial.

Aprovada por:

Prof. Alexandre Plastino de Carvalho / IC-UFF (Presidente)

Profa. Bianca Zadrozny / IC-UFF

Prof. Caetano Traina Júnior / ICMC-USP

Niterói, Dezembro de 2006.

Em memória das minhas queridas avós,

Gracieta e Maria Felicíssima

Agradecimentos

Gostaria de agradecer primeiramente a Deus por todos os momentos maravilhosos que tenho tido em minha vida. Por todos os momentos felizes, e porque não os tristes? Por ter me dado saúde e capacidade para aproveitá-los.

A conclusão do mestrado é mais um sonho que se realiza. Uma tarefa importante cumprida, que só pôde ser realizada com a ajuda das pessoas mais importantes da minha vida: Deus, familiares e amigos. Essa é a oportunidade para agradecê-los e dizer que sem elas nada disso seria possível.

Obrigado pelos ensinamentos e puxões-de-orelha do professor Alexandre Plastino que me orientou no árduo trabalho de desenvolvimento desta dissertação e cuidou para que eu não desanimasse. Agradeço aos professores de mestrado da UFF, que compartilharam comigo seus conhecimentos.

Resumo

Ao longo dos últimos dez anos, estratégias para extração de padrões seqüenciais vêm sendo desenvolvidas e aprimoradas. Algumas delas têm como base o algoritmo iterativo *Apriori*, desenvolvido para a extração de conjuntos freqüentes, como por exemplo a estratégia *GSP*. Experimentos computacionais realizados nesta categoria de estratégias indicam que a etapa de identificação das seqüências freqüentes (padrões seqüenciais), ou seja, a fase de contagem do suporte das seqüências candidatas consome grande parte do tempo total de execução. Sendo assim, nesta dissertação, com o objetivo de reduzir o custo de diversas leituras da base de dados e o esforço computacional da fase de contagem de seqüências candidatas, típicos dos algoritmos iterativos de extração de padrões seqüenciais, propõe-se a redução progressiva da base de dados ao longo da execução das iterações. Desta forma, menos transações são lidas a cada iteração e menor passa a ser o custo computacional para a obtenção do suporte de cada seqüência candidata. Os resultados avaliados, a partir de diferentes combinações de bases de dados e suportes mínimos, mostraram que as técnicas de redução de base implementadas no algoritmo proposto *GSP2P* reduzem significativamente o tempo de execução total do algoritmo sem poda de base *GSP2* (implementação do *GSP*). Neste mesmo trabalho, com o objetivo de validar o uso das técnicas propostas e estender as suas aplicações, as técnicas de redução de base foram aplicadas ao problema de extração de padrões seqüenciais baseada em restrições. Os resultados avaliados, a partir de diferentes combinações de bases de dados e valores de seletividade das restrições, mostraram que as técnicas de redução de base implementadas no algoritmo proposto *GSP2P-F* reduzem significativamente o tempo de execução total do algoritmo sem poda de base *GSP2-F*.

Abstract

During the last ten years, many algorithms have been proposed to mine sequential patterns. Some of them are based on the *Apriori* algorithm, developed to iteratively mine frequent itemsets, for example the *GSP* algorithm. Results obtained from experiments using these category of algorithms have shown that the candidate support count phase spends a huge part of the execution time. In this work, aiming at reducing the computational cost of multiple database scans and the computational effort to count the support of the candidate sequences, typical of iterative algorithms for the problem of mining sequential patterns, we propose the progressive reduction of the database during the execution of the algorithm. Therefore, fewer transactions are read at each iteration and the computational cost of counting the support of each candidate is reduced. Results obtained from evaluating different combinations of databases and minimum supports have shown that the database pruning techniques, adopted by the proposed algorithm *GSP2P*, significantly reduce the total execution time of the *GSP2* algorithm (implementation of *GSP*), which does not use pruning mechanisms. In this same work, aiming at validating the use of the proposed database pruning techniques and extending their applications, the techniques were applied to the problem of constraint-based sequential patterns mining. Results obtained from evaluating different combinations of databases and constraint selectivity values have shown that the database pruning techniques, adopted by the proposed algorithm *GSP2P-F*, significantly reduce the total execution time of the *GSP2-F* algorithm, which does not use pruning mechanisms.

Palavras-chave

1. Mineração de Dados
2. Padrões Seqüenciais
3. Algoritmos Iterativos
4. Redução da Base de Dados

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Um padrão seqüencial é caracterizado por eventos que se sucedem no tempo e que aparecem com significativa freqüência em uma base de dados. Estes padrões podem ser utilizados para prever um evento futuro baseado nos anteriores. Pode-se citar, por exemplo, um padrão seqüencial extraído de uma base de dados de uma locadora de vídeo ([2]) em que: parte significativa dos clientes alugam os seguintes filmes nesta ordem: "Guerra nas Estrelas", "O Império Contra-Ataca" e "Retorno de Jedi". A representação deste padrão seria <(Guerra nas Estrelas) (O Império Contra-Ataca) (Retorno de Jedi)>. A partir deste exemplo, pode-se prever que um cliente, depois de alugar "Guerra nas Estrelas" e "O Império Contra-Ataca", tem grande probabilidade de alugar o "Retorno de Jedi".

A mineração de padrões seqüenciais pode ter várias aplicações na área comercial e de pesquisa. Na medicina, por exemplo, o diagnóstico de doenças ou a necessidade de aplicação de medicamentos podem ser previstos a partir do histórico de sintomas ou doenças ocorridas anteriormente no paciente, como citado em [14, 15, 17].

Pode-se realizar a mineração sobre o histórico de seqüências de acesso a páginas *web* pelos diferentes usuários. Os padrões minerados representam seqüências de páginas com maior freqüência de acesso. Estas informações são úteis para reestruturar sites ou para inserir dinamicamente *links* em determinadas páginas a partir dos padrões de acesso, como citado em [5, 7, 16, 20].

Na área comercial, é possível estimar possíveis compras futuras dos clientes através de padrões seqüenciais minerados do histórico de compras. Estas informações podem ser utilizadas para gerar sugestões de produtos através de *emails* promocionais.

A busca de padrões seqüenciais ocorre sobre uma base de dados transacional, em que cada transação possui: o identificador do seu proprietário, chamado consumidor, a informação de

quando ela foi realizada e os itens que a compõem. Em uma analogia com uma base de dados de transações de um supermercado, a identificação do consumidor poderia ser o número do seu cartão de crédito, o momento de realização da transação seria a data e a hora da compra e os itens seriam os artigos que foram adquiridos pelo cliente. Cada consumidor possui sua própria seqüência de transações, ordenadas pela data e hora de cada transação, chamada seqüência de consumidor.

A mineração de padrões seqüenciais objetiva descobrir todas as seqüências freqüentes de eventos, ordenados no tempo, em que cada evento é um conjunto de itens adquiridos em uma mesma transação.

Os padrões seqüenciais representam todas as seqüências de eventos existentes na base de dados, que aparecem em um número mínimo, preestabelecido, de seqüências de consumidores. Este número mínimo de consumidores é chamado de suporte mínimo e, se a seqüência de eventos tiver um suporte igual ou maior que o mínimo, ela é chamada freqüente. O suporte de uma seqüência de eventos indica a relevância desta. O valor do suporte mínimo, que uma seqüência deve ter para ser considerada freqüente, deve ser especificado pelo usuário. Como exemplo, considerando a seqüência de eventos <(brinquedo)(pilha)> e um suporte mínimo de dois consumidores, esta seqüência será freqüente se pelo menos dois consumidores comprarem os itens "brinquedo" e "pilha" em transações diferentes, nesta ordem. O suporte de um padrão seqüencial também pode ser definido como uma porcentagem do número de seqüências de consumidores em que o padrão ocorre em relação ao número total de seqüências de consumidores da base de dados.

Ao longo dos últimos dez anos, estratégias para extração de padrões seqüenciais vêm sendo desenvolvidas e aprimoradas. Em geral, os algoritmos propostos para este problema podem ser categorizados em três classes: (1) métodos baseados no *Apriori* [1], como o *AprioriAll* [2] e o *GSP* [17]; (2) métodos que exploram a base de dados em um formato verticalizado, como o *SPADE* [20] e o *SPAM* [3]; e (3) métodos que projetam e particionam recursivamente a base de dados, como o *FreeSpan* [9] e o *PrefixSpan* [13].

O algoritmo *GSP* (***G**eneralized **S**equential **P**atterns*) [17], desenvolvido por *Agrawal* e *Srikant* em 1996, foi uma das primeiras estratégias propostas para solução deste problema e será objeto de estudo neste trabalho. O *GSP* tem como base o algoritmo *Apriori*, proposto para o problema de extração de conjuntos freqüentes pelos mesmos autores. Assim como o algoritmo *Apriori*, o *GSP* realiza várias iterações para encontrar todos os padrões seqüenciais. Considerando que o tamanho de uma seqüência é determinado pelo número de itens que ela possui, inicialmente, em uma primeira leitura da base, o conjunto de seqüências freqüentes de

tamanho 1 (F_1) é encontrado. Estas são utilizadas para gerar o conjunto de todas as possíveis seqüências de tamanho 2 (C_2), que são chamadas seqüências candidatas. As seqüências candidatas são armazenadas em uma estrutura de dados chamada árvore *hash*. Na segunda leitura da base, a seqüência de cada consumidor é confrontada com as seqüências candidatas de C_2, para a contagem de suporte. As que possuírem suporte maior ou igual ao mínimo formam o conjunto F_2 (seqüências freqüentes de tamanho 2). O conjunto F_2 será utilizado para gerar as candidatas de tamanho 3 (C_3) na próxima iteração do algoritmo. Na k-ésima iteração, candidatas de tamanho k (C_k) são geradas através da combinação das seqüências freqüentes de tamanho $k-1$ (F_{k-1}). Cada seqüência candidata de C_k possui um item a mais do que as seqüências de F_{k-1}. Ainda na k-ésima iteração, percorre-se a base de dados para contar o suporte das seqüências candidatas de C_k. O algoritmo termina quando nenhuma nova candidata é gerada ou quando nenhuma candidata possui suporte maior ou igual ao suporte mínimo.

O algoritmo *GSP* é uma estratégia importante, tendo sido a base de vários outros algoritmos, tais como o *PSP* [10] e o *GSP2* [6]. O algoritmo *PSP* (***P**refix Tree for **S**equential **P**atterns*) utiliza uma árvore de prefixo para armazenar as seqüências candidatas no lugar da árvore *hash* utilizada no *GSP*. A árvore de prefixo foi adotada para melhorar o desempenho da etapa de contagem do suporte das candidatas e reduzir a utilização de memória principal. O algoritmo *GSP2*, nas iterações iniciais, utiliza uma estrutura de dados, chamada *ABS-SP* (***A**rray-**B**ased **S**tructure for **S**equential **P**atterns*), em substituição à mesma árvore *hash*. Na estrutura *ABS-SP*, o acesso ao conjunto de candidatas é feito diretamente, através da indexação das candidatas em um vetor e em uma matriz, reduzindo o tempo de acesso às candidatas e, conseqüentemente, o tempo de execução do algoritmo.

O algoritmo *GSP* é a base também de algoritmos desenvolvidos para a extração de padrões seqüenciais baseada em restrições, tais como os algoritmos *GSP-F* [19] e *SPIRIT* [8].

Experimentos computacionais realizados em [6, 10] indicam que a etapa de contagem do suporte das seqüências candidatas consomem a maior parte do tempo total de execução em estratégias como o *GSP*. Por este motivo, pesquisadores têm concentrado esforços na melhoria desta etapa.

Neste contexto, com o objetivo de reduzir o custo de diversas leituras da base de dados e o esforço computacional da fase de contagem das seqüências candidatas, típicos dos algoritmos iterativos de extração de padrões seqüenciais, propõe-se, neste trabalho, a utilização de técnicas de redução progressiva da base de dados ao longo das iterações. Desta forma, menos seqüências de consumidores são lidas a cada iteração da base de dados e menor passa a ser o custo computacional para a obtenção do suporte de cada seqüência candidata.

As técnicas propostas são baseadas nas estratégias de redução de base de dados apresentadas pelos autores dos algoritmos *DHP* [12] e *kDCI++* [11] no contexto da mineração de conjuntos freqüentes. A redução de base é realizada progressivamente a cada iteração eliminando-se itens nas seqüências de consumidores que, necessariamente, não serão úteis na contagem de suporte de nenhuma seqüência freqüente.

O bom desempenho de algoritmos como *SPADE*, *SPAM*, *FreeSpan* e *PrefixSpan* depende de a base de dados caber inteiramente (ou de forma particionada, dependendo da estratégia) na memória principal. Desta forma, estes algoritmos podem também se beneficiar de técnicas de redução da base de dados. A poda da base seria progressivamente executada até que esta (ou cada partição desta) pudesse ser armazenada em memória principal em estruturas de dados específicas de cada estratégia.

As técnicas propostas para a redução da base de dados serão incorporadas ao algoritmo *GSP*. A avaliação destas técnicas será realizada a partir da comparação de duas implementações do GSP: a versão *GSP2*, apresentada em [6], e a versão *GSP2P* (*GSP2* com Poda), desenvolvida neste trabalho, que incorpora ao *GSP2* as técnicas propostas de poda da base.

Com o objetivo de validar o uso das técnicas propostas e estender as suas aplicações, propõe-se também, neste trabalho, a utilização das técnicas de redução da base de dados no problema de extração de padrões seqüenciais baseada em restrições.

A extração de padrões seqüenciais baseada em restrições permite ao usuário restringir o espaço de busca através da definição de critérios, eliminando padrões seqüenciais que não o interessam. Assim, são geradas menos seqüências freqüentes e o esforço computacional necessário é reduzido, tornando a extração dos padrões seqüenciais mais eficiente.

Neste contexto, em [19], o algoritmo *GSP-F* (*GSP with Dataset Filtering*) foi proposto. Este algoritmo extrai os padrões seqüenciais iterativamente, da mesma forma que o *GSP*, em que, a cada iteração k, a base de dados é inteiramente lida e são encontradas as seqüências freqüentes de tamanho k. Embora a base de dados seja inteiramente lida, apenas as seqüências de consumidores que possam contribuir na contagem de algum padrão seqüencial que satisfaça às restrições do usuário são consideradas na fase de contagem de suporte das seqüências candidatas a cada iteração.

A avaliação das técnicas de redução da base de dados, quando aplicadas ao problema de extração de padrões seqüenciais baseada em restrições, será realizada a partir da comparação de duas estratégias. A primeira, denominada *GSP2-F*, é uma implementação da estratégia *GSP-F*, desenvolvida no escopo deste trabalho. A segunda, denominada *GSP2P-F*, equivale à estratégia

GSP2-F implementada com as técnicas propostas para poda da base.

O restante desta dissertação está organizado nos seguintes capítulos:

- Capítulo 2 - Algoritmo *GSP*. Neste capítulo, será apresentada uma descrição detalhada do algoritmo *GSP* e uma definição do problema de extração de padrões seqüenciais.
- Capítulo 3 - Técnicas de Redução da Base de Dados. Serão apresentadas, neste capítulo, as técnicas de redução de base de dados propostas e o algoritmo que implementa tais técnicas, chamado *GSP2P* (*GSP2* com Poda).
- Capítulo 4 - Experimentos Computacionais. Avaliações comparativas de desempenho entre os algoritmos *GSP2* e *GSP2P* serão apresentadas neste capítulo. Estão incluídas também avaliações de desempenho de cada técnica de redução de base de dados separadamente.
- Capítulo 5 - Redução de Base de Dados na Extração de Padrões Seqüenciais Baseada em Restrições. Neste capítulo, o problema de extração de padrões seqüenciais baseada em restrições será explorado. O algoritmo proposto, que incorpora as técnicas de redução de base de dados ao algoritmo *GSP-F*, chamado *GSP2P-F*, será apresentado. Avaliações de desempenho também serão realizadas.
- Capítulo 6 - Conclusões. Serão destacadas as contribuições deste trabalho, as conclusões finais e as possibilidades de trabalhos futuros.

Capítulo 2

Algoritmo GSP

Neste capítulo, apresenta-se o algoritmo *GSP*, proposto por *Ramakrishnan Srikant* e *Rakesh Agrawal* em [17]. O *GSP* é um algoritmo iterativo de extração de padrões seqüenciais, e está descrito em detalhes na Seção 2.2. A definição formal do problema de padrões seqüenciais é apresentada na Seção 2.1.

2.1 Padrões Seqüenciais

Um padrão seqüencial é caracterizado por eventos que se sucedem no tempo e que podem ser utilizados para prever um evento futuro baseados nos anteriores. Um exemplo de padrão seqüencial extraído de uma base de dados de uma locadora de vídeos poderia ser [2]: parte significativa dos clientes aluga os seguintes filmes, nesta ordem: "Guerra nas Estrelas", "O Império Contra-Ataca" e "Retorno de Jedi". A representação desse padrão seria <(Guerra nas Estrelas) (O Império Contra-Ataca) (Retorno de Jedi)>. Apesar de as locações aparecerem nesta ordem, elas podem ter sido realizadas de forma não consecutiva. A partir deste exemplo, pode-se prever que um cliente, depois de alugar "Guerra nas Estrelas" e o "Império Contra-Ataca", tem grande probabilidade de alugar o "Retorno de Jedi".

A busca de padrões seqüenciais ocorre sobre uma base de dados transacional, em que cada transação possui o identificador do seu proprietário, chamado consumidor, a informação de quando ela foi realizada e os itens que a caracterizam. Em uma analogia com a base de dados de transações de um supermercado, a identificação do consumidor poderia ser o número do seu cartão de crédito. O momento de realização da transação seria a data e hora da compra e os itens seriam os artigos que foram adquiridos pelo cliente.

A Tabela 2.1 representa uma base de dados de transações de um supermercado, em que

cada registro possui um identificador do consumidor (CID), a identificação de cada transação (TID) e o conjunto de itens comprados. O TID, além de identificar as transações, indica a ordem em que foram realizadas. Deve-se considerar que cada consumidor não possui duas transações realizadas ao mesmo tempo, ou seja, com o mesmo TID.

Tabela 2.1: Exemplo de base de dados transacional.

CID	*TID*	*ITENS*
$Alice$	1	$leite, pão, suco$
$João$	2	$café, manteiga$
$Alice$	3	$ovos, pão$
$Maria$	4	$biscoito, sal$
$João$	5	$laranja, leite$
$Alice$	6	$café, leite$
$Maria$	7	$café, manteiga$
$João$	8	$ovos, pão$

As transações podem ser agrupadas por consumidor, formando uma base de dados de seqüências de consumidores, em que cada seqüência é formada pela lista ordenada (segundo a ocorrência) de transações de um determinado consumidor. Um exemplo de base de seqüências de consumidores, referente à base de dados transacional do exemplo anterior, está apresentado na Tabela 2.2.

Tabela 2.2: Exemplo de base de dados de seqüências de consumidores.

CID	*SEQÜÊNCIA DE CONSUMIDOR*
$Alice$	$\langle(leite, pão, suco)(ovos, pão)(café, leite)\rangle$
$João$	$\langle(café, manteiga)(laranja, leite)(ovos, pão)\rangle$
$Maria$	$\langle(biscoito, sal)(café, manteiga)\rangle$

Conforme definido em [2], uma seqüência é uma lista ordenada de eventos, sendo cada evento um conjunto não vazio de itens. Seja I=$\{i_1, i_2, \ldots, i_m\}$ o conjunto de itens do domínio da aplicação. Um evento e é um conjunto de itens tal que $e \subseteq I$. Considera-se que os itens de cada evento estão ordenados lexicograficamente. Por exemplo, o primeiro conjunto de itens da base de dados apresentada na Tabela 2.1 é $(leite, pão, suco)$.

Uma seqüência s é definida por $\langle e_1, e_2, \ldots, e_n\rangle$, onde cada $e_j, 1 \leq j \leq n$, é um evento. Pode-se chamar e_j de elemento da seqüência. Um exemplo de seqüência de consumidor é a lista de transações do consumidor $Alice$: $\langle(leite, pão, suco)(ovos, pão)(café, leite)\rangle$, na qual cada transação representa um conjunto de itens adquiridos.

O tamanho da seqüência é definido pelo seu número de itens. Por exemplo, a seqüência $\langle(leite, pão, suco)(pão, ovos)(café, leite)\rangle$ tem tamanho 7. Uma seqüência de

tamanho k possui k itens, sendo que um determinado item pode aparecer inúmeras vezes na seqüência, mas uma única vez por elemento da seqüência. Por exemplo, a seqüência anterior possui 7 itens, porém apenas 5 distintos.

Define-se que uma seqüência $s = \langle e_1, e_2, \ldots, e_n \rangle$ é uma subseqüência de outra seqüência $q = \langle q_1, q_2, \ldots, q_m \rangle$, $n \leq m$, se existirem inteiros $i_1 < i_2 < \cdots < i_n$, tais que $e_1 \subseteq q_{i_1}$, $e_2 \subseteq q_{i_2}$, ..., $e_n \subseteq q_{i_n}$. A seqüência $\langle (b)(a,c)(d,e) \rangle$ é uma subseqüência de $\langle (a,b)(e)(a,c,e)(d,e) \rangle$ já que $(b) \subseteq (a,b)$, $(a,c) \subseteq (a,c,e)$ e $(d,e) \subseteq (d,e)$. Se uma seqüência s é uma subseqüência de uma seqüência q, diz-se que s está contida em q.

Padrões seqüenciais são todas as seqüências que estão contidas em um número mínimo preestabelecido de seqüências de consumidores, também chamado de suporte mínimo. Um consumidor suporta uma seqüência s se ela estiver contida na sua seqüência. O suporte da seqüência é definido como a fração do número de consumidores que suportam a seqüência em relação ao número total de consumidores. Seqüências freqüentes, ou padrões seqüenciais, são aquelas que possuem o suporte maior ou igual ao suporte mínimo.

Como exemplo, utilizando a base de dados da Tabela 2.2, as seqüências $\langle (café, manteiga) \rangle$ e $\langle (leite)(pão) \rangle$ são freqüentes para um suporte mínimo de 2 consumidores. A primeira é uma seqüência de tamanho 2 com 1 elemento e está contida nos consumidores $João$ e $Maria$, e a segunda é uma seqüência de tamanho 2 com 2 elementos e está contida nos consumidores $Alice$ e $João$.

Neste trabalho, será estudado o processo de mineração de padrões seqüenciais, ou seja, o processo de identificação de todas as seqüências que possuem suporte maior ou igual ao suporte mínimo, especificado pelo usuário.

2.2 O Algoritmo *GSP*

O algoritmo *GSP* (***G**eneralized **S**equential **P**atterns*) [17] foi uma das primeiras estratégias desenvolvidas para extração de padrões seqüenciais. Este algoritmo extrai os padrões seqüenciais iterativamente, de tal forma que, a cada iteração k, a base de dados é inteiramente lida e são encontradas as seqüências freqüentes de tamanho k.

Em uma primeira iteração, obtém-se o suporte dos conjuntos de itens de tamanho $k = 1$. Os itens que aparecem em um número mínimo de consumidores formam o conjunto de seqüências freqüentes de tamanho $k = 1$, chamado F_1. Em seguida, em cada iteração $k \geq 2$, as possíveis seqüências freqüentes de tamanho k, chamadas seqüências candidatas de tamanho k,

são geradas a partir das seqüências freqüentes de tamanho $k-1$, obtidas na iteração anterior. Ao conjunto de seqüências candidatas de tamanho k, dá-se o nome de C_k. Após a geração de C_k, a base de dados é lida a fim de se obter o suporte das seqüências candidatas de tamanho k e, conseqüentemente, obter o conjunto das seqüências freqüentes de tamanho k, F_k. O algoritmo termina quando nenhuma nova seqüência freqüente é encontrada em uma iteração, ou quando nenhuma seqüência candidata é gerada.

É importante ressaltar que o *GSP* é um algoritmo que não utiliza a base de dados carregada em memória principal, como é o caso de outras estratégias como o *SPADE* [20]. Apenas o conjunto de seqüências candidatas é mantido em memória principal durante cada iteração e apenas uma seqüência de consumidor, por vez, é armazenada em memória principal durante o seu processamento.

