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

Ciro Barbosa, Eduardo Ponte, Adriana Prado, Alexandre Plastino

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)

{cbarbosa, eponte, aprado, plastino}@ic.uff.br

Abstract. *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 algorithm* GSP2*, which does not use pruning mechanisms.*

Resumo. *Neste trabalho, 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*.*

1. Introdução

Padrões seqüenciais representam um importante tipo de informação extraído em processos de mineração de dados. A busca por padrões seqüenciais ocorre sobre uma base de dados transacional, onde cada transação possui o identificador do seu proprietário, chamado consumidor, a informação de quando ela foi realizada e os itens que a caracterizam. As transações podem ser agrupadas por consumidores, formando uma base de seqüências de consumidores, em que cada seqüência é formada pela lista ordenada (segundo a ocorrência) de transações de um determinado consumidor.

Conforme definido em [Agrawal and Srikant 1995], uma seqüência é uma lista ordenada de eventos, sendo cada evento um conjunto não vazio de itens. Um conjunto de itens é descrito por $(i_1, i_2, \ldots, i_m)$, onde cada i_j, $1 \leq j \leq m$, é um item do domínio da aplicação. Considera-se que os itens de cada evento estão ordenados lexicograficamente.

Uma seqüência s é descrita por $< s_1, s_2, \ldots, s_n >$, onde s_j, $1 \leq j \leq n$, é um conjunto de itens, também chamado de elemento da seqüência. O tamanho da seqüência é definido pelo seu número de itens. A seqüência $< (a, b, c)(d, b)(e, a) >$ tem tamanho 7.

Um mesmo item pode aparecer várias vezes na seqüência, mas uma única vez por elemento da seqüência. A seqüência anterior possui 7 itens, porém apenas 5 itens distintos.

Define-se que uma seqüência $s =< s_1, s_2, \ldots, s_n >$ é uma subseqüência de outra seqüência $q =< q_1, q_2, \ldots, q_m >$, $n \leq m$, se existirem inteiros $i_1 < i_2 < \ldots < i_n$, tais que $s_1 \subseteq q_{i_1}$, $s_2 \subseteq q_{i_2}$, ..., $s_n \subseteq q_{i_n}$. A seqüência $< (b)(a,c)(d,e) >$ é uma subseqüência de $< (a,b)(e)(a,c,e)(d,e) >$ já que $(b) \subseteq (a,b)$, $(a,c) \subseteq (a,c,e)$ e $(d,e) \subseteq (d,e)$.

Padrões seqüenciais são 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 suporte maior ou igual ao suporte mínimo.

Ao longo dos últimos dez anos, estratégias para extração de padrões seqüenciais vêm sendo desenvolvidas e aprimoradas: *AprioriAll* [Agrawal and Srikant 1995], *GSP* [Srikant and Agrawal 1996], *SPADE* [Zaki 2001], *FreeSpan* [Han et al. 2000], *PrefixSpan* [Pei et al. 2001], *SPAM* [Ayres et al. 2002]. Uma das primeiras estratégias propostas para solução deste problema, foi o algoritmo *GSP*. Esta estratégia tem como base o algoritmo *Apriori* [Agrawal and Srikant 1994], proposto para o problema da extração de conjuntos freqüentes pelos mesmos autores. Assim como o algoritmo *Apriori*, o algoritmo *GSP* extrai os padrões seqüenciais iterativamente. A cada iteração k, a base de dados é inteiramente lida em busca dos padrões seqüenciais de tamanho k.

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, garantindo, contudo, a extração correta de todos os padrões seqüenciais existentes na base de dados. 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.

O bom desempenho de algoritmos como *SPADE*, *FreeSpan*, *PrefixSpan* e *SPAM* 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.

A avaliação das técnicas propostas para a redução da base de dados será realizada a partir da comparação de duas estratégias. A primeira, denominada *GSP2*, é uma implementação da estratégia iterativa *GSP*, à qual foi incorporada a técnica de indexação de seqüências candidatas de tamanho $k = 2$, explorada em [da Ponte 2003], tornando o algoritmo *GSP* mais eficiente. A segunda, denominada *GSP2P* (*GSP2 with Pruning*), equivale à estratégia *GSP2* implementada com as técnicas propostas para poda da base.

