Explorando a Técnica de Indexação de Conjuntos Candidatos na Mineração de Conjuntos Freqüentes

Adriana Prado[1] **, Alexandre Plastino**[1] **(Orientador)**

[1]Instituto de Computação – Universidade Federal Fluminense (UFF),
Rua Passo da Pátria, 156 – Bloco E – 3^o andar – Boa Viagem,
24210-240 – Niterói – RJ – Brasil

{aprado, plastino}@ic.uff.br

Abstract. *The algorithm* kDCI++ *is considered a state-of-the-art strategy for frequent itemset mining. However, computational experiments have shown that this algorithm does not present a good performance for low minimum supports on sparse databases. Aiming at improving* kDCI++*, in this work, we present the* kDCI-3 *algorithm. In this proposal, an efficient candidate accessing method is explored, which reduces the total execution time of the* kDCI++ *algorithm. Experimental results have shown that* kDCI-3 *outperforms* kDCI++ *in the conducted experiments. When compared to other important algorithms,* kDCI-3 *shows up as a more competitive version of* kDCI++*.*

Resumo. *O algoritmo* kDCI++ *é considerado uma importante e eficiente estratégia para mineração de conjuntos freqüentes da atualidade. Entretanto, experimentos computacionais mostram que este algoritmo não apresenta um bom desempenho quando suportes baixos são considerados sobre bases de dados esparsas. A fim de aprimorar seu desempenho, neste trabalho, é apresentado o algoritmo* kDCI-3*. Nesta proposta, explora-se um método de acesso mais eficiente a conjuntos candidatos, reduzindo o tempo total de processamento. Os experimentos realizados mostram que o* kDCI-3 *reduz significativamente o tempo total de execução do* kDCI++*. Quando comparado a outras importantes estratégias, o* kDCI-3 *apresenta-se como uma versão mais competitiva do* kDCI++*.*

1. Introdução

Conjuntos freqüentes são conjuntos de itens que aparecem em pelo menos s% (s é chamado suporte mínimo) das transações em uma base de dados. A extração de tais conjuntos representa a fase mais custosa, computacionalmente, do proceso de mineração de Regras de Associação [Agrawal and Srikant 1994]. Por esta razão, ao longo dos últimos dez anos, várias estratégias para extração de conjuntos freqüentes têm sido propostas.

Com o objetivo de comparar o desempenho destas estratégias, em 2003, foi organizado um *Workshop* de implementações de algoritmos para mineração de conjuntos freqüentes (*IEEE/ICDM FIMI'03*) [Goethals and Zaki 2003]. De acordo com os resultados obtidos, o algoritmo *kDCI++* [Lucchese et al. 2003] foi considerado um dos principais algoritmos da atualidade para esta tarefa. Entretanto, analisando seus resultados mais detalhadamente, observa-se que o algoritmo *kDCI++* não se mostra tão eficiente quando valores baixos de suporte mínimo são considerados sobre bases de dados esparsas.

Como uma contribuição preliminar deste trabalho, a fim de melhor compreender o comportamento do algoritmo *kDCI++* considerando tal cenário, foram realizados vários experimentos sobre diferentes combinações de treze bases de dados esparsas (sintéticas e reais) e cinco valores de suporte mínimo, também avaliados no *Workshop FIMI'03* e em outros trabalhos relacionados. De acordo com os resultados obtidos, identificou-se que a terceira iteração do algoritmo *kDCI++* apresenta um custo computacional bastante elevado se comparado ao custo computacional das demais iterações. Nos experimentos realizados com suporte mais baixo, a terceira iteração consome, em média, 59% do tempo total de execução. Este comportamento pode ser em grande parte justificado pela grande quantidade de conjuntos candidatos a freqüentes a ser avaliada nesta iteração.

Neste contexto, a fim de aprimorar o desempenho do *kDCI++* e torná-lo também eficiente sobre bases de dados esparsas quando valores de suporte baixos são considerados, apresenta-se, como principal contribuição deste trabalho, o algoritmo *kDCI-3*. Nesta proposta, através de um novo método de acesso, os candidatos são acessados diretamente, de forma mais eficiente, não apenas durante as duas iterações iniciais, mas especialmente durante a terceira iteração, avaliada como tendo um alto custo computacional.

2. O Algoritmo *kDCI++*

O algoritmo *kDCI++* é uma estratégia iterativa híbrida para mineração de conjuntos freqüentes, que tem por base o algoritmo *Apriori* [Agrawal and Srikant 1994].