Como o conjunto de seqüências candidatas, geradas pela combinação das seqüências freqüentes da iteração anterior, pode se tornar extremamente grande, o *GSP* utiliza as seguintes propriedades para diminuir o espaço de busca, ou seja, diminuir o número de seqüências candidatas a serem avaliadas:

- Toda seqüência que possui alguma subseqüência não freqüente também não é freqüente. Suponha, por exemplo, que a seqüência de tamanho 2 $\langle(leite)(ovos)\rangle$ não seja freqüente. Então, necessariamente, a seqüência candidata de tamanho 3 $\langle(leite)(ovos, pão)\rangle$ também não será freqüente. Essa propriedade permite a redução do número de seqüências candidatas geradas a cada iteração.

- Todas as subseqüências de uma seqüência freqüente são, necessariamente, freqüentes. Por exemplo, se a seqüência de tamanho 3 $\langle(leite)(ovos, pão)\rangle$ é freqüente, então todas as suas subseqüências de tamanho 2 $\langle(leite)(ovos)\rangle$, $\langle(leite)(pão)\rangle$ e $\langle(ovos, pão)\rangle$ também são freqüentes. Esta propriedade permite eliminar as seqüências candidatas que possuem alguma subseqüência que não seja freqüente.

Destacam-se no *GSP* duas funções principais, que ocorrem a cada iteração k:

- Geração de Seqüências Candidatas: no início de cada iteração, o conjunto de seqüências candidatas de tamanho k (C_k) é gerado. A geração é feita em duas etapas: a etapa de Junção, quando ocorre a geração das seqüências candidatas através da combinação das seqüências do conjunto de seqüências freqüentes de tamanho $k-1$ (F_{k-1}); e a etapa de Poda, quando as seqüências candidatas que possuem alguma subseqüência não freqüente são eliminadas. Para armazenar as seqüências candidatas em memória principal é usada

uma árvore *hash* para reduzir o tempo de acesso às seqüências no momento da contagem de suporte.

- Contagem do Suporte das Seqüências Candidatas: esta função é executada após a geração do conjunto C_k. É feita uma leitura completa da base de dados para a contagem do suporte das seqüências de C_k. O suporte de cada candidata é incrementado todas as vezes em que ela estiver contida em uma seqüência de consumidor.

Um pseudo-código do *GSP* é apresentado no Algoritmo 2.1.

```
1  algoritmo GSP (D, suporte_minimo)
2      F_1 = seqüências freqüentes de tamanho 1;
3      k = 1;
4      enquanto (F_k ≠ ∅) faça
5          k = k+1;
6          C_k = candidatas de tamanho k geradas a partir de F_{k-1};
7          para cada seqüência de consumidor t ∈ D faça
8              incrementar o contador das candidatas de C_k contidas em t;
9          fim
10         F_k = todas as candidatas de C_k com suporte ≥ suporte_minimo;
11     fim
12     retorna união de todos os conjuntos de freqüentes F_k;
13 fim
```

Algoritmo 2.1: Pseudo-código do algoritmo *GSP*.

Na primeira iteração do algoritmo, todas as seqüências candidatas de tamanho 1 (todos os itens da base) são contadas através de uma leitura completa da base de dados (D). Aquelas que aparecerem em um número mínimo de seqüências de consumidores formarão o conjunto de seqüências freqüentes de tamanho 1 (F_1) (linha 2). As linhas de 4 a 11 apresentam os passos necessários para se encontrar todas as seqüências freqüentes de tamanho maior do que 1. Na segunda iteração, é feita a geração do conjunto de seqüências candidatas de tamanho 2 (C_2) (linha 6) a partir do conjunto de freqüentes F_1 e, em seguida, é feita uma nova leitura da base de dados, quando cada seqüência de consumidor lida da base é comparada com as seqüências candidatas de C_2 (linhas 7 e 8), armazenadas em uma árvore *hash*. Todas as seqüências candidatas que estiverem contidas em cada seqüência de consumidor têm seu suporte incrementado. Após a leitura da base de dados e da contagem de suporte, as seqüências candidatas que possuem suporte maior ou igual ao suporte mínimo ($suporte\_minimo$) formam o conjunto F_2 (linha 10), que são as seqüências freqüentes de tamanho 2. O conjunto F_2 será usado para gerar o conjunto de seqüências candidatas de tamanho 3 (C_3) na próxima iteração do algoritmo. O algoritmo termina quando nenhuma seqüência candidata em C_k possuir um suporte maior ou igual ao

suporte mínimo, ou seja, quando F_k for vazio. A união de todos os conjuntos de seqüências freqüentes (linha 12), obtidos nas diversas iterações, representa todos os padrões seqüenciais existentes na base de dados para determinado suporte mínimo.

É importante ressaltar que, nas iterações $k \geq 2$, após ler cada seqüência de consumidor t da base de dados, o *GSP* mantém em t apenas os itens freqüentes. Desta forma, na linha 8, cada seqüência de consumidor t possui apenas itens freqüentes.

2.2.1 Geração de Seqüências Candidatas

A geração das seqüências candidatas de tamanho k é feita através da combinação das seqüências freqüentes de tamanho $k-1$.

As seqüências candidatas são geradas em duas fases:

- Fase de junção: as seqüências freqüentes existentes em F_{k-1} são unidas duas a duas para formar as seqüências candidatas de C_k. Esta junção entre duas seqüências freqüentes s_1 e s_2 só é feita se a subseqüência obtida a partir da remoção do primeiro item de s_1 for igual à subseqüência obtida a partir da remoção do último item de s_2. A seqüência candidata gerada é s_1 acrescentada do último item de s_2. O item acrescentado ficará em um elemento separado na seqüência candidata gerada, caso esteja sozinho no último elemento de s_2, caso contrário, o último item estará contido no último elemento de s_1. Por exemplo, as seqüências de tamanho 3, $a = \langle(1,2)(3)\rangle$ e $b = \langle(2)(3)(5)\rangle$, geram a mesma subseqüência $\langle(2)(3)\rangle$ quando se é retirado o primeiro elemento de a e o último elemento de b. Pode-se gerar, então, a seqüência candidata de tamanho 4 $\langle(1,2)(3)(5)\rangle$. A seqüência $a = \langle(1,2)(3)\rangle$ não pode formar candidata com a seqüência $b = \langle(2,3)(4)\rangle$, porque a subseqüência gerada em a, $\langle(2)(3)\rangle$, é diferente da subseqüência gerada em b, $\langle(2,3)\rangle$. Em outro exemplo, a seqüência $a = \langle(1,2)(3)\rangle$ pode unir-se a $b = \langle(2)(3,4)\rangle$, para formar a candidata $\langle(1,2)(3,4)\rangle$.

  O procedimento de junção na segunda iteração é diferente. Cada seqüência de F_1 é combinada com todas as outras seqüências do mesmo conjunto, inclusive com ela mesma, para a geração do conjunto de seqüências candidatas de tamanho 2 (C_2). Se $|F_1| = z$, então existem z^2 combinações possíveis das seqüências freqüentes de tamanho 1, sendo que cada combinação pode gerar até duas diferentes seqüências candidatas. Por exemplo, considerando $\langle(X)\rangle$ e $\langle(Y)\rangle$ seqüências freqüentes de tamanho 1, a primeira candidata gerada possui a forma $\langle(X)(Y)\rangle$, na qual seus dois itens pertencem a elementos diferentes e a segunda candidata gerada possui a forma $\langle(X,Y)\rangle$, na qual seus dois itens pertencem

a um mesmo elemento. Considerando a ordem lexicográfica das seqüências de F_1, a candidata com a forma $\langle(X, Y)\rangle$ somente pode ser gerada, se o item da segunda seqüência a ser combinada é maior lexicograficamente que o item da primeira ($Y > X$). Desta forma, evita-se gerar duas vezes as candidatas equivalentes $\langle(X, Y)\rangle$ e $\langle(Y, X)\rangle$. Além disso, se a seqüência $\langle(X)\rangle$ for combinada com ela mesma, gera-se somente a candidata $\langle(X)(X)\rangle$.

O número de seqüências candidatas da forma $\langle(X)(Y)\rangle$ é determinado por $|F_1| \times |F_1|$ e o número de seqüências candidatas da forma $\langle(X, Y)\rangle$ é determinado por $\binom{|F_1|}{2}$. Desta forma, o número total de seqüências candidatas de tamanho 2 é determinado pela fórmula:

$$|C_2| = (|F_1| \times |F_1|) + (|F_1| \times (|F_1| - 1))/2 \qquad (2.1)$$

- Fase de poda: elimina as seqüências candidatas geradas na fase de junção que possuem alguma subseqüência não freqüente (considerando a propriedade citada anteriormente) e, portanto, não precisam ser armazenadas na árvore *hash* para o processo posterior de contagem. Por exemplo, se na seqüência candidata $a = \langle(1, 2)(3, 5)\rangle$, a subseqüência $\langle(1, 2)(5)\rangle$ não for freqüente, então, obrigatoriamente, a seqüência a também não será freqüente e será podada.

  As candidatas de tamanho 2 não passam por este processo de poda, porque todas as suas subseqüências são necessariamente freqüentes.

Um exemplo de geração de seqüências candidatas de tamanho 4 é apresentado na Tabela 2.3. Na fase de junção, as seqüências freqüentes de tamanho 3 $\langle(1, 2)(3)\rangle$ e $\langle(2)(3, 4)\rangle$ são unidas para formar a seqüência candidata de tamanho 4 $\langle(1, 2)(3, 4)\rangle$, e as seqüências freqüentes $\langle(1, 2)(3)\rangle$ e $\langle(2)(3)(5)\rangle$ são unidas para formar a seqüência candidata $\langle(1, 2)(3)(5)\rangle$. Na fase de poda, as seqüências candidatas que possuem alguma subseqüência não freqüente são descartadas. Sendo assim, somente a seqüência candidata $\langle(1, 2)(3, 4)\rangle$ compõe o conjunto C_4, já que a seqüência $\langle(1, 2)(3)(5)\rangle$ possui as subseqüências não freqüentes $\langle(1)(3)(5)\rangle$ e $\langle(1, 2)(5)\rangle$.

2.2.2 Árvore Hash

Árvore *hash* é a estrutura de dados utilizada para armazenar os conjuntos de seqüências candidatas geradas (C_k), quando o tamanho das seqüências for maior ou igual a 2.

A Figura 2.1 (baseada em uma figura apresentada em [18]) mostra duas árvores *hash*, que armazenam seqüências de tamanho 3. Cada nó da árvore *hash* pode conter uma lista de seqüências (nó folha) ou uma tabela *hash* (nó interno), em que cada entrada aponta para outro nó. O

Tabela 2.3: Geração de seqüências candidatas pelo *GSP*.

Seqüências freqüentes de tamanho 3 (F_3)	*Seqüências candidatas de tamanho 4 após a junção*	*Seqüências candidatas de tamanho 4 após a poda* (C_4)
$\langle(1,2)(3)\rangle$ $\langle(1,2)(4)\rangle$ $\langle(1)(3,4)\rangle$ $\langle(2)(3,4)\rangle$ $\langle(2)(3)(5)\rangle$ $\langle(2,3)(4)\rangle$	$\langle(1,2)(3,4)\rangle$ $\langle(1,2)(3)(5)\rangle$	$\langle(1,2)(3,4)\rangle$

número de entradas na tabela *hash* determina o grau da árvore, que na Figura 2.1 (a) é igual a 3. Um nó folha armazena, de forma ordenada, uma lista de seqüências candidatas, sendo que cada uma possui um contador para armazenar sua freqüência na base de dados. A raiz da árvore *hash* é definida como tendo profundidade 1. Um nó interno de profundidade n aponta para outro nó de profundidade $n + 1$.

A partir do nó raiz, as seqüências candidatas geradas são adicionadas à árvore *hash*. Percorre-se a árvore *hash* até alcançar um nó folha. O caminho percorrido através dos nós internos até alcançar um nó folha é determinado pelo resultado de uma função *hash* aplicada sobre os itens da seqüência. Quando estiver percorrendo a árvore *hash* através de um nó interno de profundidade n, deve-se aplicar a função *hash* sobre o item de ordem n da seqüência. O retorno desta função informa qual ramo do nó será alcançado.

Inicialmente, quando a árvore *hash* é criada, o nó raiz é o único existente e é também um nó folha. No momento em que o número de seqüências em cada nó folha ultrapassar um limite preestabelecido, chamado saturação da folha, e sua profundidade for menor do que k (tamanho das seqüências inseridas na árvore *hash*), o nó folha transforma-se em nó interno. Nós filhos do novo nó interno são criados para receber as seqüências, que são distribuídas entre eles através da aplicação da função *hash* sobre os itens das seqüências. Se o nó folha estiver no último nível, igual a k, ele receberá a seqüência independente do número de seqüências já armazenadas.

Na Figura 2.1 (a), à esquerda, é apresentada a função *hash*. Considerando-se que os itens que compõem a base de dados estejam no intervalo numérico [1,9], a partir do nó raiz, a função *hash* é aplicada sobre cada item da seqüência. A função *hash* é definida da seguinte forma: se o item for 1, 4 ou 7, percorre-se a árvore pelo nó esquerdo; se for 2, 5 ou 8, percorre-se a árvore pelo nó central e se for 3, 6 ou 9, percorre-se a árvore pelo nó direito. Como exemplo, utilizando a árvore *hash* da Figura 2.1 (a) para localizar a seqüência $s = \langle(4,5)(1)\rangle$, a função *hash* deve ser aplicada sobre seus itens. Inicia-se pelo nó raiz. A função *hash* é aplicada no primeiro item da seqüência s, alcançando o nó esquerdo da raiz. Percorre-se a árvore *hash* pelo nó esquerdo

e outro nó interno, com profundidade 2, é alcançado. Então, a função *hash* deve ser aplicada sobre o segundo item da seqüência s, tendo como resultado o ramo central. Percorre-se a árvore *hash* pelo ramo central e um nó folha é alcançado, onde poderá ser encontrada a seqüência s.

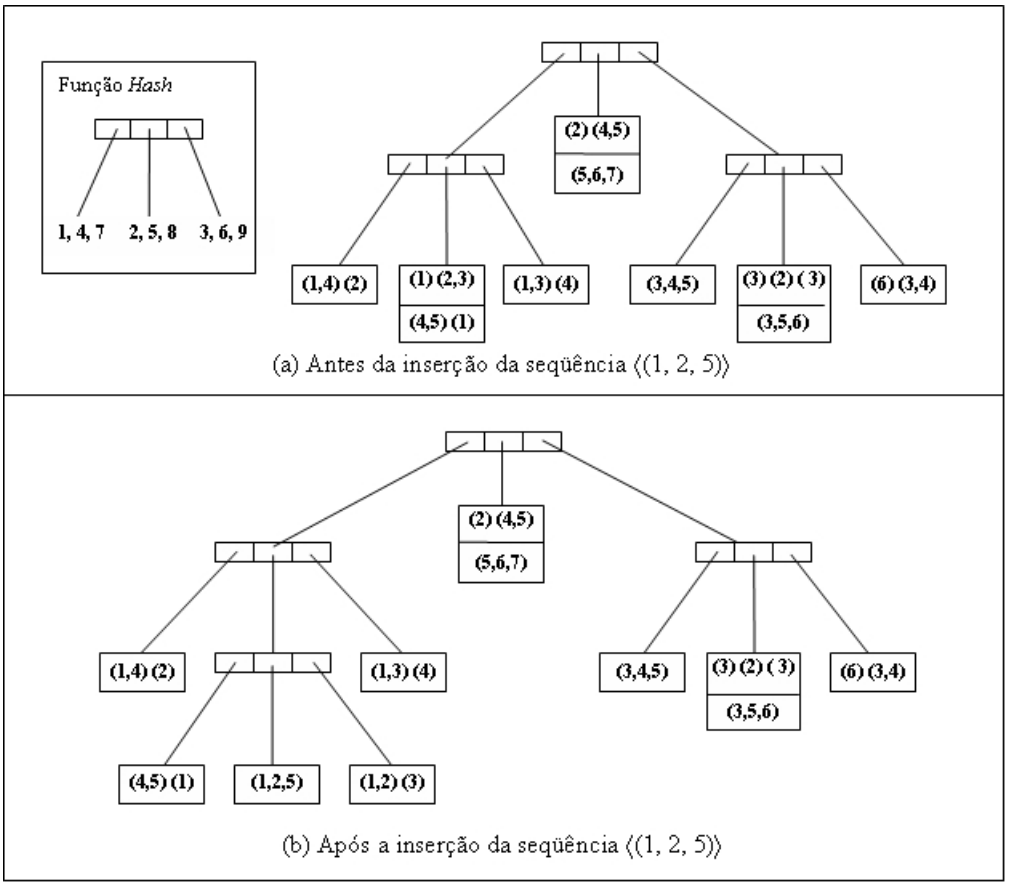

Figura 2.1: Exemplo de árvore *hash*.

Na inserção da seqüência $\langle(1, 2, 5)\rangle$ na árvore *hash* apresentada na Figura 2.1 (a), a função *hash* é aplicada recursivamente sobre seus itens até encontrar um nó folha. O nó folha encontrado possui duas seqüências: $\langle(1)(2, 3)\rangle$ e $\langle(4, 5)(1)\rangle$. Como a saturação da folha é 2 e a profundidade é menor do que 3, este nó folha será transformado em nó interno de profundidade 3 e suas seqüências serão distribuídas entre seus nós filhos, aplicando a função *hash* sobre o terceiro item de cada uma das seqüências, conforme ilustrado na Figura 2.1 (b).

2.2.3 Contagem do Suporte das Seqüências Candidatas

A cada iteração k do algoritmo, após a geração das seqüências candidatas, cada seqüência de consumidor t da base de dados é lida e incrementa-se o suporte de todas as seqüências candidatas de C_k, armazenadas na árvore *hash*, contidas em t. Após todas as seqüenciais de consumidores serem analisadas, gera-se F_k, eliminando-se as seqüências candidatas que não

estiverem contidas em um número mínimo de consumidores.

A busca na árvore *hash* pelas seqüências candidatas contidas em cada seqüência de consumidor t é feita pela utilização de uma função *hash* que é aplicada sobre os itens de t. Na k-ésima iteração, cada subseqüência t' de tamanho k contida em t deve ser considerada. Percorre-se a árvore *hash* na tentativa de se encontrar alguma seqüência candidata igual a t' para ter seu suporte incrementado. Sendo assim, quanto maior o tamanho de t, maior será o número de subseqüências a serem avaliadas nesse processo.

A Figura 2.2 apresenta um exemplo de contagem do suporte das seqüências candidatas contidas na seqüência de consumidor $t = \langle(1,2)(3)(4,5)\rangle$. Os nós internos estão rotulados para melhor exemplificação. A árvore *hash* deste exemplo armazena candidatas de tamanho 3. O resultado da função *hash* aplicada sobre os itens de t indica qual caminho será percorrido do nó raiz até um nó folha.

A função *hash* será aplicada a todos os itens a partir do nó raiz. Inicialmente, ela é aplicada sobre o primeiro item de t. As candidatas que possuem o item 1, como o primeiro item da seqüência, estão na sub-árvore esquerda da raiz. Como o nó esquerdo da raiz ($N1$) é um nó interno, a função *hash* também será aplicada sobre cada um dos itens posteriores a 1 na seqüência de consumidor, para se chegar até um nó folha. O resultado da função *hash* sobre o item 2 aponta para o nó $N3$. Nesta sub-árvore central do nó $N1$, podem existir seqüências que iniciam com os itens 1 e 2. Como o nó $N3$ é um nó interno, a função *hash* será aplicada sobre os itens posteriores a 1 e 2, que são $(3)(4,5)$. Ao se aplicar a função *hash* sobre o item 3, um nó folha é alcançado. As seqüências candidatas encontradas neste nó folha são $\langle(1,2)(3)\rangle$ e $\langle(4)(2,3)\rangle$. Todas as seqüências candidatas existentes no nó folha são comparadas com a seqüência de consumidor t, para a contagem do suporte. Somente a seqüência $\langle(1,2)(3)\rangle$ teve seu suporte incrementado, pois é uma subseqüência da seqüência de consumidor.

Quando um nó folha é alcançado, ele ficará marcado para aquela seqüência de consumidor, evitando que seqüências candidatas sejam comparadas mais de uma vez com a mesma seqüência de consumidor.

Como a seqüência de consumidor ainda não foi completamente processada na busca de candidatas que iniciam com os itens 1 e 2, a função *hash* é aplicada, a partir do nó $N3$, sobre o item 4 e, posteriormente, sobre o item 5. Ao aplicar a função *hash* sobre o item 4, uma outra folha é alcançada. A única seqüência $\langle(4,5)(1)\rangle$ deste nó folha, como não está contida na seqüência de consumidor, não tem o seu suporte incrementado. Ao aplicar a função *hash* sobre o item 5, uma outra folha é alcançada. A única seqüência $\langle(1,2)(5)\rangle$ deste nó folha tem o seu suporte incrementado.

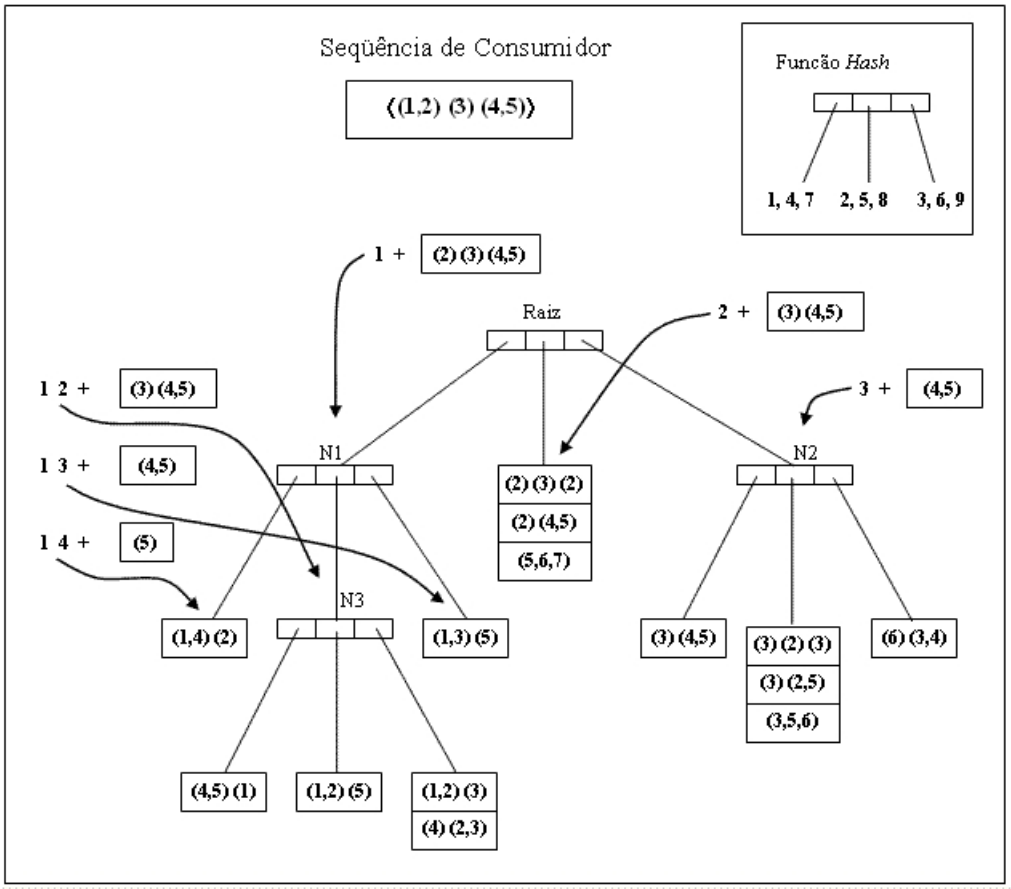

Figura 2.2: Exemplo da contagem do suporte das seqüências candidatas.

Como a seqüência de consumidor foi completamente processada na busca de seqüências candidatas que iniciam com os itens 1 e 2, o procedimento retorna ao nó interno $N1$ para alcançar seqüências candidatas através da aplicação da função *hash* sobre o item 3. Aplicando-se a função *hash* sobre este item, um nó folha é alcançado e sua única seqüência $\langle(1,3)(5)\rangle$ não tem o seu suporte incrementado. A função *hash* é aplicada aos itens subseqüentes da seqüência de consumidor até que todas as possíveis combinações de tamanho 3, que começam com o item 1, tenham sido exploradas. O procedimento recursivo, que percorre a árvore *hash*, retorna ao nó raiz e a função *hash* é aplicada aos itens restantes da seqüência de consumidor $(2)(3)(4,5)$, criando todas as combinações possíveis de tamanho 3. Ao aplicar a função *hash* no segundo item da seqüência de consumidor, item 2, serão verificadas seqüências candidatas de tamanho 3, iniciadas com o item 2. O processo segue para os itens posteriores da seqüência de consumidor $(3)(4,5)$, como ilustra a Figura 2.2.