Este trabalho está organizado da seguinte forma. Na Seção 2, é descrito o algoritmo *GSP2*. A Seção 3 é dedicada à descrição das técnicas de redução da base de dados implementadas no algoritmo *GSP2P*. Os resultados computacionais são avaliados na Seção 4. Finalmente, na Seção 5, são apresentadas as conclusões deste trabalho.

2. O algoritmo *GSP2*

O algoritmo *GSP2* é uma implementação da estratégia iterativa *GSP*, à qual foi incorporada a técnica de indexação de seqüências de tamanho $k = 2$ explorada em [da Ponte 2003], que torna o algoritmo *GSP* mais eficiente.

O *GSP2* é um algoritmo iterativo. Em uma primeira iteração, obtém-se o suporte dos conjuntos de itens de tamanho $k = 1$, através de um leitura da base de dados. Os itens que aparecem em um número mínimo de consumidores formarão 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, em uma determinada iteração, nenhuma nova seqüência candidata tem suporte maior ou igual ao suporte mínimo.

Durante a geração de seqüências candidatas, o algoritmo *GSP2* considera a propriedade de que toda seqüência que contém uma subseqüência não freqüente, também não é freqüente, reduzindo, assim, o número de seqüências candidatas a serem avaliadas. A técnica de geração de candidatas é dividida em duas fases: junção e poda.

Na geração de candidatas de tamanho $k = 2$, cada seqüência de F_1 é combinada com todas as outras seqüências do mesmo conjunto, inclusive com ela mesma, para gerar o conjunto C_2. Se $|F_1| = z$, então existem z^2 combinações possíveis das seqüências freqüentes de tamanho 1. Cada combinação pode gerar até duas diferentes seqüências candidatas. Considerando, por exemplo, as seqüências freqüentes de tamanho 1, $< (X) >$ e $< (Y) >$, a primeira candidata gerada possui a forma $< (X)(Y) >$, na qual seus dois itens pertencem a elementos diferentes e a segunda candidata possui a forma $< (X, Y) >$, na qual seus dois itens pertencem a um mesmo elemento. Considerando a ordem lexicográfica das seqüências de F_1, a candidata com a forma $< (X, Y) >$ somente pode ser gerada, se o item da segunda seqüência a ser combinada é maior lexicograficamente que o item da primeira $(X > Y)$. Desta forma, evita-se gerar duas vezes as candidatas equivalentes $< (X, Y) >$ e $< (Y, X) >$. Além disso, se a seqüência $< (X) >$ for combinada com ela mesma, gera-se somente a candidata $< (X)(X) >$.

Observe que o número de seqüências candidatas da forma $< (X)(Y) >$ é determinado por $|F_1| * |F_1|$ e o número de seqüências candidatas da forma $< (X, Y) >$ é determinado por $\binom{|F_1|}{2}$. Desta forma, para a contagem das candidatas de tamanho 2, o algoritmo *GSP2* não utiliza a árvore *hash*, como o algoritmo *GSP*, mas sim uma estrutura de dados composta por uma matriz M e um vetor V. A matriz M possui $|F_1| * |F_1|$ posições, sendo que cada posição $M[i][j]$, $1 \leq i \leq |F_1|$ e $1 \leq j \leq |F_1|$, armazena o suporte de uma seqüência candidata da forma $< (X)(Y) >$. Já o vetor V possui $\binom{|F_1|}{2}$ posições, onde cada posição $V[i]$, $1 \leq i \leq \binom{|F_1|}{2}$, armazena o suporte de uma candidata da forma $< (X, Y) >$. O acesso a esta estrutura é feito diretamente, o que reduz, significativamente, o custo computacional exigido pela segunda iteração do algoritmo *GSP*.