Em suas primeiras iterações, o *kDCI++* utiliza estruturas de dados específicas para acesso aos conjuntos candidatos a freqüentes, além de adotar procedimentos que reduzem gradativamente a base de dados durante a execução. Durante a primeira e segunda iterações, na contagem do suporte dos candidatos, estes são acessados diretamente a partir de uma estrutura de dados baseada em um vetor. Já a partir da terceira iteração, um candidato `c` é acessado executando-se uma busca seqüencial sobre o conjunto de candidatos com o mesmo prefixo de tamanho 2 presente em `c`.

Quando a base de dados, reduzida gradativamente durante a execução, se torna pequena o suficiente para ser carregada em memória, o algoritmo passa a utilizar uma representação verticalizada desta base em memória principal. Tal representação é composta por um conjunto de m listas de bits de tamanho n, onde m é o número de itens e n, o número de transações. Cada lista está associada a um item da base de dados e cada bit desta lista está associado a uma transação. Nesta nova fase, os suportes dos candidatos são calculados a partir de interseções das listas de bits associadas aos itens que os compõe.

3. O Algoritmo *kDCI-3*

Com o objetivo de tornar o cálculo do suporte dos candidatos de tamanho 3 mais eficiente, o algoritmo proposto utiliza um novo método de acesso aos candidatos baseado em uma tabela de prefixos (T_3) e em um vetor (C), ilustrados na Figura 1.

O vetor C representa os conjuntos candidatos de tamanho 3, em ordem lexicográfica. Os candidatos são gerados combinando-se os pares de conjuntos freqüentes de tamanho 2 que possuem o mesmo prefixo de tamanho 1.

Em T_3, cada entrada representa um prefixo de tamanho 2 presente nos conjuntos candidatos de tamanho 3 e é composta por dois campos. O primeiro campo armazena a

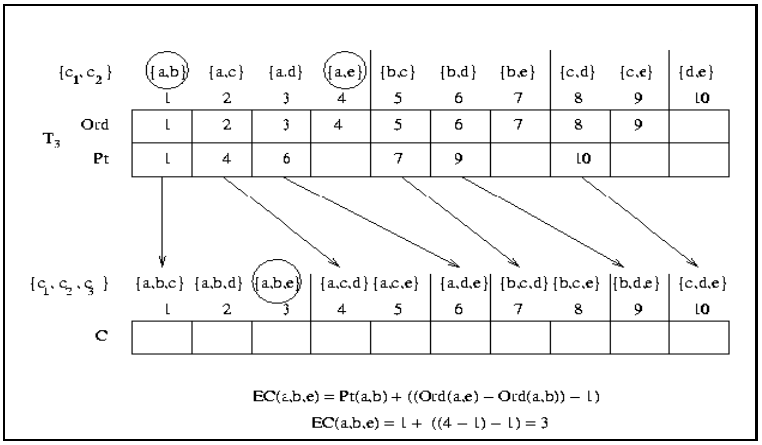

Figura 1. A estrutura de dados utilizada pelo algoritmo *kDCI-3*.

ordem, representada por $Ord(c_1, c_2)$, em que o conjunto freqüente $\{c_1, c_2\}$ de tamanho 2 é localizado em T_3. Se $\{c_1, c_2\}$ não é um conjunto freqüente, este valor é nulo. O segundo campo, representado por $Pt(c_1, c_2)$, aponta para a primeira entrada do vetor C que representa um candidato com o prefixo $\{c_1, c_2\}$. Este campo é nulo caso $\{c_1, c_2\}$ não tenha gerado nenhum candidato com este mesmo prefixo.

A entrada correspondente a um conjunto candidato genérico $c = \{c_1, c_2, c_3\}$ em C, aqui chamada de $EC(c_1, c_2, c_3)$, é definida pela Equação 1. Observe que esta entrada pode ser encontrada a partir de $Pt(c_1, c_2)$ mais um deslocamento, definido pelo número de conjuntos freqüentes de tamanho 2 representados entre $\{c_1, c_2\}$ e $\{c_1, c_3\}$ em T_3.

$$EC(c_1, c_2, c_3) = Pt(c_1, c_2) + ((Ord(c_1, c_3) - Ord(c_1, c_2)) - 1) \; . \qquad (1)$$

Note que, desta forma, candidatos de mesmo prefixo de tamanho 2 são acessados diretamente, em tempo constante $O(1)$, sem que um algoritmo de busca seqüencial $O(n)$ tenha que ser executado sobre um conjunto de candidatos (suponha de cardinalidade n), como ocorre no algoritmo *kDCI++*.