A contagem é finalizada quando todas as seqüências de consumidores existentes na base de dados são comparadas com as seqüências candidatas armazenadas na árvore *hash*. Após a contagem, as seqüências que tiverem os suportes menores do que o mínimo são eliminadas da árvore *hash*.

2.2.4 *GSP2* - Uma Implementação do *GSP*

O código do algoritmo *GSP* utilizado neste trabalho, chamado aqui de *GSP2*, foi escrito na linguagem C e cedido pelo autor da dissertação de mestrado *Mineração Eficiente de Padrões Seqüenciais através da Indexação de Seqüências Candidatas* [6]. O algoritmo *GSP2* é uma adaptação do algoritmo *GSP* original que utiliza uma estrutura de dados baseada em vetor. Esta estrutura permite a indexação direta das seqüências candidatas na segunda iteração em vez da utilização da árvore *hash*. As demais iterações são executadas de forma idêntica ao algoritmo original.

Os parâmetros de execução do algoritmo *GSP2* são: base de dados, suporte mínimo, grau da árvore hash, saturação da folha, o número de diferentes itens existentes na base de dados e a memória disponível para sua execução.

2.2.4.1 Gerencia de Memória

Uma das características do algoritmo *GSP* é a geração de um grande número de seqüências candidatas. Por exemplo, na segunda iteração do algoritmo, se houver 1.000 seqüências freqüentes de tamanho 1, serão geradas, de acordo com a Fórmula 2.1, $(1.000 \times 1.000) + (1.000 \times 999)/2 = 1.499.500$ seqüências candidatas. Ao usar um suporte mínimo baixo, as seqüências candidatas em uma determinada iteração podem não caber na memória principal, ocasionando o acesso a disco para o processo de contagem, tornando-o muito lento. Desta forma, na implementação do *GSP2*, desenvolveu-se uma estratégia para gerenciamento de memória.

Considerando-se uma determinada quantidade de memória reservada para a execução da aplicação, o *GSP2* vai alocando estruturas de dados e armazenando as seqüências candidatas. A quantidade de memória reservada é constantemente monitorada, para que não seja ultrapassado o limite preestabelecido. Se todo o conjunto de seqüências candidatas não couber em memória, ele deverá ser particionado e a contagem do suporte será realizada separadamente sobre cada uma destas partes, até que todo o conjunto de candidatas seja avaliado. Em cada processo de contagem de suporte, aplicado às partições do conjunto de seqüências candidatas, é feita uma leitura completa da base de dados.

Capítulo 3

Técnicas de Redução da Base de Dados

Neste capítulo, são apresentadas as técnicas de redução da base de dados propostas e o algoritmo que implementa tais técnicas, chamado *GSP2P* (*GSP2 com Poda*). As técnicas de redução são descritas na Seção 3.1. O algoritmo *GSP2P* é apresentado em detalhes na Seção 3.2. Parte do conteúdo deste capítulo foi apresentado em [4].

3.1 Técnicas de Poda Propostas

A redução da base de dados tem como objetivo diminuir o custo de várias leituras da base de dados e o esforço computacional da fase de contagem das seqüências candidatas, típicos de algoritmos iterativos de extração de padrões seqüenciais. Será garantida, contudo, a extração correta de todos os padrões seqüências existentes na base de dados.

As técnicas propostas são baseadas nas estratégias de redução de base de dados apresentadas pelos autores dos algoritmos *DHP* [12] e *kDCI++* [11] para o problema de mineração de conjuntos freqüentes e adaptadas aqui para o problema de extração de padrões seqüenciais.

Duas estratégias de redução da base de dados são propostas: a *poda local* da base, adaptação da poda local proposta pelos autores do algoritmo *DHP* e também utilizada pelo algoritmo *kDCI++*, e a *poda global* da base, adaptação da poda global proposta pelos autores do algoritmo *kDCI++*. No algoritmo *GSP2P* proposto, as técnicas de poda global e local são aplicadas a cada seqüência de consumidor t, respectivamente, antes e depois da contagem de suporte das seqüências candidatas presentes em t. O objetivo principal é reduzir o número de itens presentes em t e, conseqüentemente, reduzir o número de seqüências presentes em t que serão analisadas durante o processo de contagem de suporte. Uma extensão das podas local e global, chamada de *poda de itens isolados*, é também proposta. A poda de itens isolados tem como objetivo

reduzir ainda mais o tamanho das seqüências de consumidores, removendo itens desnecessários que aparecem isolados em elementos da seqüência. Uma nova base de dados é escrita em disco ao final de cada iteração e utilizada na iteração seguinte.

A seguir, as técnicas de poda são apresentadas separadamente. As descrições das podas local e global são baseadas naquelas apresentadas em [11], porém adaptadas para o problema de extração de padrões seqüenciais.

3.1.1 Poda Local

A poda local é realizada em cada iteração $k \geq 3$, durante a contagem de suporte das seqüências candidatas em C_k. Para cada seqüência de consumidor t da base, após a análise de todas as seqüências candidatas em t, são determinados os itens em t que necessariamente não serão úteis na próxima iteração, ou seja, os itens que não contribuirão na contagem de uma seqüência freqüente e que poderão ser removidos de t. Desta forma, reduz-se o tamanho da seqüência de consumidor t, podendo até mesmo ser eliminada por completo da base de dados.

A técnica é baseada nas seguintes propriedades:

- Propriedade 1: Uma seqüência s de tamanho k presente em uma seqüência de consumidor t da base de dados poderá ser freqüente somente se todas as suas subseqüências de tamanho $k-1$ forem freqüentes, ou seja, pertencerem a F_{k-1}.
- Propriedade 2: Em uma seqüência s de tamanho k, existem k subseqüências de tamanho $k-1$ e cada item de s aparece em exatamente $k-1$ dessas k subseqüências.

Considere, por exemplo, a seqüência $s = \langle(a,b)(c)(a)\rangle$, de tamanho $k = 4$. Observe que s será uma seqüência freqüente somente se todas as suas subseqüências de tamanho 3, $s_1 = \langle(a,b)(c)\rangle$, $s_2 = \langle(a,b)(a)\rangle$, $s_3 = \langle(a)(c)(a)\rangle$ e $s_4 = \langle(b)(c)(a)\rangle$, forem seqüências freqüentes (Propriedade 1). Note que há 4 (k) subseqüências de tamanho 3 em s e cada item em s aparece em apenas 3 ($k-1$) destas (Propriedade 2). O item a, primeiro item de s, apareceu nas subseqüências s_1, s_2, e s_3, o item b apareceu nas subseqüências s_1, s_2 e s_4, o item c apareceu nas subseqüências s_1, s_3 e s_4, e o item a, último item de s, apareceu nas subseqüências s_2, s_3 e s_4.

Levando-se em consideração as propriedades descritas acima, pode-se concluir que, na k-ésima iteração, se um item i pertencente a uma seqüência de consumidor t não contribuir na contagem de suporte de pelo menos k seqüências candidatas de tamanho k, então este item i, na seqüência de consumidor t especificamente, não será útil na contagem de suporte de nenhuma

seqüência freqüente de tamanho $k+1$ na iteração seguinte. Desta forma, o item i poderá ser removido de t gerando uma nova seqüência de consumidor t', que irá compor a nova base de dados que será usada na próxima iteração.

Sendo assim, a poda local é aplicada a cada seqüência de consumidor da base de dados corrente da seguinte maneira. Durante o processo de contagem de suporte das seqüências candidatas de tamanho k, no processamento de cada seqüência de consumidor t da base, é criado um contador para cada item de t, ou seja, $|t|$ contadores, e, quando uma seqüência candidata s está presente em t e tem seu suporte incrementado, os contadores dos itens de t que contribuem na contagem de s também são incrementados. O conjunto destes contadores é chamado L_t. Desta forma, ao final do processamento da seqüência de consumidor t (após a análise de todas as seqüências candidatas em t), temos em L_t o número de vezes que cada item de t contribuiu na contagem de suporte das seqüências candidatas de tamanho k. Pode-se assim eliminar de t os itens que não contribuem na contagem de suporte de pelo menos k seqüências candidatas.

Caso o tamanho da nova seqüência de consumidor t' seja menor do que $k+1$, t' não será gravada na nova base de dados, D_{k+1}, e será desconsiderada na contagem de suporte das seqüências candidatas de tamanho $k+1$, já que não poderá contribuir para contagem de suporte de tais candidatas.

A Figura 3.1 apresenta um exemplo da aplicação da poda local na seqüência de consumidor $t = \langle(a,b)(b)(c)(a,e)\rangle$ na iteração $k = 3$. Para a contagem dos itens de t é criado o conjunto L_t de contadores, um para cada item em t. Durante o processo de contagem de suporte das seqüências candidatas em C_3, cada candidata é testada contra t a fim de ter o seu suporte incrementado. As seqüências candidatas contidas em t, $s_1 = \langle(a,b)(c)\rangle$, $s_2 = \langle(a,b)(a)\rangle$, $s_3 = \langle(a)(c)(a)\rangle$ e $s_6 = \langle(b)(a,e)\rangle$, estão destacadas em negrito. Toda vez que o suporte de uma seqüência candidata é incrementado, os respectivos contadores dos seus itens são atualizados. Assim, observa-se que a primeira ocorrência do item a em t contribuiu na contagem de três destas seqüências candidatas (s_1, s_2 e s_3), a primeira ocorrência do item b em t contribuiu na contagem de três (s_1, s_2 e s_6), a segunda ocorrência do item b, terceiro item de t, contribuiu na contagem de uma (s_6), o item c, na contagem de duas (s_1 e s_3), a segunda ocorrência do item a, quinto item de t, na contagem de três (s_2, s_3 e s_6) e o item e, na contagem de uma candidata (s_6). Como a segunda ocorrência de b (terceiro item de t), o item c e o item e contribuíram na contagem de suporte de menos de três seqüências candidatas, eles foram excluídos da nova seqüência de consumidor t'. E como a seqüência de consumidor t' de tamanho 3 não poderá contar o suporte das seqüência candidata de tamanho 4 ($k+1$), t' não irá compor a nova base de dados e será descartada por completo.

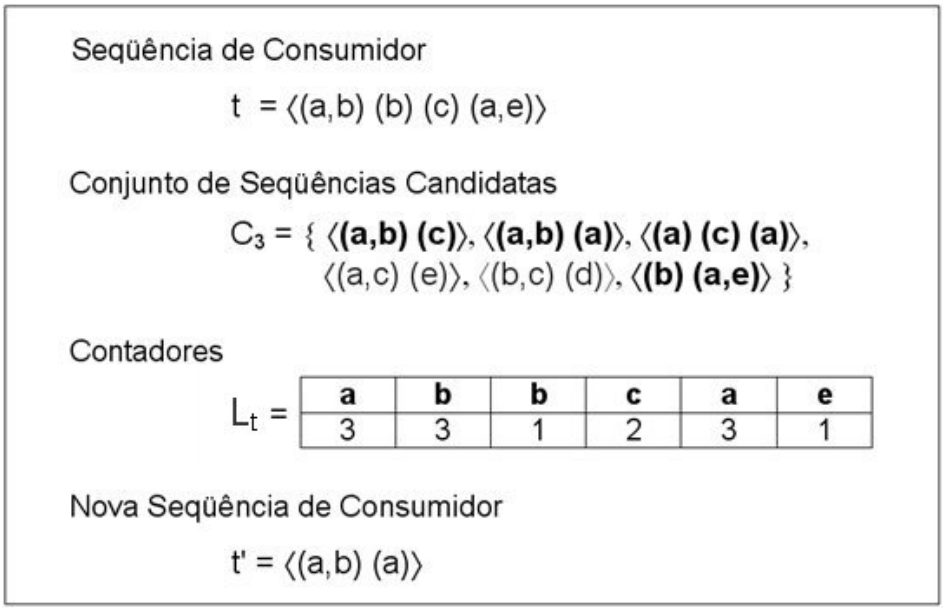

Figura 3.1: Exemplo de aplicação da poda local.

No exemplo anterior, nota-se que a seqüência candidata $s_6 = \langle(b)(a,e)\rangle$ está presente na seqüência de consumidor $t = \langle(a,b)(b)(c)(a,e)\rangle$ de duas formas: a primeira, quando o primeiro elemento de s_6, (b), está contido no primeiro elemento de t, (a,b), e o segundo elemento de s_6, (a,e), está contido no último elemento de t, (a,e), e a segunda forma, quando o primeiro elemento de s_6, (b), está contido no segundo elemento de t, (b), e o segundo elemento de s_6, (a,e), está contido no último elemento de t, (a,e). Logo, os contadores das duas ocorrências de b, da segunda ocorrência de a e da única ocorrência de e são incrementados. Observe que, no processo de contagem de suporte, basta conhecer a primeira ocorrência de uma seqüência candidata s em t para que s tenha o seu suporte incrementado. Já no processo de poda local, é necessário conhecer todas as ocorrências de s em t para que os contadores L_t sejam incrementados. Esta verificação adicional aumenta muito o esforço computacional da fase de contagem de suporte, o que compromete o objetivo principal da aplicação das estratégias de redução de base de dados, que é a redução do custo computacional da fase de contagem.

Sendo assim, propõe-se uma segunda maneira de se realizar a poda local. Durante o processo de contagem de suporte das seqüências candidatas de tamanho k, no processamento de cada seqüência de consumidor t da base, é criado um contador para cada item distinto de t e, quando uma seqüência candidata s está presente em t e tem seu suporte incrementado, os contadores dos itens distintos de t que compõem a seqüência candidata s também são incrementados. Sendo que, no caso de o item aparecer em mais de um elemento da seqüência candidata s, o seu respectivo contador é incrementado apenas uma vez. Pode-se assim eliminar de t todas as ocorrências dos itens que não contribuem na contagem de suporte de pelo menos k seqüências

candidatas. Nesta proposta alternativa, a poda local deixa de eliminar alguns itens que poderiam ser excluídos de t, porém não requer processamento adicional no processo de contagem de suporte.

A Figura 3.2 apresenta um exemplo da aplicação da poda local modificada na seqüência de consumidor $t = \langle(a,b)(b)(c)(a,e)\rangle$ na iteração $k = 3$. Para a contagem dos itens de t é criado um contador para cada item distinto em t (L_t). Durante o processo de contagem de suporte das seqüências candidatas, cada seqüência de C_3 é testada contra t a fim de ter o seu suporte incrementado. As seqüências candidatas contidas em t, $s_1 = \langle(a,b)(c)\rangle$, $s_2 = \langle(a,b)(a)\rangle$, $s_3 = \langle(a)(c)(a)\rangle$ e $s_6 = \langle(b)(a,e)\rangle$, estão destacadas em negrito. Toda vez que o suporte de uma seqüência candidata é incrementado, os respectivos contadores dos seus itens são atualizados. Assim, observa-se que o item a apareceu em quatro destas seqüências candidatas (s_1, s_2, s_3 e s_6), o item b, em três (s_1, s_2 e s_6), o item c, em duas (s_1 e s_3) e o item e, em uma (s_6). Como os itens c e e apareceram em menos de três seqüências candidatas, eles foram excluídos da nova seqüência de consumidor t'. E como a seqüência de consumidor t' de tamanho 4 poderá contar o suporte das seqüência candidata de tamanho 4 ($k + 1$), t' irá compor a nova base de dados.

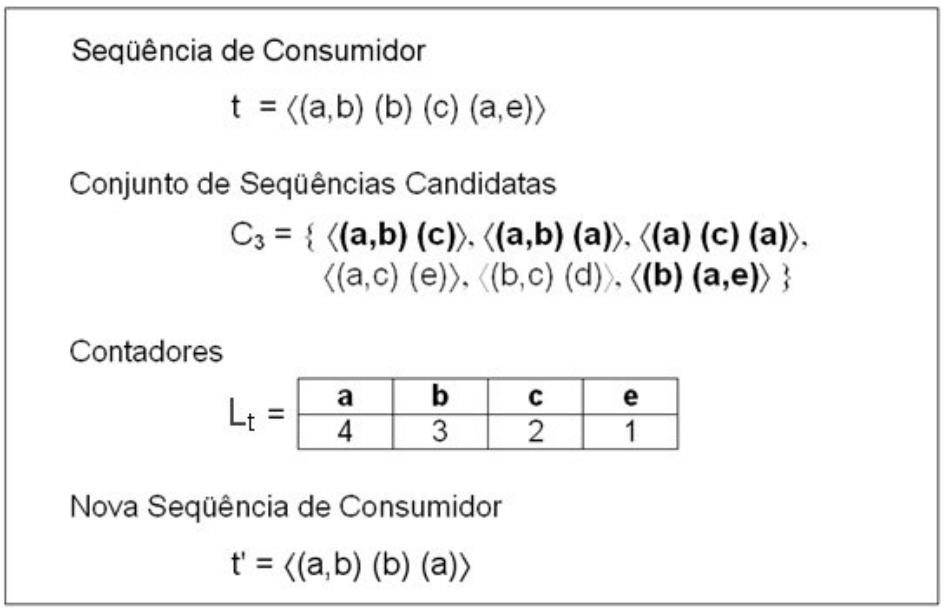

Figura 3.2: Exemplo de aplicação da poda local modificada.

Sendo assim, daqui pra frente, entende-se como poda local a poda local modificada.

A poda local só é realizada a partir da terceira iteração. Isso porque, na segunda iteração, não existe a possibilidade de redução do tamanho das seqüências de consumidores utilizando-se a poda local. Antes de uma seqüência de consumidor t ser analisada, conforme visto no final da Seção 2.2, o *GSP* retira de t os itens que não são freqüentes. Observa-se ainda que, sendo C_2 formado pela combinação de todos os itens freqüentes, qualquer subseqüência de tamanho

2 em t estará contida em C_2. Logo, cada item i freqüente em t vai contribuir na contagem de pelo menos duas seqüências candidatas de C_2, já que o item i estará presente em mais de uma subseqüência de tamanho 2 em t (considerando $|t| > 2$), e não haverá redução do tamanho de t.

3.1.2 Poda Global

Enquanto a poda local verifica se um item é necessário em uma seqüência de consumidor específica, a poda global verifica se um item pode ser removido por completo da base de dados, ou seja, removido de todas as seqüências de consumidores.

A poda local é aplicada a cada seqüência de consumidor após a sua utilização na contagem de suporte. Já a poda global é aplicada a cada seqüência de consumidor imediatamente antes da contagem de suporte, removendo-se os itens desnecessários para a iteração k corrente, sendo $k \geq 2$. A partir das Propriedades 1 e 2 vistas anteriormente, observa-se que, na k-ésima iteração, se um item i não estiver presente em pelo menos $k-1$ seqüências freqüentes de tamanho $k-1$, então i não aparecerá em nenhuma seqüência freqüente de tamanho k. Desta forma, o item i poderá ser removido de todas as seqüências de consumidores da base.

Sendo assim, antes do processo de contagem de suporte das seqüências candidatas de tamanho k presentes em uma seqüência de consumidor t, para cada item i em t, i será eliminado caso apareça em menos de $k-1$ seqüências freqüentes de F_{k-1}. Caso o tamanho da nova seqüência de consumidor t' seja menor do que k, t' será desconsiderada no processo de contagem de suporte das seqüências candidatas de tamanho k, já que não poderá contribuir para a contagem de suporte de tais candidatas, e não será gravada na nova base de dados D_{k+1}.

Para se descobrir a quantidade de seqüências freqüentes de F_{k-1} em que cada item da base de dados aparece, é realizada uma contagem dos itens durante o processo de geração das seqüências candidatas no passo corrente, onde todas as seqüências freqüentes de tamanho $k-1$ são acessadas. Quando uma seqüência freqüente s é lida, cada item de s tem o seu contador atualizado. O conjunto destes contadores é chamado G_{k-1}. Ao final do processo, obtém-se a quantidade de seqüências freqüentes de F_{k-1} em que cada item da base de dados aparece.

Na Figura 3.3, é apresentado um exemplo da aplicação da poda global na iteração $k = 4$. Durante o processo de geração das seqüências candidatas, os contadores para cada item do domínio $\{a, b, c, d, e, f\}$ são gerados e incrementados. As seqüências freqüentes em F_3 são: $f_1 = \langle(a, b)(c)\rangle$, $f_2 = \langle(a, b)(a)\rangle$, $f_3 = \langle(a)(c)(a)\rangle$, $f_4 = \langle(b)(a, e)\rangle$ e $f_5 = \langle(c)(d, e)\rangle$. Assim, observou-se que o item a apareceu em quatro destas seqüências freqüentes (f_1, f_2, f_3 e f_4), o item b, em três (f_1, f_2 e f_4), o item c, em três (f_1, f_3 e f_5), o item d, em uma (f_5), o item e, em

duas (f_4 e f_5), e o item f, em nenhuma. Desta forma, os itens d, e e f, que aparecem em menos de 3 ($k-1$) seqüências freqüentes, podem ser eliminados das seqüências de consumidores da base de dados. Retirando tais itens da seqüência de consumidor $t = \langle(a,b,e)(a)(e)\rangle$, obtém-se $t' = \langle(a,b)(a)\rangle$. E como a seqüência de consumidor t' de tamanho 3 não poderá contar o suporte das seqüência candidata de tamanho 4 (k), t' não irá compor a nova base de dados e será descartada por completo.

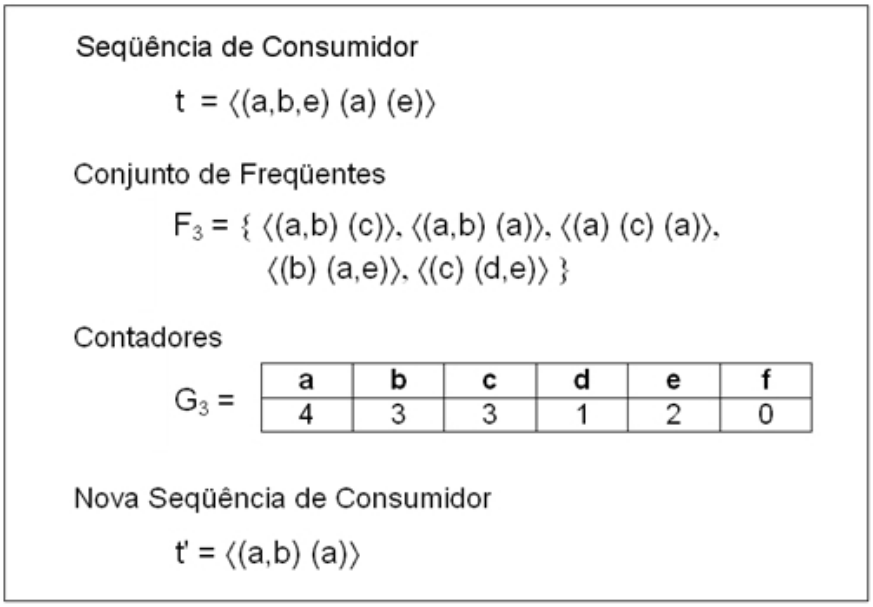

Figura 3.3: Exemplo de aplicação da poda global.

A aplicação da poda global sobre as seqüências de consumidores pode ser feita a partir da segunda iteração, porém, por ter efeito pequeno nesta iteração, somente a partir da terceira a poda global é aplicada, gerando-se uma nova base de dados. Isto porque, de acordo com experimentos computacionais realizados, a redução da base de dados na segunda iteração é pequena e o ganho com a leitura de uma base menor na iteração seguinte não compensa o tempo gasto para a sua gravação em disco ao final da segunda iteração.

3.1.3 Poda de Itens Isolados

Um item i pertencente a uma seqüência s é considerado isolado se i é o único item do elemento de s ao qual pertence. Por exemplo, na seqüência $s = \langle(a,b)(c)(d)(e,f)\rangle$, os itens c e d são os únicos que aparecem isolados.