Na geração de candidatas de tamanho $k > 2$, combina-se pares de seqüências freqüentes de tamanho $k - 1$ obedecendo à seguinte regra: uma seqüência s pode unir-se

a uma seqüência q, somente se a subseqüência gerada através da retirada do primeiro item de s é a mesma subseqüência gerada através da retirada do último item de q. A seqüência candidata gerada é s acrescentada do último item de q. Por exemplo, as seqüências de tamanho 3 $s =< (a, b, c)(a) >$ e $q =< (b, c)(a, b) >$ podem unir-se para formar a seqüência candidata de tamanho $4 < (a, b, c)(a, b) >$. A partir da iteração $k > 2$, as seqüências candidatas têm seus suportes contados com auxílio da árvore *hash*, assim como no algoritmo *GSP*.

A fase de poda elimina as seqüências candidatas geradas na fase de junção que possuem alguma subseqüência não freqüente (considerando a propriedade citada anteriormente). Se na seqüência candidata $c =< (a, b, c)(a, b) >$, a subseqüência $< (a, b, c)(b) >$ não for uma seqüência freqüente de tamanho 4, a candidata c será descartada.

3. Técnicas de Redução da Base de Dados (Algoritmo *GSP2P*)

Nesta seção, serão apresentadas as técnicas de redução progressiva da base de dados para algoritmos iterativos de extração de padrões seqüenciais. As técnicas utilizadas são baseadas nas estratégias de redução de base de dados propostas pelos autores dos algoritmos *DHP* [Park et al. 1995] e *kDCI++* [Orlando et al. 2003] para o problema de mineração de conjuntos freqüentes e adaptadas aqui para o problema de extração de padrões seqüenciais. As técnicas de poda propostas serão avaliadas através da implementação do algoritmo *GSP2P* (*GSP2 with Pruning*).

Duas técnicas de redução da base de dados são implementadas: *poda global* da base, adaptação da poda global proposta pelos autores do algoritmo *kDCI++*, e a *poda local* da base, adaptação da poda local proposta pelo autores do algoritmo *DHP* e também utilizada pelo algoritmo *kDCI++*. Assim como nos algoritmos para extração de conjuntos freqüentes, no algoritmo *GSP2P*, as técnicas de poda global e local são aplicadas a cada transação t, respectivamente, antes e depois da contagem das seqüências candidatas presentes em t com o objetivo principal de reduzir o número de itens presentes em t e, conseqüentemente, reduzir o número de seqüências em t que deverão ser avaliadas.

A seguir, as técnicas são apresentadas separadamente. Sua descrição é baseada naquela apresentada em [Orlando et al. 2003], porém adaptada para o problema de extração de padrões seqüenciais.

3.1. Poda Global

A técnica de poda global é baseada nas seguintes propriedades: (a) Uma seqüência s de tamanho k presente em uma transação t da base de dados poderá ser freqüente somente se todos as suas subseqüências de tamanho $k - 1$ forem freqüentes, ou seja, pertencerem a F_{k-1}. (b) Em uma seqüência s de tamanho k, existem k subseqüências de tamanho $k - 1$ e cada item de s aparece em apenas $k - 1$ destas k subseqüências. (c) Itens que aparecem sozinhos nos elementos de uma seqüência s de tamanho k, além de aparecerem em apenas $k - 1$ subseqüências de tamanho $k - 1$ de s, conforme item (b), aparecem também sozinhos nestas subseqüências.

Considere, como exemplo, a seqüência $s =< (a, b)(c)(d) >$, de tamanho $k = 4$. Observe que s será uma seqüência freqüente somente se todas suas subseqüências de tamanho 3, $< (a, b)(c) >$, $< (a, b)(d) >$, $< (a)(c)(d) >$ e $< (b)(c)(d) >$ forem seqüências freqüentes. Note que há 4 (k) subseqüências de tamanho 3 em s e que cada

item desta seqüência aparece em apenas 3 ($k-1$) destas. Observe ainda que o item c (assim como o item d), que aparece sozinho em s, ocorre também sozinho em $k-1=3$ subseqüências de s.