Além da utilização desta nova estratégia, foi desenvolvido para o algoritmo *kDCI-3* uma técnica de gerenciamento de memória que possibilita a contagem de blocos de candidatos isoladamente, de modo que a quantidade de memória necessária para a contagem de cada bloco não ultrapasse a quantidade de memória principal disponível.

4. Resultados Computacionais

Nesta seção, é apresentada uma síntese da análise comparativa entre o algoritmo proposto *kDCI-3* e os principais algoritmos de extração de conjuntos freqüentes segundo a avaliação do *Workshop FIMI'03* [Goethals and Zaki 2003]. São eles: *kDCI++*, *PatriciaMine*, *FPgrowth** e *LCMfreq*.

Os experimentos foram realizados em uma máquina de processador *Pentium* III, 600 Mhz, com 256 *megabytes* de memória principal e sistema operacional Fedora Linux 2.4.22. Utilizaram-se dez bases de dados sintéticas e três bases de dados reais disponíveis na página do *Workshop FIMI'03*: *BMSPOS*, *Kosarak* e *Retail*. Sobre cada base de dados, consideraram-se cinco valores de suporte mínimo. Para cada combinação base/suporte,

cada um dos cinco algoritmos foram executados três vezes (considerou-se a média dos resultados obtidos), levando a um total de novecentos e setenta e cinco execuções.

Os resultados computacionais mostraram que, considerando-se os valores de suporte baixos, o tempo da terceira iteração consumido pelo algoritmo proposto *kDCI-3* foi significativamente inferior ao tempo consumido pelo *kDCI++*. Para os suportes mais críticos, o tempo do *kDCI-3* nesta iteração foi, em média, 27% (variando de 5% a 88%) do tempo do *kDCI++*, logo alcançando uma redução média de 73%. Como conseqüência, o tempo total de execução do *kDCI-3* variou de 29% a 97%, representando, em média, 53% do tempo total de execução do *kDCI++*. A redução de tempo foi verificada mesmo quando, na terceira iteração, a base já havia sido verticalizada pelo *kDCI++*.

A Figura 2 apresenta um resumo dos resultados obtidos com o *kDCI-3*, observados a partir de diversos tipos de análise realizados. O gráfico (a) apresenta o tempo total de execução relativo (eixo y) dos algoritmos *kDCI-3* e *kDCI++*, considerando-se os valores de suporte (eixo x) avaliados para uma base de dados esparsa com 1.000 itens e 200.000 transações. O valor 1 no eixo y representa o tempo relativo de execução do algoritmo que obteve o pior desempenho. Para os valores de suporte mais altos (5 e 2.5), casos em que o número de candidatos gerados é menor e o tempo total de processamento é pequeno, os desempenhos das duas estratégias são semelhantes. Por outro lado, com suportes menores (1, 0.5 e 0.25) e o conseqüente aumento do número de candidatos, observa-se que o tempo de execução do *kDCI-3* é significativamente menor do que o apresentado pelo *kDCI++*, devido à maior eficiência do novo método de acesso aos candidatos.