O processo de poda de itens isolados não é considerado um processo de poda por si só, trata-se de uma extensão que pode ser aplicada às podas local e global, com o objetivo de reduzir ainda mais o tamanho das seqüências de consumidores. A técnica é baseada na Propriedade 3,

apresentada a seguir. Tal propriedade é derivada da Propriedade 2, descrita na Seção 3.1.1:

- Propriedade 3: Em uma seqüência s de tamanho k, existem k subseqüências de tamanho $k-1$ e cada item que aparece isolado em elementos de s, além de aparecer em $k-1$ dessas k subseqüências, conforme a Propriedade 2, aparece também isolado nestas subseqüências.

Considere, por exemplo, a seqüência $s = \langle(a,b)(c)(d)\rangle$, de tamanho $k = 4$. As subseqüências de tamanho 3 de s são: $s_1 = \langle(a,b)(c)\rangle$, $s_2 = \langle(a,b)(d)\rangle$, $s_3 = \langle(a)(c)(d)\rangle$ e $s_4 = \langle(b)(c)(d)\rangle$. Note que há 4 ($k$) subseqüências de tamanho 3 em s e cada item que aparece isolado em s, aparece também isolado em apenas 3 ($k-1$) destas (Propriedade 3). O item c, que aparece isolado no segundo elemento de s, também aparece isolado nas subseqüências s_1, s_3 e s_4, e o item d, que aparece isolado no terceiro elemento de s, aparece também isolado nas subseqüências s_2, s_3 e s_4.

Levando-se em consideração a Propriedade 3, pode-se concluir que, na k-ésima iteração, se um item i que aparece isolado em uma seqüência de consumidor t não contribuir na contagem de suporte de pelo menos k seqüências candidatas de tamanho k, então i, em t especificamente, não será útil na contagem de suporte de nenhuma seqüência freqüente de tamanho $k+1$ na iteração seguinte e poderá ser removido de t gerando uma nova seqüência de consumidor t'. Esta seqüência t' irá compor a nova base de dados que será usada na próxima iteração.

Sendo assim, a poda local estendida com a poda de itens isolados é aplicada a cada seqüência de consumidor da base de dados corrente da seguinte maneira. Durante o processo de contagem de suporte das seqüências candidatas de tamanho k, no processamento de cada seqüência de consumidor t da base, são criados dois conjuntos de contadores: L_t e L'_t. O conjunto de contadores L_t contém um contador para cada item distinto de t (pois a segunda alternativa de poda local está sendo adotada). O conjunto de contadores L'_t é utilizado na contagem das ocorrências isoladas (nas seqüências candidatas) dos itens de t. Ou seja, o contador de um item de t deverá refletir o número de seqüências candidatas em que este item aparece de forma isolada. Quando uma seqüência candidata s está presente em t e tem seu suporte incrementado, os contadores em L_t dos itens distintos de t que contribuem na contagem de s também são incrementados. Adicionalmente, os contadores em L'_t são incrementados quando tais itens aparecem isolados em s. Desta forma, ao final do processamento da seqüência de consumidor t (após a análise de todas as seqüências candidatas em t), temos em L_t o número de vezes que cada item distinto de t contribuiu na contagem de suporte das seqüências candidatas de tamanho k, e em L'_t o número de vezes que cada item distinto de t contribuiu, de forma isolada, na contagem

de suporte das seqüências candidatas de tamanho k. Pode-se assim eliminar de t: (a) os itens que possuem os respectivos contadores em L_t com valores inferiores a k, e (b) os itens que aparecem isolados e que possuem os respectivos contadores em L'_t com valores inferiores a k. São eliminados então: (a) os itens que aparecem não isolados em t e que não contribuem, de forma isolada e não isolada, na contagem de suporte de pelo menos k seqüências candidatas, e (b) os itens que aparecem isolados em t e que não contribuem, de forma isolada, na contagem de suporte de pelo menos k seqüências candidatas.

É importante ressaltar que, quando se elimina um item de um elemento e de tamanho 2 de t e e passa a conter apenas um item, ou seja, quando um item i que antes aparecia não isolado em e passa então a ficar isolado, uma segunda verificação faz-se necessária (caso i já tenha sido verificado). Essa segunda verificação irá remover o item i, se i não contribui, de forma isolada, na contagem de pelo menos k seqüências candidatas.

Caso o tamanho da nova seqüência de consumidor t' seja menor do que $k+1$, t' não será gravada na nova base de dados, D_{k+1}, e será desconsiderada na contagem de suporte das seqüências candidatas de tamanho $k+1$, já que não poderá contribuir para contagem de suporte de tais candidatas.

A Figura 3.4 apresenta um exemplo de aplicação da poda de itens isolados juntamente com processo de poda local sobre a seqüência de consumidor $t = \langle(a,b)(b)(c)(a,e)\rangle$ na iteração $k = 3$. Durante o processo de contagem de suporte, quando as seqüências candidatas contidas em t, $s_1 = \langle(a,b)(c)\rangle$, $s_2 = \langle(a,b)(a)\rangle$, $s_3 = \langle(a)(c)(a)\rangle$ e $s_6 = \langle(b)(a,e)\rangle$, destacadas em negrito, têm os seus suportes incrementados, os contadores L_t e L'_t são atualizados. Em L'_t, os contadores relacionados aos itens que aparecem isolados nas seqüências candidatas são incrementados. Assim, observa-se que o item a apareceu isolado em duas destas seqüências candidatas (s_2 e s_3), o item b, em uma (s_6) e os itens c e e, em nenhuma seqüência candidata. Como os itens isolados b e c de t aparecem também isolados em menos de três seqüências candidatas, eles são excluídos da nova seqüência de consumidor t'. O item e, que não aparece isolado, foi removido pelos critérios da poda local, pois aparece em menos de três seqüências candidatas, isoladamente ou não. Note que, após a remoção do item e pertencente ao último elemento de t, o item a passou a ficar isolado e, como a só aparece em duas seqüências candidatas também isolado, então esta ocorrência do item a isolado pode ser removida. Na utilização da poda local com a poda de itens isolados, t' tem tamanho 2 e não poderá contribuir na contagem de suporte das seqüência candidata de tamanho 4 ($k+1$). Logo, t' não irá compor a nova base de dados e será descartada por completo. Utilizando-se a poda local sem a poda de itens isolados, t' teria tamanho 4 e seria utilizada na iteração seguinte.

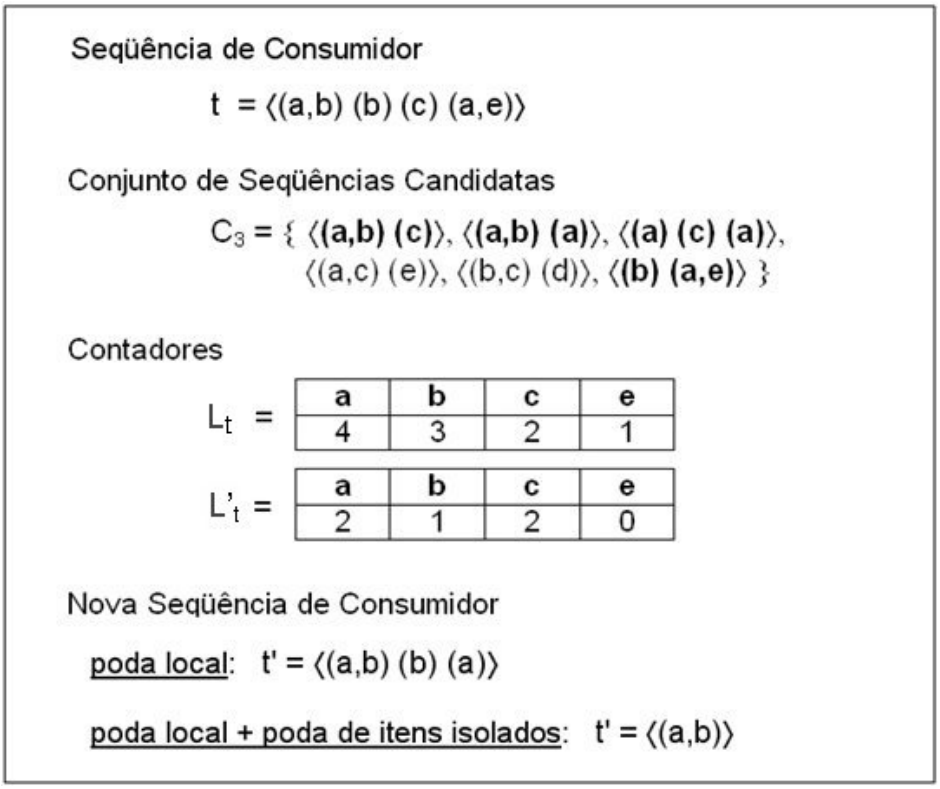

Figura 3.4: Exemplo de aplicação da poda local estendida com a poda de itens isolados.

Na poda global, a implantação da poda de itens isolados é feita de maneira análoga à realizada na poda local. São criados contadores adicionais usados somente pela poda de itens isolados, representados por G'_{k-1}. Os valores dos contadores são consultados sempre que algum item da seqüência de consumidor t aparece isolado. Os itens isolados em t podem ser removidos quando estes aparecem também isolados em menos do que $k-1$ seqüências freqüentes em F_{k-1}.

3.2 O Algoritmo *GSP2P*

No Capítulo 2, o algoritmo *GSP* foi apresentado e sua implementação, chamada *GSP2*, que adota uma estrutura alternativa à árvore hash na segunda iteração, foi discutida.

Nesta seção, será apresentado o algoritmo *GSP2P* (*GSP2 com Poda*), uma adaptação do *GSP2* que incorpora as técnicas de redução de base propostas nas seções anteriores. O *GSP2P* extrai os padrões seqüenciais iterativamente da mesma forma que o *GSP2*, porém ao final de cada iteração k, uma nova base de dados, D_{k+1}, é criada e utilizada na iteração seguinte $k+1$.

Para $k \leq 2$, o *GSP2P* possui um comportamento idêntico ao *GSP2*, pois as podas são realizadas a partir da terceira iteração, como explicado anteriormente. Só na iteração 3, o *GSP2P*

passa a se beneficiar com a redução das seqüências de consumidores, que são submetidas ao processo de poda global antes da contagem de suporte das seqüências candidatas. O processo de poda local também é realizado a partir da terceira iteração, mas como, nesta poda, a redução das seqüências de consumidor só ocorre ao final da iteração, o seu benefício só é percebido a partir da iteração 4.

O Algoritmo 3.1 apresenta os passos do algoritmo *GSP2P* nas iterações $k \geq 3$. As iterações 1 e 2 são executadas nos moldes do *GSP2*, conforme indicam as linhas 2 e 3. A base de dados utilizada na iteração 3 é a base original do problema, D, como indicado na linha 4 do algoritmo. A partir da quarta iteração, o algoritmo passa a ler a base de dados reduzida gerada na iteração anterior. Na linha 10, a função que realiza o processo de poda global é ativada. A partir dos contadores G_{k-1}, calculados durante a geração das seqüências candidatas (linha 8), esta função elimina os itens desnecessários da seqüência de consumidor lida t, gerando a nova seqüência t'. Esta seqüência t' será utilizada na contagem de suporte das seqüências candidatas, somente se o seu tamanho for maior ou igual a k (linha 11). Após a contagem de suporte das seqüências candidatas presentes em t' (linha 12), o processo de poda local é executado (linha 13). A poda local consulta os contadores L_t, criados durante o processo de contagem de suporte (linha 12), e elimina os itens desnecessários de t', criando uma nova seqüência de consumidor t''. A seqüência t'' é gravada na nova base de dados D_{k+1} (linha 15), se o seu tamanho for superior a k.

É importante ressaltar que, os processos de poda global e local do *GSP2P*, implementados pelas funções PodaGlobal e PodaLocal, incorporam o processo de poda de itens isolados.

Na seção a seguir, são apresentadas algumas considerações sobre a implementação do *GSP2P*.

3.2.1 Considerações de Implementação

O algoritmo *GSP2P* foi implementado utilizando-se a linguagem C. Seus parâmetros de execução são idênticos aos parâmetros utilizados na execução do algoritmo *GSP2*: base de dados, suporte mínimo, grau da árvore hash, saturação da folha, o número de diferentes itens existentes na base de dados e a memória disponível para sua execução. No *GSP2P*, são implementadas a poda local e a poda global, ambas utilizando também a poda de itens isolados.

Além do *GSP2P*, com o objetivo de avaliar o desempenho de cada tipo de poda isoladamente, foram implementadas algumas variações, listadas a seguir:

- *GSP2L*: *GSP2* com a utilização somente da poda local, sem a poda de itens isolados.

```
1  algoritmo GSP2P (D, suporte_minimo)
2      F_1 = seqüências freqüentes de tamanho 1 obtidas pelo GSP2;
3      F_2 = seqüências freqüentes de tamanho 2 obtidas pelo GSP2;
4      D_3 = D;
5      k = 2;
6      enquanto (F_k ≠ ∅) faça
7          k = k+1;
8          C_k = candidatas de tamanho k geradas a partir de F_{k-1};
9          para cada seqüência de consumidor t ∈ D_k faça
10             t' = PodaGlobal(t,G_{k-1});
11             se (|t'| ≥ k) então
12                 incrementar o contador das candidatas de C_k contidas em t';
13                 t'' = PodaLocal(t',L_t);
14                 se (|t''| > k) então
15                     D_{k+1} = D_{k+1} ∪ t'';
16                 fim
17             fim
18         fim
19         F_k = todas as candidatas de C_k com suporte ≥ suporte_minimo;
20     fim
21     retorna união de todos os conjuntos de freqüentes F_k;
22 fim
```

Algoritmo 3.1: Pseudo-código do algoritmo *GSP2P*.

- *GSP2G*: *GSP2* com a utilização somente da poda global, sem a poda de itens isolados.
- *GSP2LG*: *GSP2* com a utilização das podas local e global, sem a poda de itens isolados.

3.2.2 Contadores L_t, G_{k-1}, L'_t e G'_{k-1}

Para a implementação dos contadores L_t e L'_t são alocados dois vetores com n posições consecutivas na memória principal, sendo n o número de itens distintos na base de dados. Cada posição destes vetores contém um valor do tipo inteiro. A alocação de espaço é feita uma única vez durante toda a execução do algoritmo. Antes da contagem de suporte das seqüências candidatas em uma seqüência de consumidor t, os contadores dos itens pertencentes a t em L_t e L'_t são inicializados com o valor zero. Os contadores dos itens são indexados diretamente já que os itens são representados por valores entre 1 e n.

Para a implementação dos contadores G_{k-1} e G'_{k-1}, também são alocados dois vetores com n posições. A alocação de espaço é feita uma única vez durante toda a execução do algoritmo. Antes da geração das seqüências candidatas da k-ésima iteração, todos os contadores em G_{k-1} e G'_{k-1} são inicializados com o valor zero. Estes contadores também são indexados diretamente.

3.2.3 Gerência de Memória

O *GSP2P* herdou o gerenciamento de memória do *GSP2*, detalhado na Seção 2.2.4.1. Quando o conjunto de seqüências candidatas não cabe em memória, ele é particionado e a contagem do suporte é realizada separadamente sobre cada uma destas partes, sendo necessária a realização de várias leituras completas da base de dados.

Porém, cabe observar que, diversas leituras da base de dados numa mesma iteração inviabilizam a execução da poda local naquela iteração, já que para eliminar um item i de uma seqüência de consumidor t é necessário saber quantas vezes cada item i distinto de t contribuiu na contagem de suporte das seqüências candidatas. Como o conjunto de seqüências candidatas é particionado, fica inviável manter um contador para cada item distinto de cada seqüência de consumidor da base de dados.

A poda global pode ser executada, porém sofre alterações, quando ocorre o particionamento. A redução de cada seqüência de consumidor deve acontecer apenas na contagem de suporte da primeira partição de seqüências candidatas, ou seja, na primeira leitura completa da base de dados. A aplicação da poda global nas leituras seguintes é desnecessária, já que os itens desnecessários foram todos removidos durante o processamento da primeira partição de seqüências candidatas. Sendo assim, a nova base de dados, D_{k+1}, é gravada em disco após o processamento da primeira partição e já utilizada durante o processamento das partições seguintes.

Capítulo 4

Experimentos Computacionais

Neste capítulo, são apresentados os resultados computacionais obtidos utilizando-se as técnicas de poda de base de dados. Na Seção 4.1, é apresentada uma descrição detalhada das bases de dados utilizadas nos experimentos. Na Seção 4.2, os resultados das avaliações computacionais são apresentados. O objetivo principal dos experimentos realizados é comprovar a viabilidade e a eficácia das técnicas propostas. Parte do conteúdo deste capítulo foi apresentado em [4].

4.1 Bases de Dados

Os experimentos foram realizados sobre bases de dados artificiais criadas através de um gerador de base de dados sintética da IBM[1], descrito em [2]. Estas bases de dados artificiais simulam bases reais de transações de consumidores e foram utilizadas em muitos trabalhos, como em [1, 2, 3, 6, 9, 13, 17, 20]. Nesta dissertação, os valores dos parâmetros de geração destas bases são os mesmos utilizados em [20]. Estes parâmetros serão descritos a seguir.

As bases artificiais representam bases de dados transacionais, em que consumidores adquirem itens através de transações. As seqüências freqüentes extraídas destas bases são formadas por listas ordenadas de transações, sendo que cada transação é um conjunto não vazio de itens. Os parâmetros C, T, S, I e D, utilizados pelo gerador de base de dados, são descritos na Tabela 4.1. O valor do parâmetro C corresponde ao número médio de transações por consumidor, T corresponde ao número médio de itens por transação, S corresponde ao tamanho médio das seqüências maximais potencialmente freqüentes[2] (*maximal potentially large sequences*), I corresponde ao tamanho médio das transações nas seqüências maximais potencialmente fre-

[1] Disponível em: http://www.almaden.ibm.com/cs/quest/

[2] Pode-se definir seqüência maximal freqüente como sendo uma seqüência freqüente que não está contida em nenhuma seqüência também freqüente.

qüentes e, finalmente, D corresponde ao número total de consumidores.

Tabela 4.1: Descrição dos parâmetros utilizados na geração de bases artificiais.

Parâmetro	*Descrição*
C	número médio de transações por consumidor
T	número médio de itens por transação
S	tamanho médio das seqüências maximais potencialmente freqüentes
I	tamanho médio das transações nas seqüências maximais potencialmente freqüentes
D	número total de consumidores

Os valores dos parâmetros C, T, S, I e D devem ser combinados e a Tabela 4.2 relaciona as principais bases de dados sintéticas utilizadas e os valores dos seus parâmetros. Cada uma destas bases possui 10.000 itens distintos.

Tabela 4.2: Bases de dados sintéticas utilizadas.

Base de Dados	*C*	*T*	*S*	*I*	*D*
C20-T2.5-S4-I1.25-D200K	20	2,5	4	1,25	200.000
C10-T5-S4-I1.25-D200K	10	5	4	1,25	200.000
C10-T2.5-S4-I1.25-D2000K	10	2,5	4	1,25	2.000.000
C20-T2.5-S4-I1.25-D500K	20	2,5	4	1,25	500.000
C30-T2.5-S4-I1.25-D200K	30	2,5	4	1,25	200.000
C20-T2.5-S4-I1.25-D2000K	20	2,5	4	1,25	2.000.000

4.2 Avaliação de Desempenho do *GSP2P*

As avaliações realizadas nesta dissertação, a partir dos experimentos conduzidos, incluem análises de tempo, de redução da base de dados, de escalabilidade, entre outras, obtidas com a utilização dos algoritmos *GSP2*, *GSP2P* e suas variações. Todas as execuções foram realizadas com saturação da folha igual a 20 e grau da árvore hash igual a 10% do número de itens distintos existentes nas bases de dados. Estes valores foram os mesmos usados em [6]. A quantidade de memória disponibilizada para execução dos algoritmos foi de 100 megabytes (parâmetro de execução dos algoritmos). Os experimentos foram realizados utilizando-se um computador com processador *Intel Centrino*, 1,6 Mhz, com 512 *megabytes* de memória principal e sistema operacional Mandrake Linux 2.6.3-7. Os algoritmos foram implementados na linguagem C, utilizando-se o compilador gcc 3.3.2. Os suportes mínimos escolhidos foram $0,25\%$, $0,5\%$ e $0,75\%$, também utilizados em [20].

A Figura 4.1 apresenta, para as seis bases de dados avaliadas, o tempo total de execução dos algoritmos *GSP2* e *GSP2P* (em segundos) obtido, para os suportes mínimos $0,25\%$, $0,5\%$ e

$0,75\%$. Os resultados computacionais mostram que as técnicas de redução progressiva da base de dados, implementadas no algoritmo *GSP2P*, levam a uma redução significativa do tempo total do algoritmo *GSP2*.

Para as bases de dados e valores de suporte mínimo considerados (com exceção apenas do teste realizado com a base de dados (c) C10-T2.5-S4-I1.25-D2000K e com suporte mínimo de $0,75\%$, onde a quantidade de iterações não passou de três), o tempo total de execução do algoritmo *GSP2P* foi, em média, 70% (variando de 63% a 78%) do tempo total de execução do algoritmo *GSP2*, alcançando uma redução média de 30%. A redução progressiva e efetiva da base foi responsável pelo bom desempenho do algoritmo *GSP2P* na medida em que reduziu significativamente o custo com as operações e E/S e o alto custo da fase de contagem das seqüências candidatas.

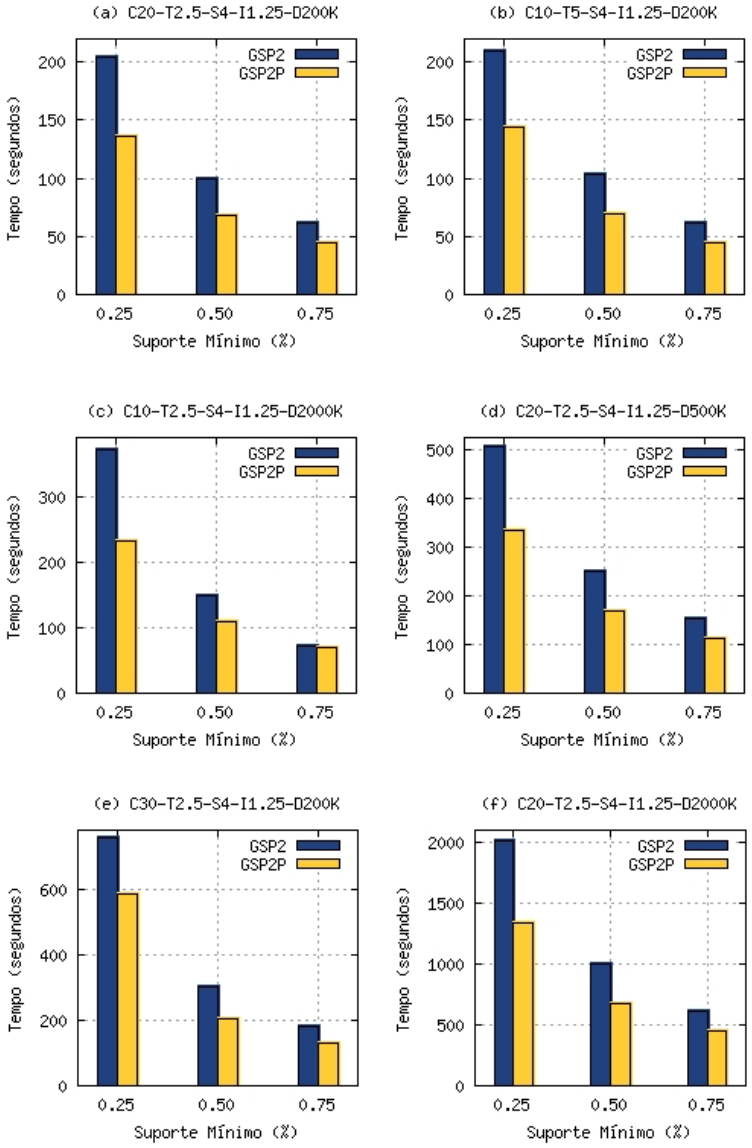

Figura 4.1: Tempo total de execução dos algoritmos GSP2 e GSP2P.