Levando em consideração as propriedades descritas acima, a poda global é aplicada sobre cada transação da base de dados corrente da seguinte maneira. Antes da contagem das seqüências candidatas de tamanho k presentes em uma determinada transação t, para cada item i presente em t, verifica-se, primeiramente, se i está sozinho no elemento correspondente em t. Se sim, a técnica eliminará o item i de t se i aparecer também sozinho em menos de $k-1$ seqüências freqüentes de F_{k-1}. Em caso negativo, o item i será eliminado caso i apareça (sozinho ou não) em menos de $k-1$ seqüências freqüentes de F_{k-1}. Ao final do processo, se na nova transação t' existirem itens sozinhos (que em t não estavam sozinhos), a poda global é reaplicada para todos os itens nesta condição.

Após a execução da poda global, caso o tamanho de t' seja menor do que k, t' não é considerada durante a contagem das candidatas de tamanho k, já que não poderá contribuir para a formação de uma seqüência freqüente de tamanho k.

A poda global sobre as transações é aplicada a partir da segunda iteração, porém, por ter uma efeito pequeno nesta iteração, somente na terceira gera-se uma nova base.

3.2. Poda Local

A técnica de poda local é baseada nas mesmas propriedades a partir das quais a poda global foi definida. Entretanto, esta técnica é aplicada a cada transação t', após a contagem das seqüências candidatas de tamanho k em t', com o objetivo de eliminar os itens de t' que não poderão contribuir na formação de uma seqüência freqüente de tamanho $k+1$, durante a próxima iteração.

Como, durante o processamento das transações, ainda não se conhece o conjunto F_k e, como o conjunto de candidatas C_k é um superconjunto de F_k, a poda local é, então, baseada na seguinte heurística. Após a contagem das seqüências candidatas de tamanho k presentes em uma determinada transação t', a técnica de poda local é aplicada de forma a eliminar todos os itens i de t' que estão presentes em menos de k candidatas de C_k. Da mesma forma que na poda global, caso o tamanho da nova transação t'' seja menor do que $k+1$, t'' é descartada, já que não poderá contribuir para a formação de um conjunto freqüente de tamanho $k+1$.

A condição proposta para os itens que aparecem sozinhos nos elementos de uma transação também é testada durante a poda local. Após a aplicação da poda local, a transação resultante t'' é incluída na nova base a ser utilizada durante a próxima iteração.

Vale observar que, durante a segunda iteração ($k=2$), todos os itens presentes em uma transação t' (gerada após a aplicação da poda global) necessariamente estarão presentes em pelo menos $k=2$ candidatos de C_2. Desta forma, a poda local só poderá surtir efeito e só será executada a partir da terceira iteração.

4. Resultados Computacionais

Nesta seção, são apresentados os resultados computacionais obtidos pelas técnicas de redução da base de dados, implementadas no algoritmo *GSP2P*. Os experimentos foram realizados utilizando-se um computador com processador *Intel Centrino*, 1.6 GHz, 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.

Para os experimentos computacionais, utilizaram-se oito bases de dados sintéticas (utilizadas em [Zaki 2001]). Estas foram construídas utilizando-se o gerador de bases sintéticas da *IBM* [Agrawal and Srikant 1995] a partir dos parâmetros Cx-Ty-Sz-Iw-Du, onde x corresponde ao número médio de transações por consumidor, y corresponde ao número médio de itens por transação, z corresponde ao tamanho médio das seqüências maximais potencialmente freqüentes, w corresponde ao tamanho médio dos conjuntos de itens nas seqüências maximais potencialmente freqüentes e, finalmente, u corresponde ao número total de consumidores. Os suportes mínimos escolhidos foram $0,5\%$, média dos suportes avaliados em [Zaki 2001], e $0,25\%$, menor suporte avaliado em [Zaki 2001].

Para efeito de comparação, as Tabelas 1, 2 e 3 apresentam, para três das bases de dados avaliadas, o tempo de execução de cada iteração (em segundos) obtidos, respectivamente, pelos algoritmos *GSP2* e *GSP2P*, para os suportes mínimos 0,5% e 0,25%. As duas últimas colunas referentes a cada um dos suportes mínimos apresentam, respectivamente, o tamanho em *megabytes* da nova base de dados construída no final de cada iteração e o número de consumidores (parâmetro D) presentes nesta nova base de dados. Para a primeira e segunda iterações, estes valores se referem aos dados originais.

