

Aprendizado Descritivo

Aula 03 – Mineração de itens frequentes: Apriori e Eclat

Professor Renato Vimieiro

DCC/ICEX/UFMG

Introdução

- Como vimos na aula anterior, o principal problema do algoritmo ingênuo para mineração de conjuntos de itens frequentes era a replicação de esforços para avaliar o suporte dos candidatos
- As múltiplas passadas no conjunto de dados (armazenado em memória secundária) torna o algoritmo impraticável até mesmo para pequenos volumes
- Os algoritmos que veremos hoje exploram propriedades do problema para amortizar o custo da computação de suporte, e evitar retrabalho na avaliação dos candidatos

Apriori

- O Apriori foi proposto por Rakesh Agrawal e Ramakrishnan Srikant em 1994
 - O artigo possui mais de 30K citações
- Os autores à época trabalhavam no projeto da IBM para o Wal-Mart
- A ideia central é evitar computações desnecessárias para candidatos infrequentes
- Isso é viabilizado pela propriedade de ***anti-monotonicidade*** da função suporte
- Essa é uma das propriedades mais importantes para a área



R. Agrawal
Data Insights Labs



R. Srikant
Google Fellow

Anti-monotonicidade do suporte

- Considere dois itemsets A e B quaisquer. Se $A \subseteq B$, então $\text{sup}(A) \geq \text{sup}(B)$.
- Essa observação nos diz que a cobertura de conjuntos de itens é, no máximo, tão grande quanto a de seus subconjuntos
 - No caso mais simples, um conjunto de dois itens não pode ocorrer em mais transações que cada um dos itens individualmente
- Consequentemente, se o itemset A é infrequente, B também será.
- Isso define a propriedade de anti-monotonicidade da função suporte, também conhecida como a propriedade do Apriori
 - **Todo superconjunto de um conjunto infrequente é infrequente**
 - **Todo subconjunto de um conjunto frequente é frequente**

R + importante

Apriori

- O Apriori utiliza uma busca em largura no espaço de busca para minerar os padrões
 - Frequentemente, o termo usado na literatura é abordagem por níveis (level-wise approach)
- A busca inicia com a identificação dos itens frequentes
- Depois, os conjuntos de tamanho k são explorados antes dos de tamanho $k+1$
- Assim como o algoritmo ingênuo, ele também opera em duas etapas:
 - Geração de candidatos
 - Cômputo do suporte e eliminação dos infrequentes

Apriori

- A geração dos candidatos é feita a partir dos conjuntos frequentes encontrados na fase anterior
- Conjuntos compartilhando um prefixo de $k-1$ itens são combinados para gerar candidatos de tamanho $k+1$
 - Novamente, assume-se que eles são ordenados pela ordem lexicográfica
- Candidatos que possuam algum subconjunto infrequente são descartados imediatamente
 - A propriedade do Apriori é empregada
- Os suportes dos candidatos são atualizados com uma única passada no conjunto de dados
 - Subconjuntos de tamanho k de cada transação são usados para atualizar o suporte dos candidatos

Apriori

2ⁱ pacⁱ

APRIORI ($\mathbf{D}, \mathcal{I}, \text{minsup}$):

```
1  $\mathcal{F} \leftarrow \emptyset$ 
2  $\mathcal{C}^{(1)} \leftarrow \{\emptyset\}$  // Initial prefix tree with single items
3 foreach  $i \in \mathcal{I}$  do Add  $i$  as child of  $\emptyset$  in  $\mathcal{C}^{(1)}$  with  $\text{sup}(i) \leftarrow 0$ 
4  $k \leftarrow 1$  //  $k$  denotes the level
5 while  $\mathcal{C}^{(k)} \neq \emptyset$  do
6   COMPUTESUPPORT ( $\mathcal{C}^{(k)}, \mathbf{D}$ )
7   foreach leaf  $X \in \mathcal{C}^{(k)}$  do
8     if  $\text{sup}(X) \geq \text{minsup}$  then  $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, \text{sup}(X))\}$ 
9     else remove  $X$  from  $\mathcal{C}^{(k)}$ 
10   $\mathcal{C}^{(k+1)} \leftarrow \text{EXTENDPREFIXTREE } (\mathcal{C}^{(k)})$ 
11   $k \leftarrow k + 1$ 
12 return  $\mathcal{F}^{(k)}$ 
```