As Figuras 4.2 e 4.3 mostram, respectivamente, o tempo de execução (em segundos) e o tamanho da base de dados lida (em megabytes) em cada iteração do *GSP2* e do *GSP2P*. Nos dois experimentos, foram utilizadas as mesmas bases de dados e o suporte mínimo de $0,5\%$. Nota-se que a redução do tempo de processamento ocorre a partir da terceira iteração, devido ao fato de ser a primeira iteração a trabalhar com transações já podadas. Os benefícios obtidos pela leitura e utilização de uma base de dados menor só é sentida a partir da quarta iteração. Vale lembrar que as novas bases de dados são geradas ao final das iterações.

A Figura 4.2 indica, por exemplo, que, enquanto a quarta iteração do algoritmo *GSP2* foi executada em 24,23 segundos sobre a base de dados (d) C20-T2.5-S4-I1.25-D500K, a quarta iteração do algoritmo *GSP2P*, para a mesma base de dados, foi executada em 8,67 segundos, representando uma redução de 64%.

Nas primeira e segunda iterações, os algoritmos obtiveram aproximadamente o mesmo tempo de execução, já que o *GSP2P* realiza o mesmo processamento que o *GSP2* nestas iterações. Na terceira iteração, o *GSP2P* obteve um tempo ligeiramente menor de processamento, pois se beneficiou da aplicação da poda global antes da contagem de suporte das seqüências candidatas. Nas iterações $k > 4$, a redução do tempo de processamento foi significativa, porém a maior redução obtida aconteceu sempre na quarta iteração. A redução representativa do tempo na quarta iteração deve-se a significativa redução da base de dados observada no final da terceira iteração, quando o processo de poda local foi aplicado pela primeira vez.

Em relação à redução da base de dados, observa-se, na Figura 4.3, que, para a mesma base (d) C20-T2.5-S4-I1.25-D500K, a nova base de dados construída ao final da terceira iteração do algoritmo *GSP2P* é aproximadamente quatro vezes menor do que a base de dados original (passando de 190,51 MB para 44,60 MB), o que certamente contribuiu para a redução do tempo de execução da quarta iteração (passando de 24,23 segundos para 8,67 segundos). Nas iterações seguintes, com esta mesma base, observa-se que, a redução foi significativa porém menor do que na terceira iteração, mas sempre contribuindo para a redução do tempo de processamento das iterações seguintes e do tempo total de processamento.

Na média, a redução da base de dados na terceira iteração, considerando-se os testes com as seis bases de dados e os suportes mínimos de $0,25\%$, $0,5\%$ e $0,75\%$, foi de aproximadamente 66% (variando de 3% a 97%) e a redução do tempo de processamento da iteração 4 foi de 61% (variando de 3% a 93%). As iterações seguintes apresentaram taxas de redução da base de dados menores, porém efetivas, contribuindo para a redução do tempo total de processamento.

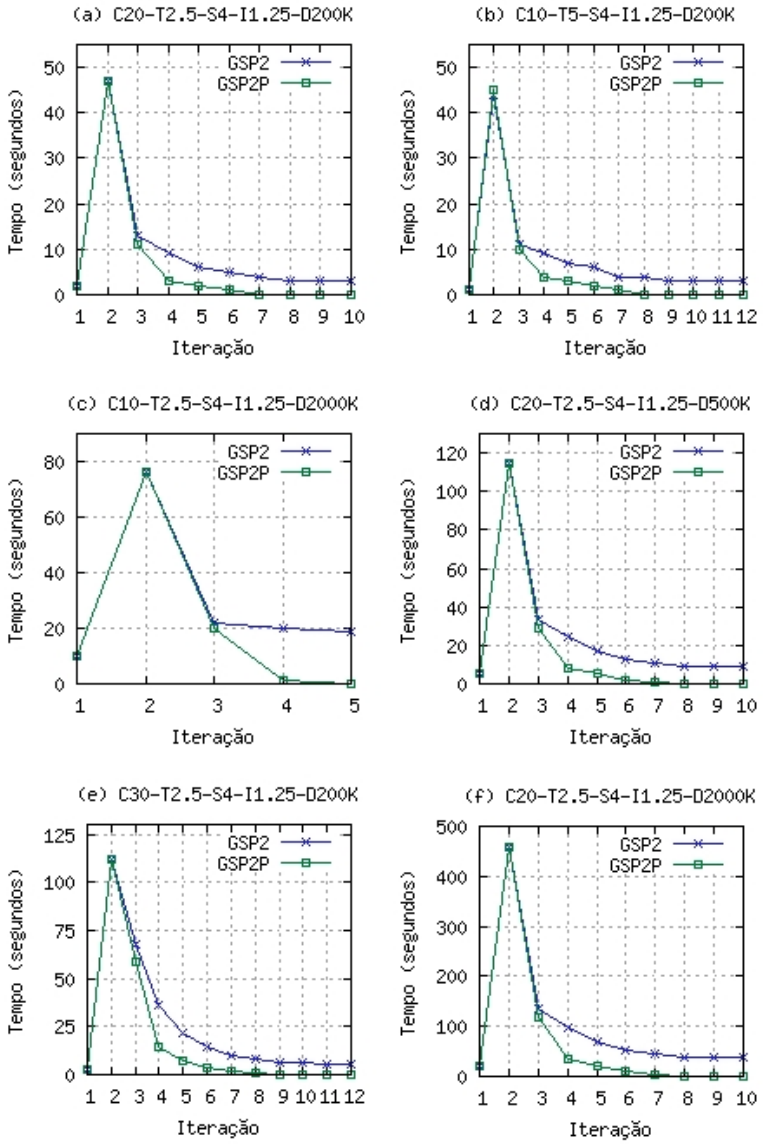

Figura 4.2: Tempo de execução de cada iteração dos algoritmos GSP2 e GSP2P, com suporte mínimo de $0,5\%$.

4.2.1 Variação do Número Médio de Transações por Consumidor (C)

Nesta subseção, é realizada a análise de desempenho dos algoritmos *GSP2* e *GSP2P* a partir da variação do número médio de transações por consumidor. São utilizadas seis bases de dados artificiais, todas com os valores dos parâmetros T, S, I e D iguais a 2,5, 4, 1,25 e 200.000, respectivamente. Cada uma delas com um valor diferente para o número médio de transações por consumidor, parâmetro C, variando de 10 a 35. Na Figura 4.4, para cada suporte utilizado – $0,25\%$, $0,5\%$ e $0,75\%$ – é apresentado um gráfico contendo o tempo total de execução do *GSP2* e *GSP2P* para cada valor distinto do parâmetro C.

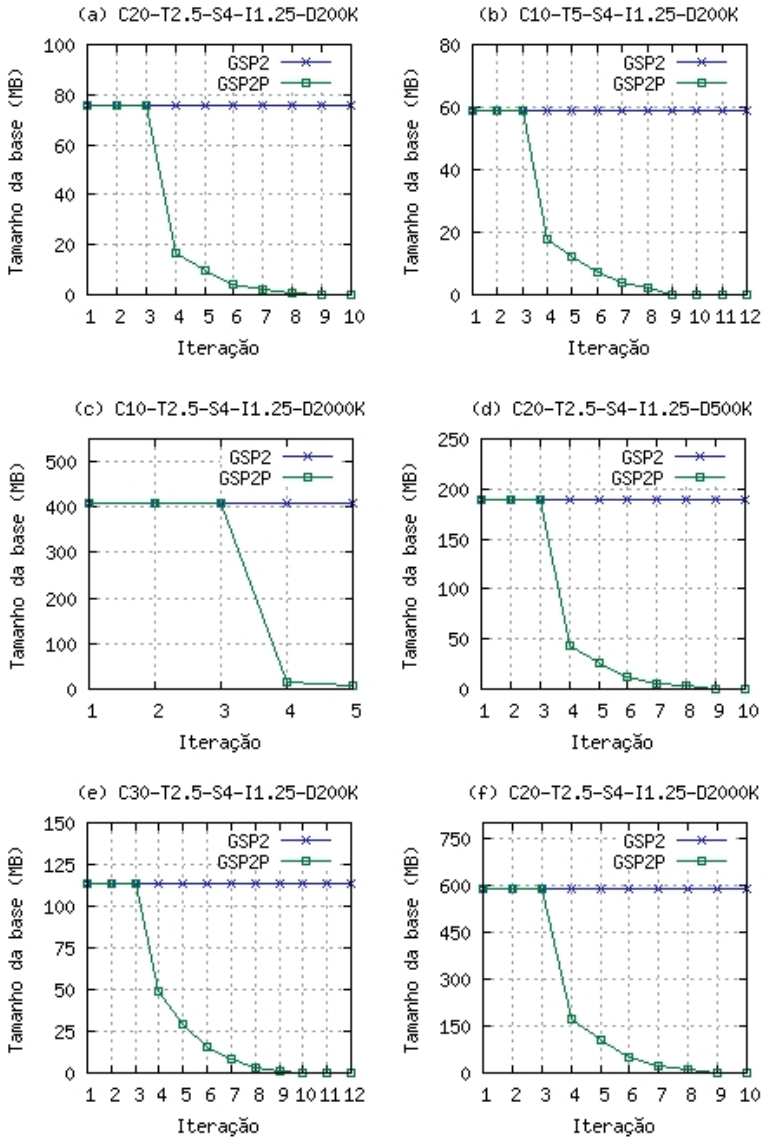

Figura 4.3: Tamanho da base de dados em cada iteração do algoritmo GSP2P, com suporte mínimo de $0,5\%$.

Observa-se um crescimento do tempo de execução das duas estratégias à medida que se aumenta o número médio de transações por consumidor. Isso porque, na etapa de contagem do suporte das seqüências candidatas, em qualquer iteração k, cada subseqüência s' de tamanho k, contida em cada seqüência de consumidor s, deve ser considerada, na tentativa de se encontrar alguma seqüência candidata igual a s', para ter seu suporte incrementado. Com o aumento no tamanho das seqüências de consumidores, maior será o número de subseqüência s' de tamanho k a serem consideradas, ocasionando um maior número de acesso às estruturas nas quais estão armazenadas as seqüências candidatas, aumentando o tempo de execução do algoritmo.

Observa-se que, também nesses experimentos, o tempo total de execução do *GSP2P* foi menor do que o tempo total do *GSP2*.

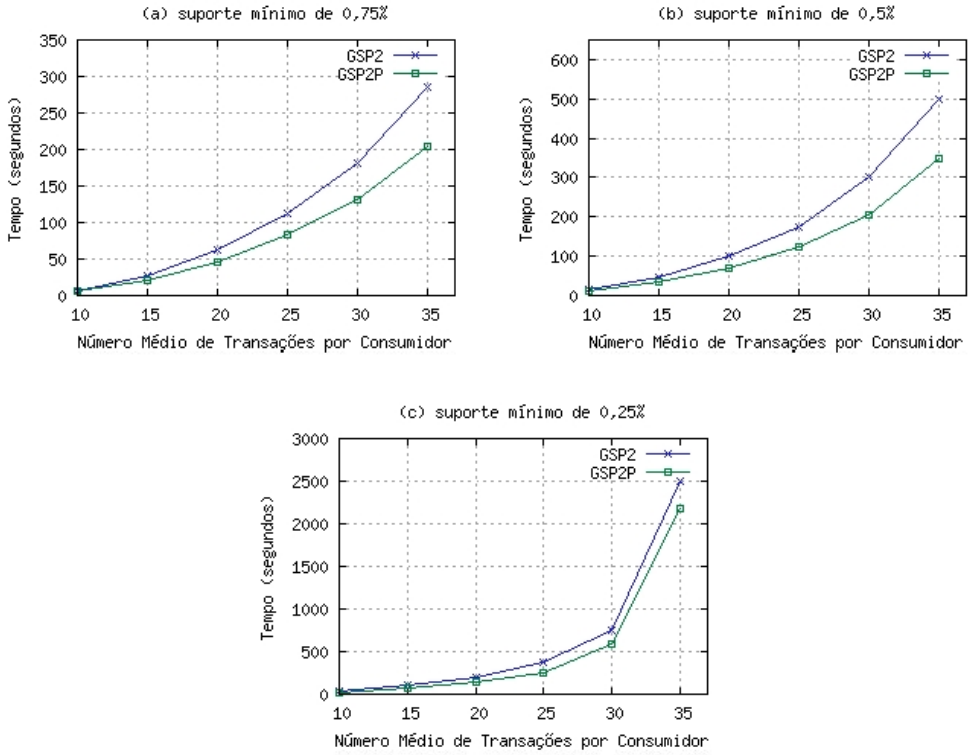

Figura 4.4: Tempo total de execução dos algoritmos GSP2 e GSP2P variando-se o número médio de transações por comsumidor.

Os tempos de execução (em segundos) dos mesmos experimentos são apresentados com mais detalhes na Tabela 4.3. Para cada base de dados utilizada, é apresentada uma linha com o tempo total de execução dos algoritmos *GSP2* e *GSP2P* e a porcentagem do tempo total de execução do *GSP2P* em relação ao do *GSP2*, para os suportes mínimos de $0,25\%$, $0,5\%$ e $0,75\%$.

O tempo total de execução obtido com o *GSP2P* foi em média 73% (variando de 64% a 97%) do tempo do *GSP2*, o que representa uma redução média de 27%.

Tabela 4.3: Variação do parâmetro C

	0,75			0,5			0,25		
Base de dados	GSP2	GSP2P	%	GSP2	GSP2P	%	GSP2	GSP2P	%
C**10**-T2.5-S4-I1.25-D200K	7,5	7,2	97	17,4	11,6	67	38,3	24,6	64
C**15**-T2.5-S4-I1.25-D200K	27,3	21,1	77	48,7	33,6	69	104,1	67,5	65
C**20**-T2.5-S4-I1.25-D200K	62,0	45,8	74	99,9	68,4	68	204,3	136,0	67
C**25**-T2.5-S4-I1.25-D200K	113,0	83,6	74	175,4	123,8	71	374,8	255,7	68
C**30**-T2.5-S4-I1.25-D200K	181,6	132,4	73	300,9	206,6	69	756,5	586,3	**78**
C**35**-T2.5-S4-I1.25-D200K	285,3	204,2	72	501,5	347,5	69	2.505,0	2.180,0	**87**
Média			78			69			71

É importante ressaltar que, nos experimentos realizados com o valor do parâmetro C igual a 30 e 35 e com o suporte mínimo de $0,25\%$, o *GSP2P* apresentou uma redução do tempo total

de execução em relação ao *GSP2* abaixo da média, com taxas de 22% e 13%, respectivamente. Isso porque, nestes experimentos, a quantidade de seqüências candidatas geradas na terceira iteração foi muito grande e o conjunto de seqüências candidatas de tamanho 3, C_3, não pôde ser armazenado por completo em memória principal. Logo, C_3 foi particionado e várias leituras completas da base de dados foram realizadas, inviabilizando a aplicação do processo de poda local nesta iteração e aumentando o tempo de processamento. Tal gerenciamento de memória foi explicado na Seção 3.2.3 em detalhes. Neste dois casos, o tempo de execução da terceira iteração foi responsável por grande parte do tempo total de processamento, o que explica a pequena redução observada no tempo total de execução do *GSP2P* em relação ao *GSP2*.

No experimento realizado com o valor do parâmetro C igual a 10 e com suporte mínimo de $0,75\%$, o *GSP2P* também apresentou uma redução do tempo total de execução em relação ao *GSP2* abaixo da média, com taxa de 3%. Isso porque, neste experimento, a quantidade de iterações não passou de três.

4.2.2 Variação do Número Médio de Itens por Transação (T)

Nesta subseção, é realizada a análise de desempenho dos algoritmos *GSP2* e *GSP2P* a partir da variação do número médio de itens por transação. São utilizadas quatro bases de dados artificiais, todas com os valores dos parâmetros C, S, I e D iguais a 10, 4, 1,25 e 200.000, respectivamente. Cada uma delas com um valor diferente para o número médio de itens por transação, parâmetro T, variando de 2,5 a 10. Na Figura 4.5, para cada suporte utilizado – $0,25\%$, $0,5\%$ e $0,75\%$ – é apresentado um gráfico contendo o tempo total de execução do *GSP2* e *GSP2P* para cada valor distinto do parâmetro T.

Observa-se um crescimento do tempo de execução das duas estratégias à medida que se aumenta o número médio de itens por transação. A justificativa é a mesma dada para o crescimento do tempo de execução observado na subseção anterior. Na etapa de contagem do suporte das seqüências candidatas, em qualquer iteração k, cada subseqüência s' de tamanho k, contida em cada seqüência de consumidor s, deve ser considerada, na tentativa de se encontrar alguma seqüência candidata igual a s', para ter seu suporte incrementado. Com um aumento no tamanho das seqüências de consumidores, maior será o número de subseqüência s' de tamanho k a serem consideradas, ocasionando um maior número de acesso às estruturas nas quais estão armazenadas as seqüências candidatas, aumentando o tempo de execução do algoritmo.

Em todos os experimentos, o *GSP2P* reduziu o tempo total de execução em relação ao *GSP2*.

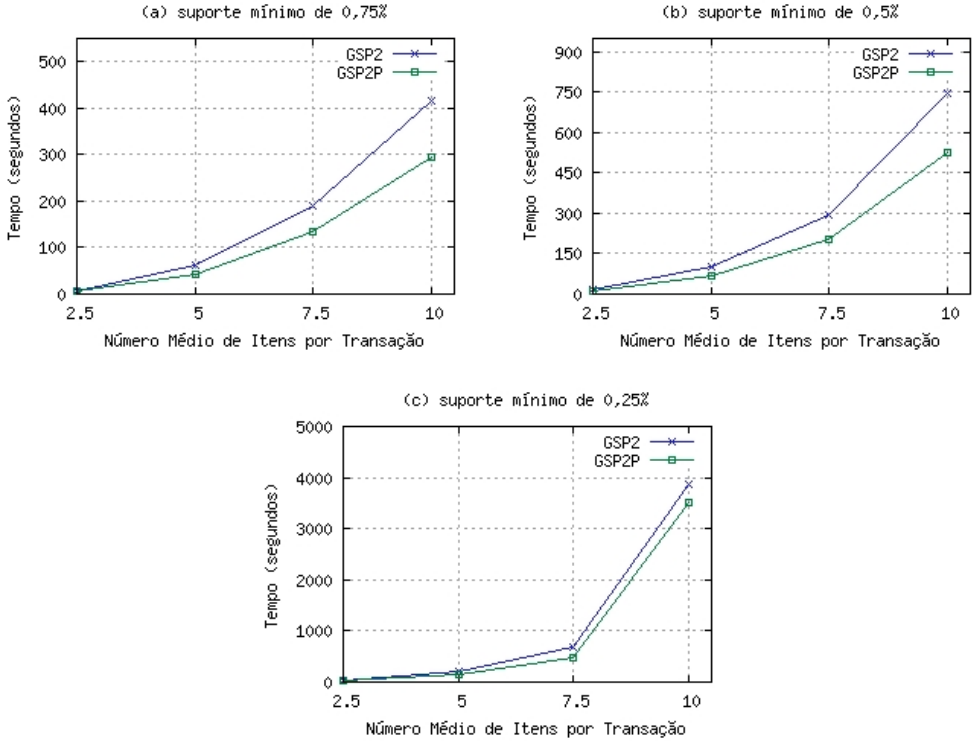

Figura 4.5: Tempo total de execução dos algoritmos GSP2 e GSP2P variando-se o número médio de itens por transação.

Os tempos de execução (em segundos) dos mesmos experimentos são apresentados com mais detalhes na Tabela 4.4. Para cada base de dados utilizada, é apresentada uma linha na tabela com o tempo total de execução dos algoritmos *GSP2* e *GSP2P* e a porcentagem do tempo total de execução do *GSP2P* em relação ao do *GSP2*, para os suportes mínimos de $0,25\%$, $0,5\%$ e $0,75\%$.

Observa-se que o tempo total de execução obtido com o *GSP2P* foi em média 73% (variando de 64% a 97%) do tempo do *GSP2*, o que representa uma redução média de 27%. Nota-se também que a redução do tempo total de processamento não sofre grande variação à medida que se altera o número médio de itens por transação, mantendo-se próximo à média de 27%.

É importante destacar que, no experimento realizado com a base de dados com T igual a 10 e suporte mínimo de $0,25\%$ a redução do tempo de processamento do *GSP2* não foi tão significativa, apenas 10%. Isso porque, na terceira iteração, a quantidade de seqüências candidatas geradas foi muito grande e o conjunto das seqüências candidatas de tamanho 3, C_3, foi particionado, sendo necessário realizar várias leituras completas da base de dados na terceira iteração, inviabilizando a execução do processo de poda local nesta iteração e aumentando o tempo de processamento. No experimento realizado com a base de dados com T igual a 2,5 e suporte mínimo de $0,75\%$, a redução do tempo de processamento do *GSP2* também não foi tão

significativa, apenas 3%. Isso porque o algoritmo não passou da terceira iteração, beneficiando-se apenas com a redução das seqüências de consumidores realizada pela poda global no início da terceira iteração.

Tabela 4.4: Variação do parâmetro T, com C igual a 10

	0,75			0,5			0,25		
Base de dados	GSP2	GSP2P	%	GSP2	GSP2P	%	GSP2	GSP2P	%
C10-T**2.5**-S4-I1.25-D200K	7,5	7,2	97	17,4	11,6	67	38,3	24,6	64
C10-T**5**-S4-I1.25-D200K	61,2	44,2	72	103,2	70,7	68	209,5	144,5	69
C10-T**7.5**-S4-I1.25-D200K	190,8	133,2	70	292,2	203,4	70	671,5	483,6	72
C10-T**10**-S4-I1.25-D200K	416,1	294,7	71	747,8	523,9	70	3.879,0	3.510,0	**90**
Média			77			69			74

Com o intuito de se obter tempos de execução maiores, são apresentados na Tabela 4.5 os mesmos experimentos realizados anteriormente, porém com o valor do parâmetro C (número médio de transações por consumidor) igual a 20. Os experimentos onde o tempo de processamento é representado por "+5h" excederam o tempo limite, estabelecido em 5 horas, e foram interrompidos.

Observa-se que, a redução do tempo total de execução se manteve significativa com a variação do número de itens por transação. O *GSP2P* reduziu em média 27% (variando de 9% a 33%) o tempo total de execução do *GSP2*.

Nos experimentos realizados com o valor do parâmetro T igual a 5 e com o suporte mínimo de $0,25\%$ e nos experimentos realizados com o valor do parâmetro T igual a 7,5 e com os suportes mínimos de $0,5\%$ e $0,75\%$, verifica-se uma menor redução no tempo de processamento do *GSP2P* (9%, 20% e 9%, respectivamente). Isso ocorreu pois a quantidade de seqüências candidatas na terceira iteração foi muito grande e, novamente, o conjunto das seqüências candidatas de tamanho 3, C_3, foi particionado, sendo necessário realizar várias leituras completas da base de dados na terceira iteração, inviabilizando a execução do processo de poda local nesta iteração e aumentando o tempo de processamento.

Tabela 4.5: Variação do parâmetro T, com C igual a 20

	0,75			0,5			0,25		
Base de dados	GSP2	GSP2P	%	GSP2	GSP2P	%	GSP2	GSP2P	%
C20-T**2.5**-S4-I1.25-D200K	62,0	45,8	74	99,9	68,4	68	204,3	136,0	67
C20-T**5**-S4-I1.25-D200K	439,3	310,4	71	810,0	573,8	71	5.378,0	4.917,0	**91**
C20-T**7.5**-S4-I1.25-D200K	3.692,0	2.952,0	**80**	15.884,0	14.460,0	**91**	+ 5 h	+ 5 h	
C20-T**10**-S4-I1.25-D200K	+ 5 h	+ 5 h		+ 5 h	+ 5 h		+ 5 h	+ 5 h	
Média			75			77			79

4.2.3 Variação do Número Total de Consumidores (D)

Nesta subseção, é realizada a análise de desempenho dos algoritmos *GSP2* e *GSP2P* a partir da variação do número total de consumidores. São utilizadas cinco bases de dados artificiais, todas com os valores dos parâmetros C, T, S e I iguais a 10, 2,5, 4 e 1,25, respectivamente. Cada uma delas com um valor diferente para o número total de consumidores, parâmetro D, variando de 100.000 a 2.000.000. Na Tabela 4.6, para cada base de dados utilizada, é apresentada uma linha na tabela com o tempo total de execução (em segundos) dos algoritmos *GSP2* e *GSP2P* e a porcentagem do tempo total de execução do *GSP2P* em relação ao do *GSP2*, para os suportes mínimos de $0,25\%$, $0,5\%$ e $0,75\%$.