Os resultados computacionais mostraram que as técnicas de redução progressiva da base de dados, implementadas no algoritmo *GSP2P*, reduzem significativamente o tempo das iterações do algoritmo *GSP2*. Esta redução pode ser notada a partir da terceira iteração, devido ao fato de ser a primeira iteração a trabalhar com transações já podadas. A Tabela 2, por exemplo, indica que, enquanto a quarta iteração do algoritmo *GSP2* foi executada em 20,14 segundos sobre a base de dados *C10-T2.5-S4-I1.25-D1000K* e suporte mínimo 0,25%, a quarta iteração do algoritmo *GSP2P*, para a mesma base de dados e suporte mínimo, foi executada em 8,77 segundos, representando uma redução de 56%.

Em relação à redução da base de dados, observe que, neste mesmo experimento, a nova base de dados construída ao final da iteração 3 do algoritmo *GSP2P* é aproximadamente quatro vezes menor que a base de dados original, passando de 192,52 MB para 45,25 MB. Da mesma forma, o número de consumidores da nova base de dados foi reduzido de 1.000.000 para 493.522, o que certamente contribuiu para a redução do tempo de execução da iteração seguinte verificada no parágrafo anterior.

Nos experimentos realizados, o tempo total de execução consumido pelo algoritmo *GSP2P* foi significativamente menor que o tempo total de execução do algoritmo *GSP2*. A Tabela 4 apresenta, para cada base de dados avaliada, o tempo total de execução dos algoritmos *GSP2* e *GSP2P*, respectivamente. A última coluna referente a cada um dos suportes mínimos considerados, representa a relação entre o tempo total do algoritmo *GSP2P* e o tempo total de execução do algoritmo *GSP2*.

Para as bases de dados e valores de suporte mínimo considerados, o tempo total de execução do algoritmo *GSP2P* foi, em média, 67,8% (variando de 62% a 76%) do tempo total de execução do algoritmo *GSP2*, logo alcançando uma redução média de 32,2%. 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.

Tabela 1. C10-T2.5-S4-I1.25-D500K (96,66 MB)

	0,5				0,25			
Iteração	GSP2	GSP2P	Base (MB)	D	GSP2	GSP2P	Base (MB)	D
1	2,71	2,66	96,66	500.000	2,70	2,66	96,66	500.000
2	19,46	19,67	96,66	500.000	34,51	34,89	96,66	500.000
3	5,61	5,05	3,56	47.102	12,70	12,40	22,64	246.588
4	5,03	0,36	1,53	19.172	9,96	4,36	13,95	150.664
5	4,84	0,13	0,36	4.814	8,01	2,92	7,56	77.948
6	4,81	0,02	0,00	0	6,43	1,69	3,27	30.782
7					5,49	0,73	1,96	17.752
8					4,98	0,25	0,45	3.531
9					4,83	0,05	0,18	1.388
10					4,78	0,01	0,00	0
Total	42,46	27,89			94,39	59,96		

Tabela 2. C10-T2.5-S4-I1.25-D1000K (192,52 MB)

	0,5				0,25			
Iteração	GSP2	GSP2P	Base (MB)	D	GSP2	GSP2P	Base (MB)	D
1	5,50	5,31	192,52	1.000.000	5,47	5,31	192,52	1.000.000
2	38,65	38,80	192,52	1.000.000	67,55	68,46	192,52	1.000.000
3	11,16	10,11	7,21	97.675	25,55	24,80	45,25	493.522
4	10,10	0,71	3,06	38.260	20,14	8,77	28,19	302.886
5	9,67	0,26	0,00	9.463	15,93	5,79	15,23	156.419
6					12,66	3,24	6,20	58.952
7					11,12	1,29	3,87	35.453
8					9,92	0,47	0,83	6.436
9					9,66	0,09	0,36	2.751
10					9,57	0,02	0,00	0
Total	75,08	55,19			187,57	118,24		

Tabela 3. C20-T2.5-S4-I1.25-D200K (76,71 MB)