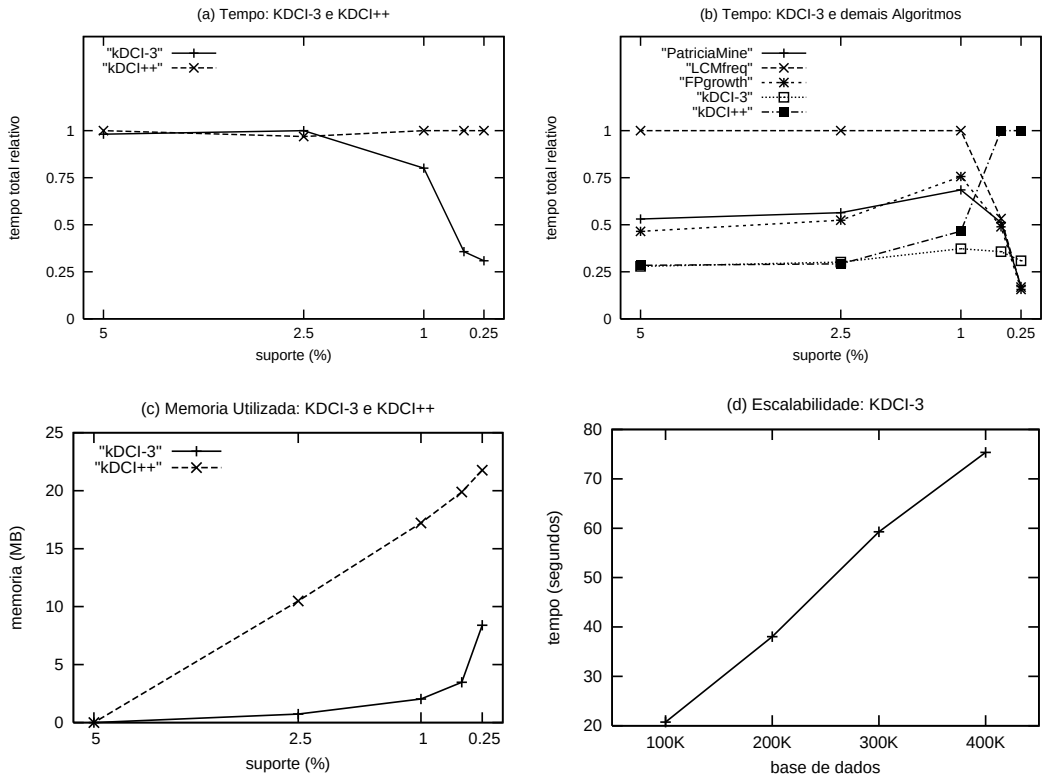

Figura 2. Avaliação de desempenho do algoritmo proposto *kDCI-3*.

O gráfico (b) ilustra o comportamento dos algoritmos *kDCI-3* e *kDCI++* quando comparados às demais estratégias. Apresentam-se os tempos relativos de cada estratégia, para a mesma base e mesmos suportes mínimos do gráfico (a). Observa-se que, para os suportes mais altos (5, 2.5 e 1), os algoritmos *kDCI-3* e *kDCI++* apresentam o melhor desempenho. Com a redução do suporte, observa-se o impacto da incorporação da nova técnica de acesso aos candidatos. Com suporte 0.5, o *kDCI++* apresenta o pior desempenho e o *kDCI-3* apresenta o melhor. Com o menor suporte, confirma-se que a nova técnica torna o *kDCI++* mais competitivo.

O gráfico (c) apresenta a quantidade de memória utilizada (eixo y) pelos algoritmos *kDCI-3* e *kDCI++*, nas execuções referentes ao gráfico (a). Observa-se a possibilidade de uma economia significativa de memória quando utilizada a estrutura de acesso do *kDCI-3*. O gráfico (d) ilustra a escalabilidade do algoritmo *kDCI-3*. Percebe-se que o tempo de execução (eixo y) cresce linearmente com o tamanho da base de dados (eixo x).

5. Conclusões

A principal contribuição deste trabalho foi o desenvolvimento do algoritmo de mineração de conjuntos freqüentes *kDCI-3*, uma extensão do importante algoritmo *kDCI++*. Neste novo algoritmo, propõe-se um método de acesso a conjuntos candidatos mais eficiente, permitindo uma redução do seu tempo total de processamento.

Os resultados obtidos sobre treze bases de dados esparsas (sintéticas e reais), utilizando-se valores de suporte baixos, mostraram que a técnica de contagem adotada pelo algoritmo *kDCI-3* reduz significativamente o tempo de execução do *kDCI++*.

Os resultados obtidos neste trabalho foram apresentados em [Prado et al. 2003] e [Prado et al. 2004]. A técnica aqui proposta foi recentemente incorporada ao algoritmo *kDCI++*, pelos seus autores, conforme apresentado em [Orlando et al. 2004].

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*.

Goethals, B. and Zaki, M. J. (2003). Advances in frequent itemset mining implementations: Introduction to fimi'03. In *Procs. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*.

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

Orlando, S., Palmerimi, P., Perego, R., Lucchese, C., and Silvestri, F. (2004). kdci: on using direct count up to the third iteration. In *In Procs. of the Second IEEE ICDM Workshop on Frequent Itemset Mining Implementations*.

Prado, A., Targa, C., and Plastino, A. (2003). Improving direct counting for frequent set mining. Technical Report RT-02/03, Universidade Federal Fluminense.

Prado, A., Targa, C., and Plastino, A. (2004). Improving direct counting for frequent itemset mining. In *Procs. of the 6th DAWAK International Conf. on Data Warehousing and Knowledge Discovery, LNCS 3181*.