Observa-se que, para os suportes de $0,25\%$ e $0,5\%$, a redução do tempo total de execução obtido com o *GSP2P* foi em média de 33% (variando de 26% a 37%) sobre o tempo do *GSP2*. Nos testes feitos com o suporte de $0,75\%$, a redução não foi tão significativa, observando-se uma redução média do tempo total de execução apenas de 3%. Isso porque os experimentos realizados com o suporte de $0,75\%$ não passaram da terceira iteração e se beneficiaram apenas da redução das seqüências de consumidores realizada pela poda global durante a terceira iteração.

Tabela 4.6: Variação do parâmetro D, com C igual a 10

	0,75			0,5			0,25		
Base de dados	GSP2	GSP2P	%	GSP2	GSP2P	%	GSP2	GSP2P	%
C10-T2.5-S4-I1.25-D**100K**	3,9	3,7	97	9,0	6,1	68	19,0	13,1	69
C10-T2.5-S4-I1.25-D**200K**	7,5	7,2	97	17,4	11,6	67	38,3	24,6	64
C10-T2.5-S4-I1.25-D**500K**	18,2	17,7	97	42,5	27,9	66	94,4	60,0	64
C10-T2.5-S4-I1.25-D**1000K**	36,2	35,2	97	75,1	55,2	74	187,6	118,2	63
C10-T2.5-S4-I1.25-D**2000K**	72,8	69,4	95	149,3	109,2	73	372,6	233,1	63
Média			97			69			65

Com o intuito de se obter tempos de execução maiores, são apresentados na Tabela 4.7 os mesmos experimentos realizados anteriormente, porém com valor do parâmetro C (número médio de transações por consumidor) igual a 20. Da mesma forma, para cada base de dados utilizada, é apresentada uma linha na tabela com o tempo total de execução dos algoritmos *GSP2* e *GSP2P* e a porcentagem do tempo total de execução do *GSP2P* em relação ao do *GSP2*, para os suportes mínimos de $0,25\%$, $0,5\%$ e $0,75\%$.

Observa-se que a redução do tempo total de execução se manteve significativa com a variação do número total de consumidores e houve uma pequena variação do percentual de redução entre os experimentos. O *GSP2P* reduziu em média 31% (variando de 22% a 34%) o tempo total de execução do *GSP2*.

Tabela 4.7: Variação do parâmetro D, com C igual a 20

	0,75			0,5			0,25		
Base de dados	GSP2	GSP2P	%	GSP2	GSP2P	%	GSP2	GSP2P	%
C20-T2.5-S4-I1.25-D**100K**	31,6	23,3	74	50,6	34,7	69	103,7	69,3	67
C20-T2.5-S4-I1.25-D**200K**	62,0	45,8	74	99,9	68,4	68	204,3	136,0	67
C20-T2.5-S4-I1.25-D**500K**	154,1	112,4	73	249,8	168,5	67	506,6	334,4	66
C20-T2.5-S4-I1.25-D**1000K**	288,4	224,1	78	497,6	334,5	67	1.005,9	664,4	66
C20-T2.5-S4-I1.25-D**2000K**	615,5	447,2	73	996,5	671,1	67	2.013,8	1.333,2	66
Média			74			68			66

4.2.4 Escalabilidade

Um algoritmo é dito escalável quando, dada uma quantidade fixa de memória principal, o comportamento do seu tempo de execução é linear em relação à variação do tamanho da base de dados.

Nesta subseção, realiza-se a análise da escalabilidade dos algoritmos *GSP2* e *GSP2P* a partir da variação do número total de consumidores. São utilizadas cinco bases de dados artificiais, todas com os valores dos parâmetros C, T, S e I iguais a 10, 2,5, 4 e 1,25, respectivamente. Cada uma delas com um valor diferente para o número total de consumidores, parâmetro D, variando de 100.000 a 2.000.000. Na Figura 4.6, é apresentado o tempo relativo de execução do algoritmo *GSP2P*, variando-se o número de consumidores para os suportes mínimos fixados em $0,25\%$, $0,5\%$ e $0,75\%$. Os tempos relativos de execução são calculados a partir do tempo de execução da base de dados que possui 100.000 consumidores.

Observa-se que, o tempo total de execução do algoritmo *GSP2P* possui um crescimento linear com o aumento da base de dados para diferentes suportes mínimos, demonstrando sua escalabilidade.

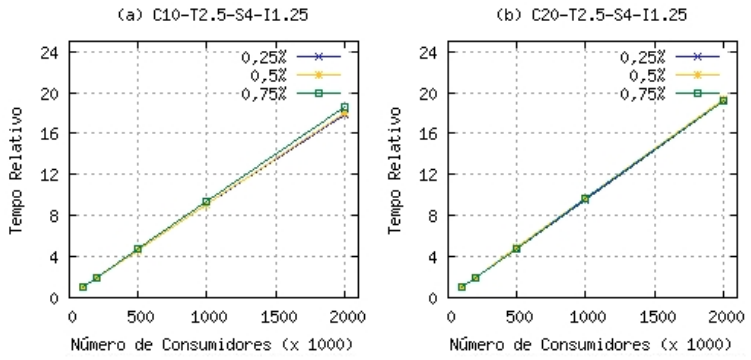

Figura 4.6: Tempo relativo de execução do algoritmo GSP2P variando-se o número de consumidores.

4.3 Avaliação de Desempenho das Técnicas de Redução de Base de Dados Separadamente

Nesta seção, são feitas avaliações de desempenho de cada uma das técnicas de redução da base de dados adotadas pelo algoritmo *GSP2P* separadamente, considerando-se tempo de execução e tamanho da base de dados a cada iteração.

Além do *GSP2* e do *GSP2P*, foram avaliados os seguintes algoritmos, variações do *GSP2P*:

- *GSP2L*: *GSP2* com a utilização somente da poda local, sem a poda de itens isolados.
- *GSP2G*: *GSP2* com a utilização somente da poda global, sem a poda de itens isolados.
- *GSP2LG*: *GSP2* com a utilização das podas local e global, sem a poda de itens isolados.

As Tabelas 4.8, 4.9 e 4.10 apresentam, para as seis bases de dados descritas anteriormente na Tabela 4.2, o tempo total de execução dos algoritmos *GSP2*, *GSP2G*, *GSP2L*, *GSP2LG* e *GSP2P* (em segundos) obtido, para os suportes mínimos de $0,75\%$, $0,5\%$ e $0,25\%$, respectivamente. Os resultados computacionais mostram que a aplicação da poda local juntamente com a poda global e a poda de itens isolados, implementadas no algoritmo *GSP2P*, levou a uma maior redução do tempo total de execução em relação ao tempo total de execução do algoritmo *GSP2*, em grande parte dos experimentos. O algoritmo *GSP2LG* obteve o segundo melhor desempenho, seguido, na ordem, pelo *GSP2L* e *GSP2G*. Este resultado evidencia a importância da utilização em conjunto dos três tipos de poda.

Para as bases de dados e valores de suporte mínimo considerados (com exceção apenas do teste realizado com a base de dados C10-T2.5-S4-I1.25-D2000K e com suporte mínimo de $0,75\%$, no qual a quantidade de iterações não passou de três), a redução do tempo total de execução do algoritmo *GSP2P* foi, em média, de $30,0\%$ em relação ao tempo total de execução do algoritmo *GSP2*, enquanto as reduções obtidas pelos algoritmos *GSP2LG*, *GSP2L* e *GSP2G* foram de $29,5\%$, $26,1\%$ e $21,6\%$, respectivamente.

Tabela 4.8: Tempo total de execução (em segundos) com o suporte mínimo de $0,75\%$

	0,75				
Base de dados	GSP2	GSP2G	GSP2L	GSP2LG	GSP2P
C20-T2.5-S4-I1.25-D200K	62,0	47,8	48,4	46,0	45,8
C10-T5-S4-I1.25-D200K	60,9	46,4	46,7	44,4	44,4
C10-T2.5-S4-I1.25-D2000K	72,8	70,9	81,2	70,6	69,4
C20-T2.5-S4-I1.25-D500K	154,1	118,0	119,3	113,3	112,4
C30-T2.5-S4-I1.25-D200K	181,6	146,1	140,8	133,4	132,4
C20-T2.5-S4-I1.25-D2000K	615,5	470,4	473,9	451,0	447,2

Tabela 4.9: Tempo total de execução (em segundos) com o suporte mínimo de $0,5\%$

	0,5				
Base de dados	GSP2	GSP2G	GSP2L	GSP2LG	GSP2P
C20-T2.5-S4-I1.25-D200K	99,9	75,7	71,9	68,4	68,4
C10-T5-S4-I1.25-D200K	103,2	77,7	73,6	70,4	70,7
C10-T2.5-S4-I1.25-D2000K	149,3	111,2	120,7	110,3	109,2
C20-T2.5-S4-I1.25-D500K	249,8	188,1	178,2	169,2	168,5
C30-T2.5-S4-I1.25-D200K	300,9	239,4	220,8	207,1	206,6
C20-T2.5-S4-I1.25-D2000K	996,5	747,1	707,9	674,4	671,1

Tabela 4.10: Tempo total de execução (em segundos) com o suporte mínimo de $0,25\%$

	0,25				
Base de dados	GSP2	GSP2G	GSP2L	GSP2LG	GSP2P
C20-T2.5-S4-I1.25-D200K	204,3	169,6	140,1	135,9	136,0
C10-T5-S4-I1.25-D200K	209,5	176,6	147,0	143,8	144,5
C10-T2.5-S4-I1.25-D2000K	372,6	264,4	247,2	235,9	233,1
C20-T2.5-S4-I1.25-D500K	506,5	418,6	345,1	333,8	334,3
C30-T2.5-S4-I1.25-D200K	756,5	676,5	624,5	596,2	586,3
C20-T2.5-S4-I1.25-D2000K	2.013,8	1.661,8	1.437,3	1.400,1	1.333,2

Para uma análise mais detalhada, os tempos de execução (em segundos) de cada iteração, utilizando-se as bases de dados C20-T2.5-S4-I1.25-D200K, C20-T2.5-S4-I1.25-D500K e C20-T2.5-S4-I1.25-D2000K, e suporte mínimo de $0,25\%$, são apresentados nas Tabelas 4.11, 4.12 e 4.13, respectivamente. Para cada iteração, é apresentada uma linha com o tempo de execução da iteração dos algoritmos *GSP2*, *GSP2G*, *GSP2L*, *GSP2LG* e *GSP2P*, e o número total de itens nas seqüências de consumidores lidas na iteração, para os algoritmos *GSP2G*, *GSP2L*, *GSP2LG* e *GSP2P*.

Observa-se nos três experimentos que, na primeira e segunda iterações, os algoritmos obtiveram aproximadamente o mesmo tempo de execução, já que o *GSP2G*, *GSP2L*, *GSP2LG* e *GSP2P* realizam o mesmo processamento que o *GSP2* nestas iterações.

Na terceira iteração, o *GSP2G* obteve os menores tempos de processamento, pois beneficiou-se da aplicação da poda global antes da contagem de suporte das seqüências candidatas e não realizou o processo de poda local, que é importante apenas para a iteração seguinte. O *GSP2L* obteve os maiores tempos de processamento, pois não se beneficiou da aplicação da poda global antes da contagem de suporte e realizou o processo de poda local. O *GSP2LG* e o *GSP2P* obtiveram tempos semelhantes.

Na quarta iteração, os maiores tempos de processamento foram obtidos pelos algoritmos *GSP2* e *GSP2G*. As reduções representativas dos tempos na quarta iteração nos algoritmos *GSP2L*, *GSP2LG* e *GSP2P* devem-se às significativas reduções das bases de dados observadas no final da terceira iteração, quando o processo de poda local foi aplicado pela primeira vez. A

redução no número de itens foi, em média, de 55% ao final da terceira iteração e a redução do tempo de execução na quarta iteração em relação ao *GSP2* foi, em média, de 49%. No *GSP2G*, no qual a poda local não é utilizada, a redução no número de itens ao final da terceira iteração foi, em média, de apenas 13% e a redução do tempo de execução na quarta iteração foi, em média, de 11%. Os tempos dos algoritmos *GSP2L*, *GSP2LG* e *GSP2P* nestas iterações foram aproximadamente os mesmos.

Nas iterações $k > 4$, as reduções dos tempos de execução foram significativas, porém menores do que as reduções obtidas na quarta iteração. As diferenças nos tempos dos algoritmos *GSP2L*, *GSP2LG* e *GSP2P* foram pequenas. Os maiores tempos de processamento foram obtidos pelos algoritmos *GSP2* e *GSP2G*.

Tabela 4.11: Tempo de execução (em segundos) de cada iteração com a base C20-T2.5-S4-I1-25-D200K

	GSP2	GSP2G		GSP2L		GSP2LG		GSP2P	
It.	tempo	tempo	itens	tempo	itens	tempo	itens	tempo	itens
1	2,1	2,2	9.745.353	2,2	9.745.353	2,2	9.745.353	2,1	9.745.353
2	58,0	58,4	9.745.353	58,3	9.745.353	58,4	9.745.353	58,4	9.745.353
3	40,1	38,3	9.745.353	43,3	9.745.353	38,6	9.745.353	38,8	9.745.353
4	32,5	28,6	8.482.350	16,3	4.159.318	16,4	4.159.318	16,4	4.159.316
5	23,0	19,1	7.454.839	9,5	2.588.072	9,6	2.588.072	9,6	2.588.058
6	14,1	11,3	6.124.998	5,3	1.533.305	5,4	1.533.305	5,4	1.533.268
7	8,3	6,5	4.604.321	2,8	835.928	2,9	835.928	2,9	835.832
8	5,9	3,5	2.978.196	1,5	369.621	1,5	369.621	1,5	369.524
9	4,7	1,3	1.208.658	0,7	137.010	0,7	137.010	0,7	136.971
10	4,1	0,4	251.054	0,3	59.233	0,3	59.233	0,3	59.233
11	3,9	0,1	63.351	0,1	23.552	0,1	23.552	0,1	23.552
12	3,9	0,0	21.025	0,0	18.451	0,0	18.451	0,0	18.433
13	3,8	0,0	13.681	0,0	9.906	0,0	9.906	0,0	9.906
Total	204,4	169,7		140,3		136,1		136,2	

Tabela 4.12: Tempo de execução (em segundos) de cada iteração com a base C20-T2.5-S4-I1-25-D500K

	GSP2	GSP2G		GSP2L		GSP2LG		GSP2P	
It.	tempo	tempo	itens	tempo	itens	tempo	itens	tempo	itens
1	5,3	5,5	24.215.214	5,5	24.215.214	5,4	24.215.214	5,3	24.215.214
2	140,3	142,3	24.215.214	141,5	24.215.214	142,0	24.215.214	140,4	24.215.214
3	99,4	94,3	24.215.214	107,8	24.215.214	95,3	24.215.214	97,0	24.215.214
4	80,8	70,8	21.240.592	40,3	10.390.889	40,4	10.390.889	40,7	10.390.889
5	57,4	47,6	18.585.932	23,5	6.420.991	23,8	6.420.991	23,9	6.420.955
6	35,4	28,2	15.300.142	13,1	3.834.772	13,3	3.834.772	13,4	3.834.685
7	21,3	16,3	11.406.716	7,1	2.073.935	7,3	2.073.935	7,3	2.073.719
8	14,8	8,8	7.501.997	3,7	988.845	3,8	988.845	3,8	988.548
9	11,9	3,5	3.332.063	1,7	352.040	1,7	352.040	1,7	351.988
10	10,4	0,9	620.889	0,7	169.445	0,7	169.445	0,7	169.441
11	9,9	0,2	196.308	0,2	59.327	0,2	59.327	0,2	59.327
12	10,0	0,1	52.788	0,1	46.959	0,1	46.959	0,1	46.913
13	9,6	0,0	34.596	0,0	25.198	0,0	25.198	0,0	25.198
Total	506,5	418,2		345,2		334,0		334,5	

Os resultados obtidos nesta seção comprovam a eficácia das técnicas propostas de redução

Tabela 4.13: Tempo de execução (em segundos) de cada iteração com a base C20-T2.5-S4-I1-25-D2000K

	GSP2	GSP2G		GSP2L		GSP2LG		GSP2P	
It.	tempo	tempo	itens	tempo	itens	tempo	itens	tempo	itens
1	21,4	22,0	81.659.568	21,9	81.659.568	22,1	81.659.568	21,7	81.659.568
2	558,8	561,9	81.659.568	561,2	81.659.568	561,6	81.659.568	557,9	81.659.568
3	393,5	376,7	81.659.568	458,2	81.659.568	413,0	81.659.568	388,1	81.659.568
4	323,2	282,9	71.628.422	177,2	41.493.208	180,4	41.493.208	164,2	41.493.208
5	227,1	188,4	62.676.265	104,6	25.792.020	106,0	25.792.020	95,1	25.791.892
6	138,4	112,1	51.595.785	58,0	15.346.540	59,2	15.346.540	52,8	15.346.128
7	84,3	64,9	38.466.210	30,8	8.356.527	31,5	8.356.527	28,6	8.355.680
8	59,8	34,5	25.298.551	15,2	3.844.041	15,8	3.844.041	14,8	3.842.824
9	47,9	13,6	11.236.524	6,7	1.406.380	6,9	1.406.380	6,6	1.406.184
10	41,9	3,6	2.093.788	2,6	623.256	2,5	623.256	2,5	623.235
11	40,1	0,9	661.998	0,8	235.054	0,9	235.054	0,8	235.054
12	38,9	0,3	178.014	0,3	183.644	0,3	183.644	0,3	183.458
13	38,7	0,1	116.666	0,1	98.929	0,1	98.929	0,1	98.929
Total	2.014,0	1.661,9		1.437,6		1.400,3		1.333,5	

da base de dados aplicadas ao problema de extração de padrões seqüenciais. A redução progressiva da base de dados reduziu significativamente o custo de várias leituras da base de dados e o tempo de contagem de suporte das seqüências candidatas. O algoritmo que obteve os melhores resultados foi o *GSP2P*, no qual as podas local e global, estendidas pela poda de itens isolados, foram aplicadas em conjunto. A poda local destacou-se como sendo a mais eficiente. A poda global foi responsável pela redução do tempo da terceira iteração. A poda de itens isolados, apesar de proporcionar uma redução pequena, mostrou-se importante.

Capítulo 5

Redução de Base de Dados na Extração de Padrões Seqüenciais Baseada em Restrições

Neste capítulo, as técnicas de redução da base de dados serão utilizadas no contexto da extração de padrões seqüenciais baseada em restrições. Na Seção 5.1, o problema é apresentado. Na Seção 5.2, é apresentado o algoritmo *GSP-F* (*GSP with Dataset Filtering*), proposto em [19]. O algoritmo que implementa as técnicas de redução da base de dados no novo contexto, chamado *GSP2P-F*, é apresentado na Seção 5.3. Os resultados das avaliações computacionais são apresentados na Seção 5.4.

5.1 Padrões Seqüenciais com Restrição

Na definição do problema de mineração de padrões seqüenciais, a única restrição que um padrão seqüencial deve satisfazer é ter um suporte mínimo. Entretanto, é comum encontrar usuários interessados em descobrir padrões seqüenciais que satisfaçam critérios mais sofisticados, como por exemplo: padrões com tamanho ou conteúdo específicos. Assim, pode-se definir a mineração de padrões seqüenciais baseada em restrições como sendo a extração de todas as seqüências freqüentes que satisfazem um conjunto de restrições estabelecido pelo usuário. O uso de restrições na extração de padrões seqüenciais restringe o espaço de busca, agilizando o processo de descoberta dos padrões.

As técnicas de extração de padrões seqüenciais baseada em restrições podem ser classificadas em três grupos, apresentados a seguir:

- Técnicas com filtragem pós-processamento: nestas estratégias, as restrições são aplicadas

após o processo normal de extração dos padrões seqüenciais, eliminando-se do conjunto freqüente os padrões seqüenciais que não satisfazem às restrições.

- Técnicas com filtragem de candidatas: nestas estratégias, as restrições são aplicadas durante o processo de geração das seqüências candidatas, eliminando-se as candidatas que não satisfazem às restrições. Assim, reduz-se o número de seqüências candidatas a serem processadas a cada iteração.

- Técnicas com filtragem de base de dados: nestas estratégias, durante o processo de contagem de suporte, apenas as seqüências de consumidores que possam contribuir na contagem de suporte de padrões que satisfaçam às restrições são consideradas. Assim, reduz-se o número de seqüências de consumidores a serem processadas.

Uma das estratégias mais representativas baseada na filtragem de candidatas é a família de algoritmos chamada *SPIRIT* [8]. Estes algoritmos são considerados extensões do *GSP* e as restrições são representadas como expressões regulares. O algoritmo *GSP-F* [19] é um exemplo de uso das técnicas de pós-processamento e de filtragem de base de dados.

Nestes algoritmos, as restrições podem ser definidas e representadas de várias maneiras. Segundo os autores do *GSP-F*, as categorias de restrições apresentadas a seguir cobrem uma grande parte das restrições importantes.

- restrição de item: estabelece qual item ou grupo de itens podem estar presentes nos padrões seqüenciais.

- restrição de tamanho: estabelece a quantidade de itens ou quantidade de elementos nos padrões seqüenciais.

- restrição de super-padrão: estabelece que os padrões seqüenciais devem ser subseqüências de uma seqüência específica definida pelo usuário.

- restrição de expressão regular: estabelece que os padrões seqüenciais devem satisfazer uma expressão regular específica definida pelo usuário.

- restrição de duração: estabelece que, nos padrões seqüenciais, a diferença de tempo entre os instantes em que a primeira e a última transações foram realizadas seja menor ou maior do que um dado período, ou que a diferença de tempo entre os instantes em que transações consecutivas foram realizadas seja menor ou maior do que um dado período.

5.2 O Algoritmo *GSP-F*

O algoritmo *GSP-F* (*GSP with Dataset Filtering*) [19] foi uma das primeiras estratégias a explorar a técnica de filtragem de base de dados na extração de padrões seqüenciais baseada em restrições. Este algoritmo extrai os padrões seqüenciais iterativamente, da mesma forma que o *GSP*, no qual, a cada iteração k, a base de dados é inteiramente lida e são encontradas as seqüências freqüentes de tamanho k. Embora a base de dados seja inteiramente lida, apenas as seqüências de consumidores que possam contribuir na contagem de algum padrão seqüencial que satisfaça às restrições do usuário são consideradas na fase de contagem de cada iteração.

Inicialmente, as restrições que são aplicadas às seqüências de consumidores, chamadas restrições de base, são geradas a partir das restrições definidas pelo usuário (a geração das restrições de base será detalhada na Seção 5.2.1). As restrições de base são aplicadas a cada seqüência de consumidor lida da base, e as seqüências de consumidores que não satisfazem tais restrições não são consideradas no processo de contagem de suporte das seqüências candidatas. Após a extração de todos os padrões seqüências sobre a base da dados filtrada, um passo adicional de filtragem é realizado, no qual os padrões que não satisfazem às restrições definidas pelo usuário são excluídos do conjunto freqüente. Este passo é necessário, pois a aplicação das restrições de base por si só não garante que somente padrões que suportam as restrições do usuário sejam extraídos.