	0,5				0,25			
Iteração	GSP2	GSP2P	Base (MB)	D	GSP2	GSP2P	Base (MB)	D
1	2,14	2,10	76,71	200.000	2,14	2,10	76,71	200.000
2	47,00	47,53	76,71	200.000	57,95	58,39	76,71	200.000
3	13,13	11,61	17,89	142.340	40,07	38,75	41,45	189.220
4	9,50	3,38	10,27	92.760	32,49	16,35	27,97	164.024
5	6,96	2,15	4,90	45.864	22,97	9,62	17,48	122.953
6	5,27	1,06	2,24	20.478	14,06	5,37	9,87	77.408
7	4,36	0,40	1,27	10.958	8,30	2,85	4,47	36.852
8	3,91	0,13	0,18	1.406	5,89	1,49	1,66	12.995
9	3,85	0,02	0,13	1.014	4,70	0,68	0,72	5.263
10	3,80	0,01	0,00	0	4,14	0,26	0,27	1.899
11					3,93	0,07	0,21	1.445
12					3,85	0,03	0,12	751
13					3,80	0,01	0,00	0
Total	99,92	68,39			204,29	135,97		

Tabela 4. Tempo de execução em segundos dos algoritmos *GSP2* e *GSP2P*

	0,5			0,25		
Base	GSP2	GSP2P	GSP2P/GSP2	GSP2	GSP2P	GSP2P/GSP2
C10-T2.5-S4-I1.25-D100K	8,96	6,06	68%	19,02	13,10	69%
C10-T2.5-S4-I1.25-D200K	17,39	11,62	67%	38,26	24,63	64%
C10-T2.5-S4-I1.25-D500K	42,46	27,89	66%	94,39	59,96	64%
C10-T2.5-S4-I1.25-D1000K	75,08	55,19	74%	187,57	118,24	63%
C10-T5.0-S4-I1.25-D200K	103,20	70,85	69%	209,43	144,21	69%
C20-T2.5-S4-I1.25-D200K	99,92	68,39	68%	204,29	135,97	67%
C20-T2.5-S4-I2.5-D500K	158,73	121,20	76%	529,27	329,16	62%
C20-T2.5-S4-I2.5-D1000K	318,07	241,85	76%	1055,80	652,16	62%
Média			70,5%			65,0%

5. Conclusões e Trabalhos Futuros

Neste trabalho, foi proposto o uso de técnicas de redução progressiva da base de dados em algoritmos iterativos para extração de padrões seqüenciais. Os resultados avaliados a partir de diferentes combinações de bases de dados e suportes mínimos mostraram que as técnicas de redução da base implementadas no algoritmo *GSP2P* reduzem significativamente o tempo de execução total do algoritmo sem poda de base *GSP2*.

Nos experimentos realizados, o tempo total de execução do algoritmo *GSP2P* foi, em média, 67,8% do tempo total de execução do algoritmo *GSP2*. A redução progressiva da base pôde ser observada tanto em termos da quantidade de memória necessária para armazenar cada nova base gerada quanto em termos do número total de seqüências.

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*, e (b) definir uma estratégia híbrida, análoga ao algoritmo *kDCI++* de extração de conjuntos freqüentes, no qual a base de dados é progressivamente reduzida até caber em memória principal, quando então é verticalizada e processada não mais em memória secundária.

Referências

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *Procs. of the 20th VLDB International Conf. on Very Large Databases*, pages 487–489.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14.

Ayres, J., Gehrke, J., Yiu, T., and Flannick, J. (2002). Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining*, pages 429–435.

da Ponte, E. N. (2003). Mineração eficiente de padrões seqüenciais através da indexação de seqüências candidatas. *Dissertação de Mestrado*, UFF.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M. C. (2000). Freespan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 355–359.

Orlando, S., Palmerimi, P., Perego, R., Lucchese, C., and Silvestri, F. (2003). kdci: a multi–strategy algorithm for discovering frequent sets in large databases. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*.

Park, J. S., Chen, M., and Yu, P. S. (1995). An effective hash-based algorithm for mining association rules. In *Proceedings of the ACM SIGMOD*, pages 175–186.

Pei, J., Han, J. W., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. C. (2001). Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Procs. of the International Conf. on Data Engineering*, pages 215–224.

Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*, pages 3–17.

Zaki, M. J. (2001). Spade: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1):31–60.