Um pseudo-código do *GSP-F* é apresentado no Algoritmo 5.1. Inicialmente, na linha 2, o conjunto de restrições de base RB é gerado a partir do conjunto de restrições de usuário RU, parâmetro do algoritmo. Na primeira iteração do algoritmo, todas as seqüências candidatas de tamanho 1 (todos os itens da base) são contadas através de uma leitura completa da base de dados. Cada seqüência de consumidor lida da base é submetida ao conjunto de restrições de base RB e apenas aquelas que satisfazem todas as restrições em RB são consideradas no processo de contagem de suporte. Na linha 3, as seqüências candidatas de tamanho 1 que aparecem em um número mínimo ($suporte\_minimo$) de seqüências de consumidores formarão o conjunto de seqüências freqüentes F_1. As linhas de 5 a 14 apresentam os passos necessários para se encontrar todas as seqüências freqüentes de tamanho maior do que 1. Na segunda iteração, é feita a geração do conjunto de seqüências candidatas de tamanho 2 (C_2) (linha 7) a partir do conjunto de freqüentes F_1, da mesma maneira que o *GSP*. Em seguida, é feita uma nova leitura da base de dados, quando cada seqüência de consumidor lida da base é submetida ao conjunto de restrições RB e apenas aquelas que satisfazem todas as restrições em RB são comparadas com as seqüências candidatas de C_2 (linhas 8 a 12). Todas as seqüências candidatas que estiverem contidas em cada seqüência de consumidor têm seu suporte incrementado. Após a leitura da

base de dados e da contagem de suporte, as seqüências candidatas que possuem suporte maior ou igual ao suporte mínimo formam o conjunto F_2 (linha 13), que são as seqüências freqüentes de tamanho 2. O conjunto F_2 será usado para gerar o conjunto de seqüências candidatas de tamanho 3 (C_3) na próxima iteração do algoritmo. Este processamento se repete iterativamente até que, em uma iteração k, nenhuma seqüência candidata em C_k possuir um suporte maior ou igual ao suporte mínimo, ou seja, quando F_k for vazio. Na linha 15, os conjuntos de seqüências freqüentes obtidos nas diversas iterações são submetidos ao conjunto de restrições do usuário RU e apenas as seqüências freqüentes que satisfazem todas as restrições em RU irão compor o conjunto solução do problema.

```
1  algoritmo GSP-F (D, suporte_minimo, RU)
2      RB = transformação de RU em restrições de base;
3      F1 = seqüências freqüentes de tamanho 1 (itens contidos em no mínimo
       suporte_minimo seqüências de consumidores que satisfaçam RB);
4      k = 1;
5      enquanto (Fk ≠ ∅) faça
6          k = k+1;
7          Ck = candidatas de tamanho k geradas a partir de Fk-1;
8          para cada seqüência de consumidor t ∈ D faça
9              se (t satisfaz RB) então
10                 incrementar o contador das candidatas de Ck contidas em t;
11             fim
12         fim
13         Fk = todas as candidatas de Ck com suporte ≥ suporte_minimo;
14     fim
15     retorna união de todos os conjuntos de freqüentes Fk que satisfazem RU;
16 fim
```

Algoritmo 5.1: Pseudo-código do algoritmo *GSP-F*.

É importante ressaltar que, o número de seqüências freqüentes encontradas em cada iteração sobre a base de dados filtrada pode ser menor do que o número de seqüências freqüentes que seriam encontradas em cada iteração se fosse usada a base de dados original. Assim, conclui-se que o algoritmo *GSP-F* pode gerar menos seqüências candidatas, reduzindo o tempo de contagem de suporte. Tais seqüências candidatas geradas têm maior chance de satisfazer às restrições de usuário.

A Figura 5.1 apresenta um exemplo da extração de seqüências freqüentes utilizando os algoritmos *GSP* e *GSP-F*, na primeira e segunda iterações. São apresentados: o suporte mínimo de 2 consumidores, a base de dados contendo cinco seqüências de consumidores e duas restrições de base que são utilizadas pelo *GSP-F*. A primeira restrição, RB_1, filtra as seqüências de consumidores que possuem tamanho inferior a quatro. A segunda restrição, RB_2, filtra as

seqüências de consumidores que não contêm o item E. Na primeira iteração, o *GSP* encontra cinco itens freqüentes, A, B, C, D e E, enquanto o *GSP-F* encontra apenas quatro itens, A, B, C e E. Esta diferença ocorre porque o *GSP-F* considera, na contagem de suporte, apenas as seqüências de consumidores que satisfazem às restrições de base, ou seja, as seqüências t_1, t_2 e t_5. As seqüências t_3 e t_4 são filtradas pelas restrições RB_2 e RB_1, respectivamente. Sendo assim, na segunda iteração, o *GSP* gera 35 seqüências candidatas e encontra 9 seqüências freqüentes, enquanto o *GSP-F* gera 22 seqüências candidatas e encontra 8 seqüências freqüentes.

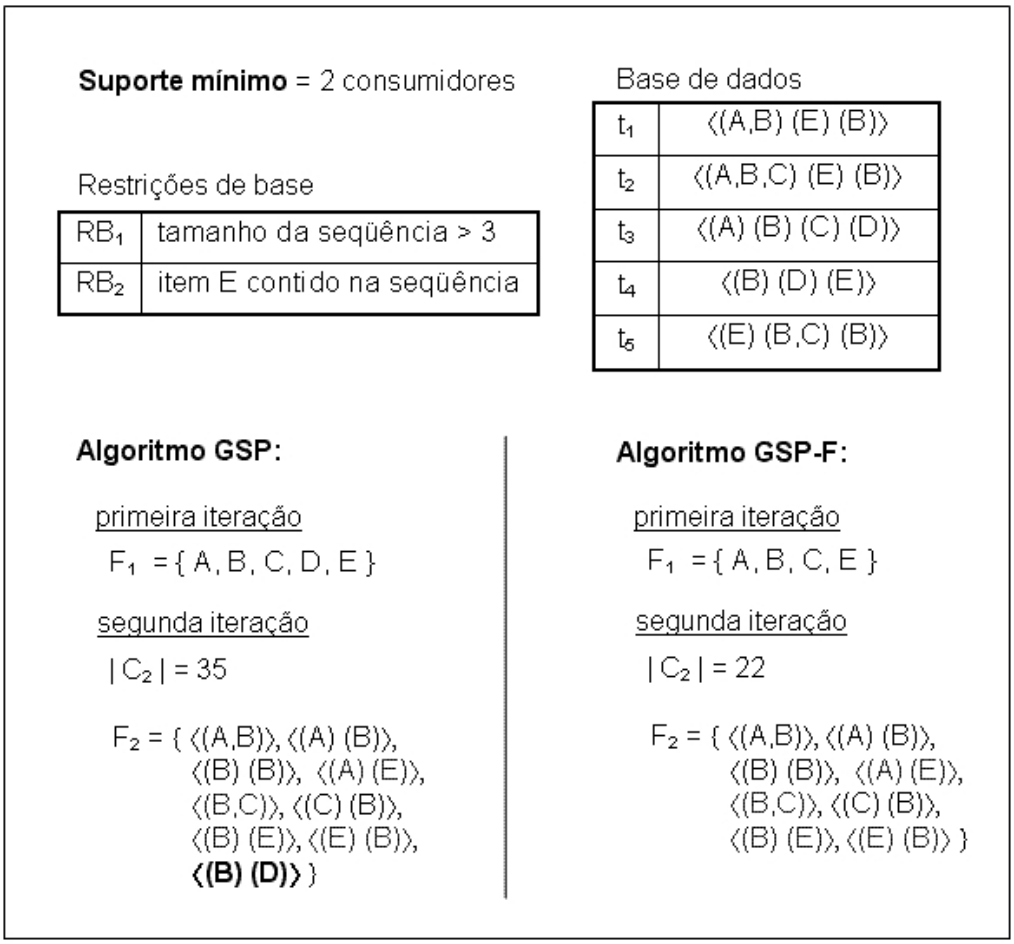

Figura 5.1: Exemplo de extração de seqüências freqüentes pelo GSP e GSP-F, na primeira e segunda iterações.

O algoritmo *GSP-F* não reduz a quantidade de seqüências de consumidores lidas da base de dados a cada iteração em relação ao *GSP*. Porém, se beneficia pela retirada das seqüências de consumidores que não satisfazem às restrições de base do processo de contagem de suporte.

5.2.1 Transformação de Restrições de Usuário em Restrições de Base

No *GSP-F*, o conjunto de restrições do usuário não é utilizado no processo de filtragem das seqüências de consumidores diretamente. Cada restrição do usuário é transformada em uma restrição de base antes de ser utilizada no processo de filtragem. Este por sua vez remove, do processo de contagem de suporte, as seqüências de consumidores que não satisfazem às restrições de base. As restrições de base são geradas de maneira que as seqüências de consumidores removidas são aquelas que não têm a possibilidade de contribuir na contagem de algum padrão seqüencial que satisfaça às restrições do usuário.

Os seguintes tipos de restrições de usuário são considerados pelo *GSP-F*. Segundo [19], tais tipos são adequados para permitir ao usuário expressar seus critérios de seleção.

- A restrição de usuário $RUa(\alpha, padrão)$ será satisfeita se o tamanho do padrão seqüencial $padrão$ for maior do que α.
- A restrição de usuário $RUb(\alpha, padrão)$ será satisfeita se a quantidade de elementos no padrão seqüencial $padrão$ for maior do que α.
- A restrição de usuário $RUc(\beta, padrão)$ será satisfeita se β for uma subseqüência do padrão seqüencial $padrão$.
- A restrição de usuário $RUd(\alpha, padrão, n)$ será satisfeita se o tamanho do n-ésimo elemento do padrão seqüencial $padrão$ for maior do que α.
- A restrição de usuário $RUe(\gamma, padrão, n)$ será satisfeita se γ for um subconjunto do n-ésimo elemento do padrão seqüencial $padrão$.

Os tipos de restrições de base utilizados pelo *GSP-F* são os seguintes.

- A restrição de base $RBa(\alpha, seqüência)$ será satisfeita se o tamanho da seqüência de consumidor $seqüência$ for maior do que α.
- A restrição de base $RBb(\alpha, seqüência)$ será satisfeita se a quantidade de transações na seqüência de consumidor $seqüência$ for maior ou igual a α.
- A restrição de base $RBc(\beta, seqüência)$ será satisfeita se a seqüência de consumidor $seqüência$ contiver a subseqüência β.

- A restrição de base $RBd(\alpha, seqüência)$ será satisfeita se existir pelo menos uma transação na seqüência de consumidor $seqüência$ com tamanho maior ou igual a α.

Na Tabela 5.1, define-se a transformação dos tipos de restrições de usuário nos tipos de restrições de base.

Tabela 5.1: Tipos de restrições de usuário e os tipos de restrições de base correspondentes.

Tipos de restrições de usuário	*Tipos de restrições de base*
$RUa(\alpha, padrão)$	$RBa(\alpha, seqüência)$
$RUb(\alpha, padrão)$	$RBb(\alpha + 1, seqüência)$
$RUc(\beta, padrão)$	$RBc(\beta, seqüência)$
$RUd(\alpha, padrão, n)$	$RBd(\alpha + 1, seqüência)$
$RUe(\gamma, padrão, n)$	$RBc(\langle\gamma\rangle, seqüência)$

A primeira transformação se baseia na propriedade de que um padrão seqüencial de tamanho maior do que α só pode estar contido em uma seqüência de consumidor de tamanho maior do que α.

A segunda transformação se baseia na propriedade de que um padrão seqüencial que possui a quantidade de elementos maior do que α só pode estar contido em uma seqüência de consumidor que possua uma quantidade de transações maior ou igual a $\alpha + 1$.

A terceira transformação se baseia na propriedade de que um padrão seqüencial que contém a seqüência β só pode estar contido em uma seqüência de consumidor que também contenha tal seqüência.

A quarta transformação se baseia na propriedade de que um padrão seqüencial que contém o n-ésimo elemento com tamanho maior do que α só pode estar contido em uma seqüência de consumidor que possua pelo menos uma transação de tamanho maior ou igual a $\alpha + 1$.

A quinta transformação se baseia na propriedade de que um padrão seqüencial que contém um dado conjunto de itens γ no n-ésimo elemento só pode estar contido em uma seqüência de consumidor que possua pelo menos uma transação contendo todos os itens desse conjunto. Escrevendo de outra forma, tal padrão seqüencial só pode estar contido em uma seqüência de consumidor que contenha a subseqüência $\langle\gamma\rangle$, sendo $\langle\gamma\rangle$ uma seqüência com apenas um elemento contendo todos os itens do conjunto γ.

Na subseção a seguir, são apresentadas algumas considerações sobre a implementação do *GSP2-F*.

5.2.2 *GSP2-F* - Uma Implementação do *GSP-F*

O código do algoritmo *GSP-F* utilizado neste trabalho, chamado aqui de *GSP2-F*, foi escrito na linguagem C. O *GSP2-F* é uma variação do *GSP2* (descrito na Seção 2.2.4) que incorpora o processamento das restrições.

Os parâmetros de execução do algoritmo *GSP2-F* são: base de dados, suporte mínimo, grau da árvore hash, saturação da folha, o número de diferentes itens existentes na base de dados, a memória disponível para sua execução e o conjunto de restrições do usuário.

5.3 O Algoritmo Proposto *GSP2P-F*

Nesta seção, será apresentado o algoritmo *GSP2P-F*, uma adaptação do *GSP2-F* que incorpora as técnicas de redução de base propostas no Capítulo 3. O *GSP2P-F* extrai os padrões seqüenciais iterativamente da mesma forma que o *GSP2-F*, porém ao final de cada iteração k, uma nova base de dados, D_{k+1}, é criada e utilizada na iteração seguinte $k+1$.

Para $k \leq 2$, o *GSP2P-F* possui um comportamento idêntico ao *GSP2-F*, pois as podas são realizadas a partir da terceira iteração, como explicado anteriormente. Só na iteração 3, o *GSP2P-F* passa a se beneficiar com a redução das seqüências de consumidores, que são submetidas ao processo de poda global antes da aplicação das restrições de base e da contagem de suporte das seqüências candidatas. O processo de poda local também é realizado a partir da terceira iteração, mas como, nesta poda, a redução das seqüências de consumidor só ocorre ao final da iteração, o seu benefício só é percebido a partir da iteração 4.

O Algoritmo 5.2 apresenta os passos do algoritmo *GSP2P-F* nas iterações $k \geq 3$. As iterações 1 e 2 são executadas nos moldes do *GSP2-F*, conforme indicam as linhas 3 e 4. A base de dados utilizada na iteração 3 é a base original do problema, D, como indicado na linha 5 do algoritmo. A partir da quarta iteração, o algoritmo passa a ler a base de dados reduzida gerada na iteração anterior. Na linha 11, a função que realiza o processo de poda global é ativada. A partir dos contadores G_{k-1}, calculados durante a geração das seqüências candidatas (linha 9), esta função elimina itens desnecessários da seqüência de consumidor lida t, gerando a nova seqüência t'. Esta seqüência t' será utilizada na contagem de suporte das seqüências candidatas somente se o seu tamanho for maior ou igual a k (linha 12) e se t' satisfizer todas as restrições em RB (linha 13). Após a contagem de suporte das seqüências candidatas presentes em t' (linha 14), o processo de poda local é executado (linha 15). A poda local consulta os contadores L_t, criados durante o processo de contagem de suporte (linha 14), e elimina os itens

desnecessários de t', criando uma nova seqüência de consumidor t''. A seqüência t'' é gravada na nova base de dados D_{k+1} (linha 17), se o seu tamanho for superior a k. Na linha 24, os conjuntos de seqüências freqüentes obtidos nas diversas iterações são submetidos ao conjunto de restrições do usuário RU e apenas as seqüências freqüentes que satisfazem todas as restrições em RU irão compor o conjunto solução do problema.

```
1  algoritmo GSP2P-F (D, suporte_minimo, RU)
2      RB = transformação de RU em restrições de base;
3      F1 = seqüências freqüentes de tamanho 1 obtidas pelo GSP2-F;
4      F2 = seqüências freqüentes de tamanho 2 obtidas pelo GSP2-F;
5      D3 = D;
6      k = 2;
7      enquanto (Fk ≠ ∅) faça
8          k = k+1;
9          Ck = candidatas de tamanho k geradas a partir de Fk−1;
10         para cada seqüência de consumidor t ∈ Dk faça
11             t' = PodaGlobal(t,Gk−1);
12             se (|t'| ≥ k) então
13                 se (t' satisfaz RB) então
14                     incrementar o contador das candidatas de Ck contidas em t';
15                     t'' = PodaLocal(t',Lt);
16                     se (|t''| > k) então
17                         Dk+1 = Dk+1 ∪ t'';
18                     fim
19                 fim
20             fim
21         fim
22         Fk = todas as candidatas de Ck com suporte ≥ suporte_minimo;
23     fim
24     retorna união de todos os conjuntos de freqüentes Fk que satisfazem RU;
25 fim
```

Algoritmo 5.2: Pseudo-código do algoritmo *GSP2P-F*.

É importante ressaltar que o número de seqüências freqüentes encontradas em cada iteração sobre a base de dados podada e filtrada deverá ser menor do que o número de seqüências freqüentes que seriam encontradas em cada iteração se fosse usada a base de dados somente filtrada. Dessa forma, o algoritmo *GSP2P-F* deverá gerar menos seqüências candidatas, reduzindo o tempo de contagem de suporte. Tais seqüências candidatas geradas têm maior chance de satisfazer às restrições de usuário.

A Figura 5.2 apresenta um exemplo da extração de seqüências freqüentes utilizando os algoritmos *GSP-F* e *GSP2P-F*, na terceira iteração. São apresentados: o suporte mínimo de 2 consumidores, a base de dados contendo cinco seqüências de consumidores, duas restrições

de usuário, as restrições de base correspondentes e o conjunto de seqüências freqüentes de tamanho 2, F_2. A primeira restrição de base, RB_1, é gerada a partir da restrição de usuário RU_1 e filtra as seqüências de consumidores que possuem tamanho inferior a quatro. A segunda restrição de base, RB_2, é gerada a partir da restrição de usuário RU_2 e filtra as seqüências de consumidores que não contêm o item E. Na terceira iteração, é gerada a mesma quantidade de seqüências candidatas, cinco, em ambos os algoritmos. Isso porque, na primeira e segunda iterações, o *GSP-F* e o *GSP2P-F* realizam o mesmo processamento. No *GSP-F*, durante o processo de contagem de suporte das seqüências candidatas de tamanho 3, apenas as seqüências de consumidores t_1, t_2 e t_5 são consideradas. As seqüências t_3 e t_4 são filtradas pelas restrições RB_2 e RB_1, respectivamente. Logo, são encontradas cinco seqüências freqüentes, ou seja, todas as candidatas são freqüentes. No *GSP2P-F*, antes do processo de contagem de suporte das seqüências candidatas de tamanho 3, é realizado o processo de poda global, durante o qual são removidos de cada seqüência de consumidor t os itens que aparecem em menos de duas seqüências freqüentes de tamanho 2, gerando uma nova seqüência de consumidor t'. Ou seja, são removidos: (a) os itens que possuem os respectivos contadores em G_2 com valores inferiores a 2, e (b) os itens que aparecem isolados e que possuem os respectivos contadores em G'_2 com valores inferiores a 2. Sendo assim, o *GSP2P-F*, durante o processo de contagem, considera apenas as seqüências de consumidores $t'_1 = \langle(A, B, C, E)\rangle$ e $t'_2 = \langle(A, B, C, D, E)\rangle$. As seqüências t'_3 e t'_4 não são consideradas, pois todos os seus itens são removidos pela poda global. A seqüência t_5, após ter o item E removido pelo poda global, $t'_5 = \langle(A, B, D)\rangle$, é filtrado pela restrição RB_1. Logo, são encontradas quatro seqüências freqüentes.

É importante lembrar que, após a extração de todos os padrões seqüenciais, uma filtragem adicional é realizada, na qual as seqüências freqüentes que não satisfazem às restrições definidas pelo usuário são excluídas do conjunto freqüente. Isso porque, a aplicação das restrições de base por si só não garante que somente padrões que suportam as restrições de usuário sejam descobertos. No exemplo apresentado na Figura 5.2, nenhuma das seqüências freqüentes apresentadas faz parte do conjunto solução do problema, pois de acordo com as restrições de usuário RU_1 e RU_2, apenas os padrões com tamanho maior do que 3 e que contêm o item E podem fazer parte do conjunto solução. Apesar disso, a geração das seqüências freqüentes de tamanho menor ou igual a 3 é importante, pois a partir delas são geradas as seqüências freqüentes de tamanho maior do que 3, que poderão conter o item E.

Após o processo de pós-processamento, o conjunto freqüente encontrado pelos algoritmos *GSP-F* e *GSP2P-F* é obrigatoriamente o mesmo.

Na subseção a seguir, são apresentadas algumas considerações sobre a implementação do

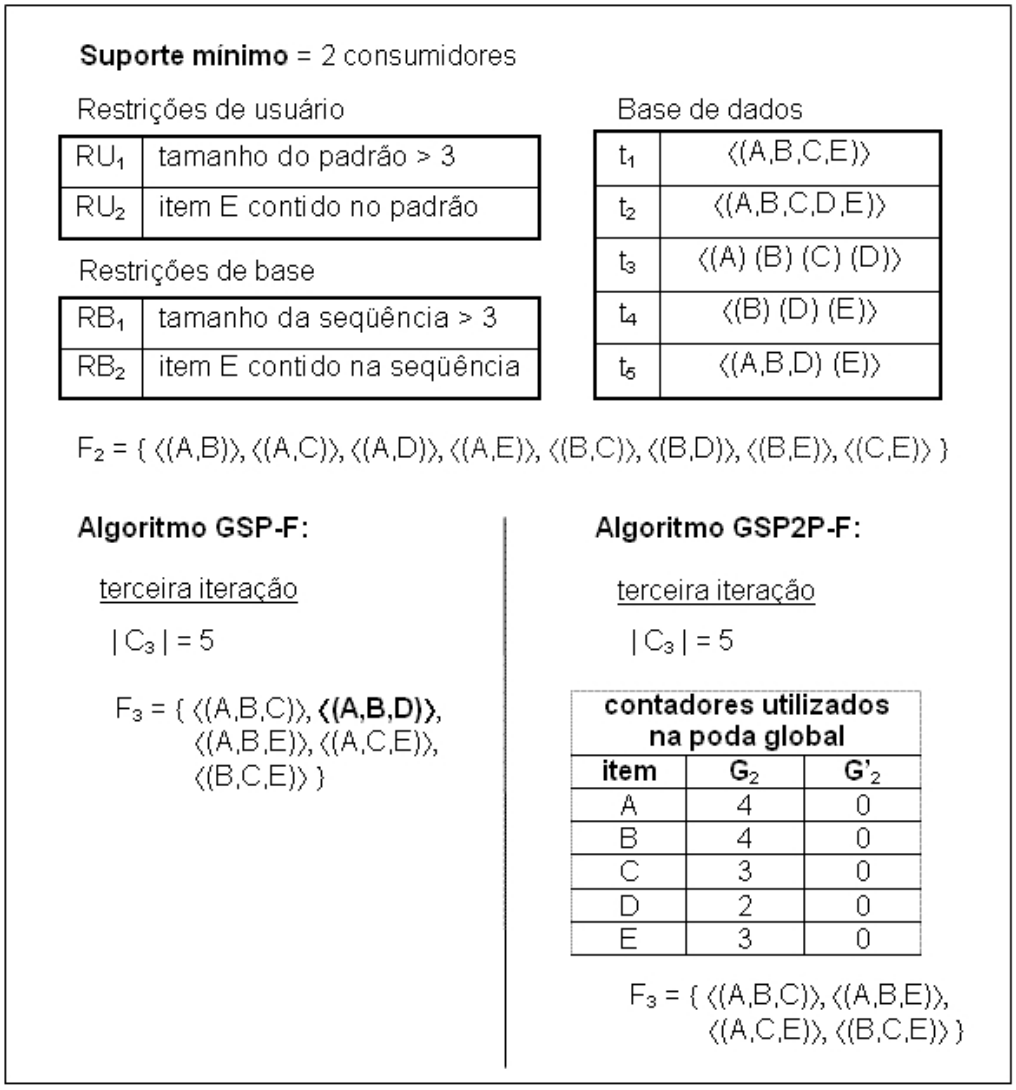

Figura 5.2: Exemplo de extração de seqüências freqüentes pelo GSP-F e GSP2P-F, na terceira iteração.

GSP2P-F.

5.3.1 Considerações de Implementação

O algoritmo *GSP2P-F* foi implementado utilizando-se a linguagem C. Seus parâmetros de execução são idênticos aos parâmetros utilizados na execução do algoritmo *GSP2-F*: base de dados, suporte mínimo, grau da árvore hash, saturação da folha, o número de diferentes itens existentes na base de dados, a memória disponível para sua execução e o conjunto de restrições do usuário. No *GSP2P-F*, são implementadas a poda local e a poda global, ambas utilizando também a poda de itens isolados.

5.4 Avaliação de Desempenho do *GSP2P-F*

As avaliações realizadas nesta seção, a partir dos experimentos conduzidos, incluem análises de tempo e de redução da base de dados obtidas com a utilização dos algoritmos *GSP2-F* e *GSP2P-F*. Todas as execuções foram realizadas com saturação da folha igual a 20 e grau da árvore hash igual a 10% do número de itens distintos existentes nas bases de dados. A quantidade de memória disponibilizada para execução dos algoritmos foi de 100 megabytes. Estes valores foram os mesmos usados no Capítulo 4. O suporte mínimo foi fixado em $0,25\%$ e as restrições utilizadas foram classificadas conforme seu grau de seletividade. O grau de seletividade é definido como a porcentagem de seqüências de consumidores da base de dados que satisfazem às restrições de base geradas a partir das restrições do usuário.

A Figura 5.3 apresenta, para as seis bases de dados descritas na Tabela 4.2, o tempo total de execução dos algoritmos *GSP2-F* e *GSP2P-F* (em segundos) obtido, utilizando restrições com os graus de seletividade 20%, 40%, 60% e 80%. Os tipos de restrições de usuário utilizados foram RUa e RUb, que estabelecem a quantidade de itens e quantidade de elementos nos padrões seqüenciais e, conseqüentemente, nas seqüências de consumidores. Para cada base de dados avaliada, os dois tipos de restrições e seus valores foram combinados para obtenção das seletividades desejadas, já que cada base de dados possui características diferentes quanto ao número médio de itens por consumidor e quanto ao número médio de transações por consumidor. Os resultados computacionais mostram que as técnicas de redução progressiva da base de dados, implementadas no algoritmo *GSP2P-F*, levam a uma redução significativa do tempo total do algoritmo *GSP2-F*.

Para as bases de dados e valores de seletividade considerados (excluindo apenas o teste realizado com a base de dados (c) C10-T2.5-S4-I1.25-D2000K e com seletividade de 20%, no qual a quantidade de iterações não passou de duas), o tempo total de execução do algoritmo *GSP2P-F* foi, em média, 51% (variando de 43% a 66%) do tempo total de execução do algoritmo *GSP2-F*, alcançando uma redução média de 49%. A redução progressiva e efetiva da base foi responsável pelo bom desempenho do algoritmo *GSP2P-F* na medida em que reduziu significativamente o custo com as operações de E/S e o alto custo da fase de contagem das seqüências candidatas.

Observa-se um crescimento do tempo de execução das duas estratégias à medida que se aumenta o grau de seletividade das restrições. Isso porque, com o aumento da seletividade, maior é a quantidade de seqüências de consumidores que serão analisadas na etapa de contagem do suporte das seqüências candidatas em cada iteração. Conseqüentemente, maior é a quantidade

de seqüências de consumidores analisadas, aumentando o tempo de execução do algoritmo.

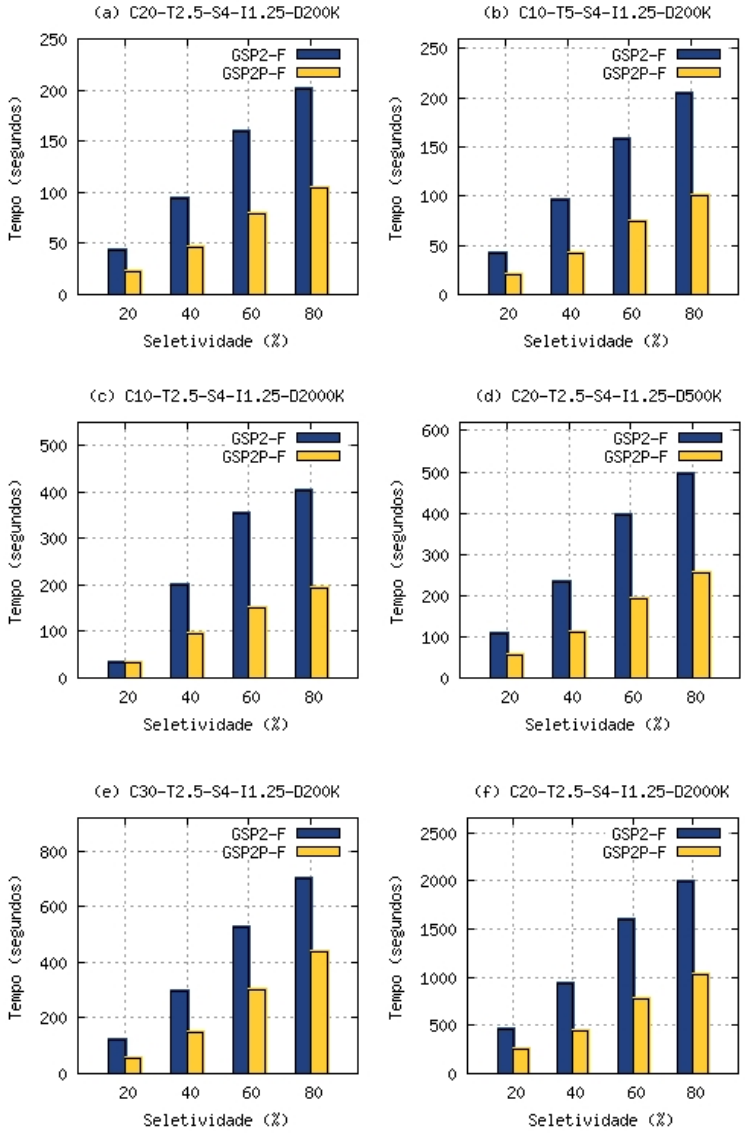

Figura 5.3: Tempo total de execução dos algoritmos GSP2-F e GSP2P-F, com suporte mínimo de $0,25\%$.

As Figuras 5.4 e 5.5 mostram, respectivamente, o tempo de execução (em segundos) e a quantidade de seqüências de consumidores analisadas no processo de contagem de suporte em cada iteração do *GSP2-F* e do *GSP2P-F*. Nos dois experimentos, foram utilizadas as mesmas bases de dados e o grau de seletividade de 60%. Nota-se que a redução do tempo de processamento ocorre a partir da terceira iteração devido ao fato de ser a primeira iteração a trabalhar com transações já podadas. Os benefícios obtidos pela leitura e utilização de uma base de dados menor só é sentida a partir da quarta iteração. Vale lembrar que as novas bases de dados são geradas ao final das iterações.

Observa-se também que a quantidade de iterações realizadas pelo *GSP2P-F* é menor do que

a quantidade realizada pelo *GSP2-F*. Isso porque o número de seqüências freqüentes encontradas pelo *GSP2P-F* em cada iteração foi menor, ocasionando a geração de menos seqüências candidatas nas iterações seguintes. Isso significa que os freqüentes encontrados em cada iteração pelo *GSP2P-F* têm maior chance de satisfazer às restrições de usuário que são aplicadas ao final do processamento. Com isso o *GSP2P-F*, em geral, termina o processamento antes, já que ambos os algoritmos param quando nenhuma nova seqüência freqüente é encontrada em uma iteração ou quando nenhuma seqüência candidata é gerada.

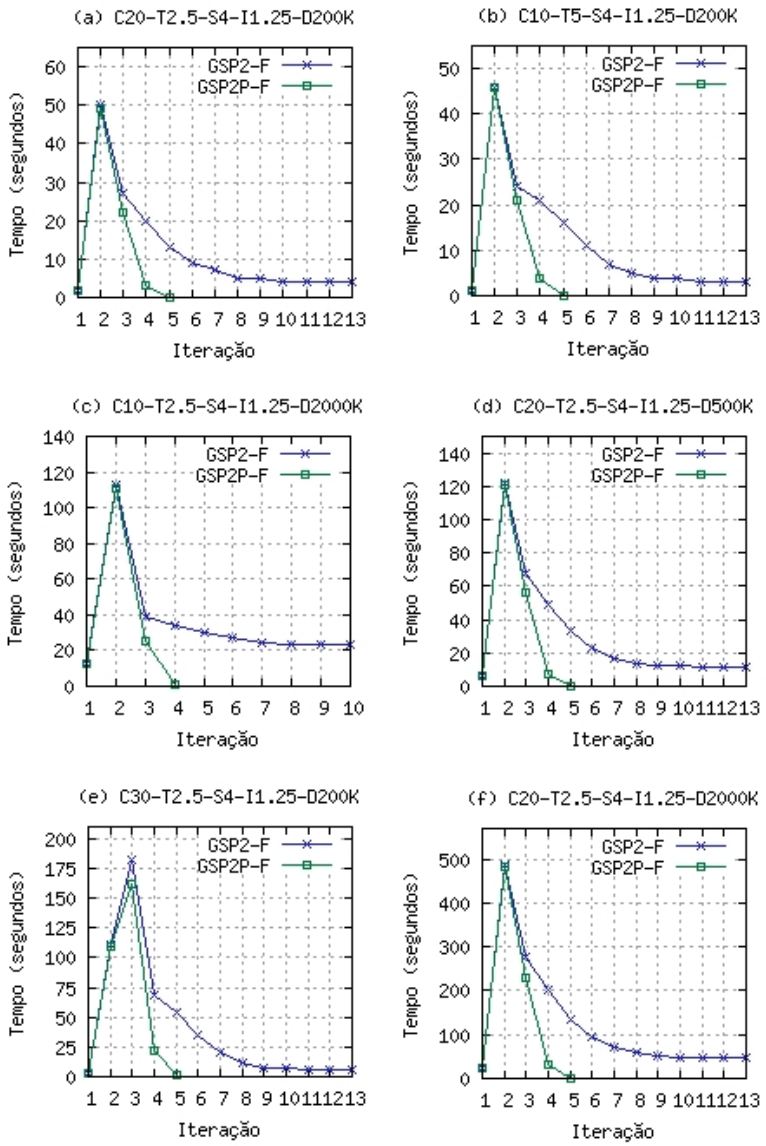

Figura 5.4: Tempo de execução de cada iteração dos algoritmos GSP2-F e GSP2P-F, com suporte mínimo de $0,25\%$ e seletividade de 60%.

A Figura 5.4 indica, por exemplo, que, enquanto a quarta iteração do algoritmo *GSP2-F* foi executada em 49,74 segundos sobre a base de dados (d) C20-T2.5-S4-I1.25-D500K, a quarta iteração do algoritmo *GSP2P-F*, para a mesma base de dados, foi executada em 7,72 segundos,

representando uma redução de 84%.

Nas primeira e segunda iterações, os algoritmos obtiveram aproximadamente o mesmo tempo de execução, já que o *GSP2P-F* realiza o mesmo processamento que o *GSP2-F* nestas iterações. Na terceira iteração, o *GSP2P-F* obteve um tempo ligeiramente menor de processamento, pois se beneficiou da aplicação da poda global antes da contagem de suporte das seqüências candidatas. Nas iterações $k > 4$, a redução do tempo de processamento foi significativa, porém a maior redução obtida aconteceu sempre na quarta iteração. A redução representativa do tempo na quarta iteração deve-se à significativa redução da base de dados observada no final da terceira iteração, quando o processo de poda local foi aplicado pela primeira vez.

Em relação à redução da base de dados, observa-se, na Figura 5.5, que, para a mesma base (d) C20-T2.5-S4-I1.25-D500K, a quantidade de seqüências de consumidores analisadas no processo de contagem de suporte na quarta iteração do algoritmo *GSP2P-F* é aproximadamente nove vezes menor do que a quantidade analisada pelo algoritmo *GSP2-F* na mesma iteração, o que certamente contribuiu para a redução do tempo de execução da iteração (passando de 49,74 segundos para 7,72 segundos). Nas iterações seguintes, observa-se que a redução da base foi significativa porém menor do que na quarta iteração, mas sempre contribuindo para a redução do tempo de processamento das iterações seguintes e do tempo total de processamento.

Os resultados obtidos nessa seção comprovam a viabilidade e a eficácia das técnicas de redução da base de dados aplicadas ao problema de extração de padrões seqüenciais baseada em restrições à medida que reduziu significativamente o custo de várias leituras da base de dados e o tempo de contagem de suporte das seqüências candidatas.

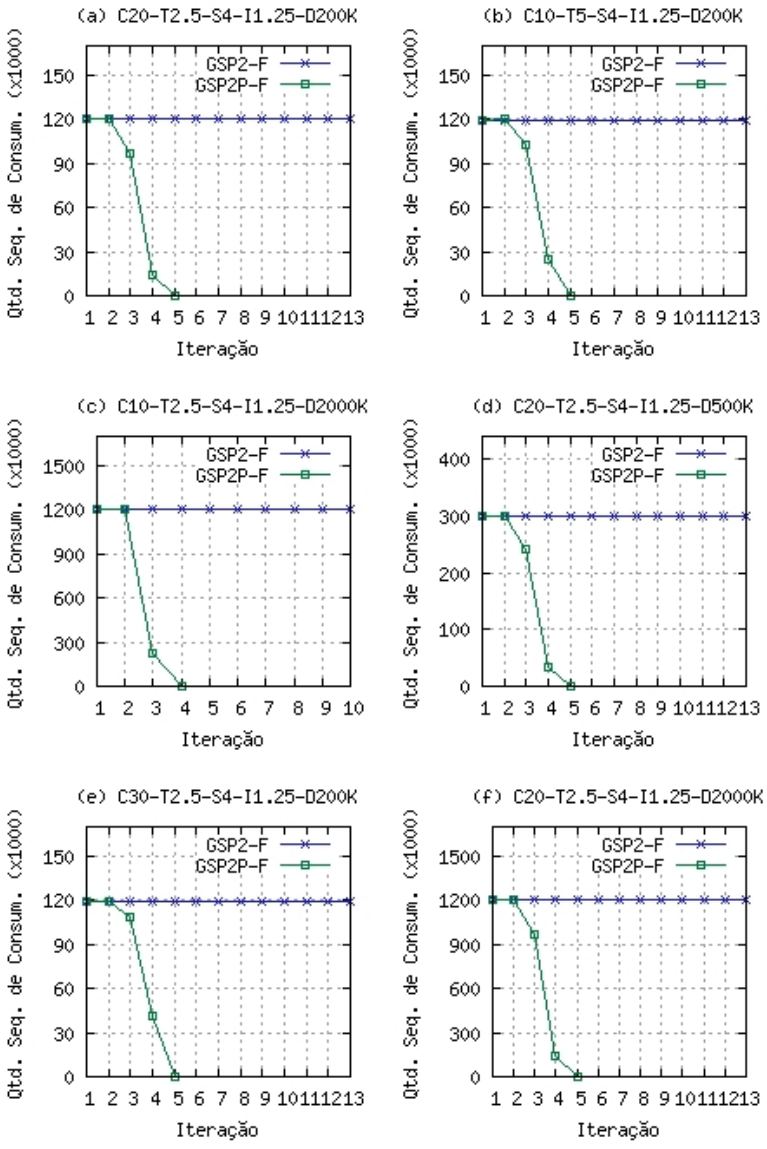

Figura 5.5: Número de seqüências de consumidores analisadas no processo de contagem de suporte a cada iteração dos algoritmos GSP2-F e GSP2P-F, com suporte mínimo de $0,25\%$ e seletividade de 60%.

Capítulo 6

Conclusões

Nesta dissertação, foi proposto o uso de técnicas de redução progressiva da base de dados em algoritmos iterativos para extração de padrões seqüenciais. Em estudos anteriores [6, 10], verificou-se que a etapa de contagem de suporte de seqüências candidatas possui um alto custo computacional. A partir desta análise, foram propostas técnicas de redução da base de dados baseadas nas estratégias de redução de base apresentadas pelos autores dos algoritmos *DHP* [12] e *kDCI++* [11] para o problema de mineração de conjuntos freqüentes. Neste contexto, a contribuição do presente trabalho foi a proposta de uma adaptação do algoritmo *GSP2* [6], denominada *GSP2P*, na qual as estratégias de redução da base de dados foram incorporadas. Os resultados avaliados a partir de diferentes combinações de bases de dados e suportes mínimos mostraram que as técnicas de redução de base implementadas no algoritmo *GSP2P* reduzem significativamente o tempo de execução total do algoritmo sem poda de base *GSP2*. Neste mesmo trabalho, com o objetivo de validar o uso das técnicas propostas e estender as suas aplicações, as técnicas foram aplicadas ao problema de extração de padrões seqüenciais baseada em restrições. Neste contexto, foi proposta uma adaptação do algoritmo *GSP-F* [19], denominada *GSP2P-F*, em que as estratégias de redução da base de dados foram incorporadas. Os resultados obtidos com os experimentos realizados sobre diferentes combinações de bases de dados e valores de seletividade das restrições mostraram que as técnicas de redução da base de dados adotadas pelo *GSP2P-F* reduzem significativamente o tempo de execução total do algoritmo sem poda de base *GSP2-F*.

Nos experimentos realizados sobre o *GSP2P*, verificou-se que o uso das técnicas de redução da base de dados reduziu o número de itens presentes nas seqüências de consumidores a cada iteração e, conseqüentemente, reduziu o número de seqüências de consumidores analisadas durante o processo de contagem de suporte. Para as bases de dados e valores de suporte mínimo considerados, o tempo total de execução do algoritmo *GSP2P* foi, em média, 70% do tempo

total de execução do algoritmo *GSP2*, alcançando uma redução média de 30%. A redução progressiva da base pôde ser observada tanto em termos da quantidade de memória lida a cada iteração quanto em termos do número total de itens.

Nas primeira e segunda iterações, os algoritmos obtiveram aproximadamente o mesmo tempo de execução, já que o *GSP2P* realiza o mesmo processamento que o *GSP2* nestas iterações. Na terceira iteração, o *GSP2P* obteve um tempo ligeiramente menor de processamento, pois se beneficiou apenas da aplicação da poda global antes da contagem de suporte das seqüências candidatas. Nas iterações $k > 4$, a redução do tempo de processamento foi significativa e a maior redução obtida aconteceu sempre na quarta iteração. A redução representativa do tempo nesta iteração deve-se à significativa redução da base de dados observada no final da terceira iteração, quando o processo de poda local foi aplicado pela primeira vez. Na média, a redução da base de dados na terceira iteração foi de aproximadamente 66% e a redução do tempo de processamento da iteração 4 foi de 61%.

Em outros experimentos realizados foram feitas avaliações de desempenho de cada uma das técnicas de redução da base de dados adotadas pelo algoritmo *GSP2P*, separadamente. Os resultados computacionais mostraram que a aplicação da poda local juntamente com a poda global e a poda de itens isolados, implementadas no algoritmo *GSP2P*, obteve a maior redução do tempo total de execução em relação ao tempo total de execução do algoritmo *GSP2*, em grande parte dos experimentos. O algoritmo que implementa as podas local e global, sem a poda de itens isolados, o *GSP2LG*, obteve o segundo melhor desempenho, seguido, na ordem, pelo algoritmo que implementa apenas a poda local, o *GSP2L*, e pelo algoritmo que implementa apenas a poda global, o *GSP2G*. A redução do tempo total de execução do algoritmo *GSP2P* foi, em média, de 30% em relação ao tempo total de execução do algoritmo *GSP2*, enquanto que as reduções obtidas pelos algoritmos *GSP2LG*, *GSP2L* e *GSP2G* foram de $29,5\%$, $26,1\%$ e $21,6\%$, respectivamente. Este resultado evidenciou a importância da utilização em conjunto dos três tipos de poda.

Nos experimentos realizados utilizando-se o *GSP2P-F* na extrações de padrões seqüenciais baseada em restrições, verificou-se que o uso das técnicas de redução da base de dados também reduziu o número de itens presentes nas seqüências de consumidores a cada iteração e, conseqüentemente, o tempo gasto na contagem de suporte. Para as bases de dados e valores de seletividade considerados, o tempo total de execução do algoritmo *GSP2P-F* foi, em média, 51% do tempo total de execução do algoritmo *GSP2-F*, alcançando uma redução média de 49%. A aplicação das técnicas de redução de base no problema de extração de padrões seqüenciais baseada em restrições evidenciou a eficiência das podas e comprovou que as mesmas podem

ser utilizadas em problemas correlatos ao problema de extração de padrões seqüenciais.

Como conseqüência natural deste trabalho, pretende-se: (a) explorar as técnicas de redução da base de dados em outras estratégias importantes de extração de padrões seqüenciais, como na versão *BFS* (*Breadth First Search*) do algoritmo *SPADE* [20], e (b) definir uma estratégia híbrida, análoga ao algoritmo *kDCI++* de extração de conjuntos freqüentes, na qual a base de dados é progressivamente reduzida até caber em memória principal, quando então é verticalizada e processada não mais em memória secundária.

O desenvolvimento e utilização das técnicas de redução progressiva da base de dados em outras estratégias importantes de extração de padrões seqüenciais baseada em restrições, como na família de algoritmos *SPIRIT* [8], também representa um provável trabalho futuro.

Referências

[1] AGRAWAL, R., AND SRIKANT, R. Fast algorithms for mining association rules. In *Proceedings of the the 20th VLDB International Conference on Very Large Databases* (1994), pp. 487–489.

[2] AGRAWAL, R., AND SRIKANT, R. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering* (1995), pp. 3–14.

[3] AYRES, J., GEHRKE, J., YIU, T., AND FLANNICK, J. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining* (2002), pp. 429–435.

[4] BARBOSA, C., PONTE, E., PRADO, A., AND PLASTINO, A. Explorando técnicas de redução de base de dados na mineração de padrões seqüenciais. In *I Workshop de Algoritmos de Mineração de Dados (WAMD2005)* (2005), vol. 1, pp. 41–48.

[5] COOLEY, R. W. Web usage mining: Discovery and application of interesting patterns from web data. *Tese de Doutorado, Department of Computer Science, University of Minnesota* (2000).

[6] DA PONTE, E. N. Mineração eficiente de padrões seqüenciais através da indexação de seqüências candidatas. *Dissertação de Mestrado, Universidade Federal Fluminense* (2003).

[7] F. MASSEGLIA, P. P., AND TEISSEIRE, M. Using data mining techniques on web access logs to dynamically improve hypertext structure. *ACM SigWeb Letters 3* (1999), 13–19.

[8] GAROFALAKIS, M. N., RASTOGI, R., AND SHIM, K. SPIRIT: Sequential pattern mining with regular expression constraints. In *The VLDB Journal* (1999), pp. 223–234.

[9] HAN, J., PEI, J., MORTAZAVI-ASL, B., CHEN, Q., DAYAL, U., AND HSU, M. C. FreeSpan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining* (2000), pp. 355–359.

[10] MASSEGLIA, F., CATHALA, F., AND PONCELET, P. The PSP approach for mining sequential patterns. In *Principles of Data Mining and Knowledge Discovery* (1998), pp. 176–184.

[11] ORLANDO, S., PALMERIMI, P., PEREGO, R., LUCCHESE, C., AND SILVESTRI, F. kDCI: a multi–strategy algorithm for discovering frequent sets in large databases. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* (2003).

[12] PARK, J. S., CHEN, M., AND YU, P. S. An effective hash-based algorithm for mining association rules. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1995), pp. 175–186.

[13] PEI, J., HAN, J. W., MORTAZAVI-ASL, B., PINTO, H., CHEN, Q., DAYAL, U., AND HSU, M. C. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the International Conference on Data Engineering* (2001), pp. 215–224.

[14] RAMIREZ, J. C. G., COOK, D. J., PETERSON, L. L., AND PETERSON, D. M. An event set approach to sequence discovery in medical data. *Intelligent Data Analysis 4* (2000), 513–530.

[15] RAMIREZ, J. C. G., COOK, D. J., PETERSON, L. L., AND PETERSON, D. M. Temporal pattern discovery in sparse course-of-disease data. *Medical Data Mining and Knowledge Discovery 4* (2001).

[16] SPILIOPOULOU, M., AND FAULSTICH, L. C. WUM: a web utilization miner. In *Workshop on the Web and Data Bases (WebDB98)* (1998), pp. 109–115.

[17] SRIKANT, R., AND AGRAWAL, R. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology* (1996), pp. 3–17.

[18] TARGA, C. N. Mineração eficiente de regras de associação através de indexação de conjuntos candidatos. *Dissertação de Mestrado, Universidade Federal Fluminense* (2002).

[19] WOJCIECHOWSKI, M., MORZY, T., AND ZAKRZEWICZ, M. Efficient constraint-based sequential pattern mining using dataset filtering techniques. In *Proceedings of the 5th IEEE International Baltic Workshop on Databases & Information Systems, Tallinn, Estonia* (2002), pp. 213–224.

[20] ZAKI, M. J. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal 42*, 1 (2001), 31–60.