Apriori

para cada transacción
sección es 'ítems'

COMPUTESUPPORT ($\mathcal{C}^{(k)}$, D):

```
13 foreach  $\langle t, \mathbf{i}(t) \rangle \in D$  do
14   foreach  $k$ -subset  $X \subseteq \mathbf{i}(t)$  do
15     if  $X \in \mathcal{C}^{(k)}$  then  $sup(X) \leftarrow sup(X) + 1$ 
```

ia no tiene e
inventario <

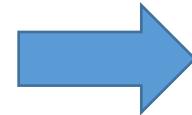
EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$):

```
16 foreach leaf  $X_a \in \mathcal{C}^{(k)}$  do
17   foreach leaf  $X_b \in \text{SIBLING}(X_a)$ , such that  $b > a$  do
18      $X_{ab} \leftarrow X_a \cup X_b$ 
      // prune candidate if there are any infrequent subsets
19     if  $X_j \in \mathcal{C}^{(k)}$ , for all  $X_j \subset X_{ab}$ , such that  $|X_j| = |X_{ab}| - 1$  then
20       Add  $X_{ab}$  as child of  $X_a$  with  $sup(X_{ab}) \leftarrow 0$ 
21     if no extensions from  $X_a$  then
22       remove  $X_a$ , and all ancestors of  $X_a$  with no extensions, from  $\mathcal{C}^{(k)}$ 
23 return  $\mathcal{C}^{(k)}$ 
```

Apriori

- Exemplo (minsup=3):

TID	Muesli	Oats	Milk	Yoghurt	Biscuits	Tea
1	1	0	1	1	0	1
2	0	1	1	0	0	0
3	0	0	1	0	1	1
4	1	0	0	1	0	0
5	0	1	1	0	0	1
6	1	0	1	0	0	1



TID	Muesli	Milk	Tea
1	1	1	1
2	0	1	0
3	0	1	1
4	1	0	0
5	0	1	1
6	1	1	1

Apriori

6

Dateset ameaça as transações mais longas

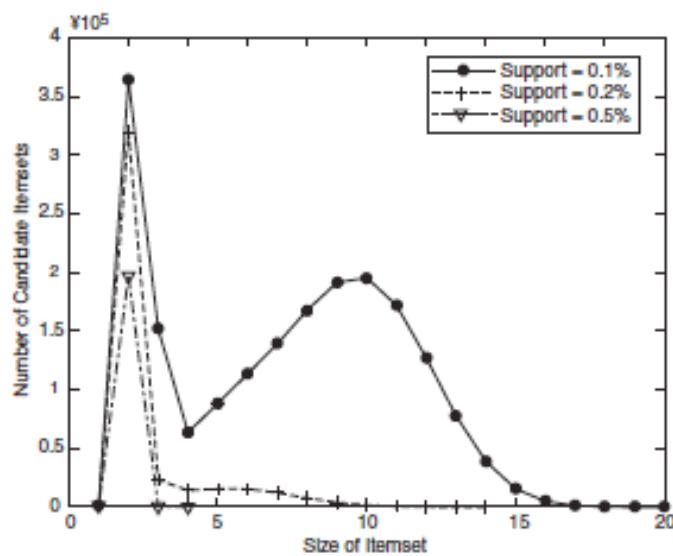
- O número de passadas é drasticamente reduzido em relação ao algoritmo ingênuo
 - $O(2^I) \rightarrow O(I)$
- As podas baseadas na anti-monotonicidade do suporte também são bastante efetivas na prática
- O algoritmo também apresenta alguns problemas:
 - Busca em largura requer que todos os candidatos de um nível sejam mantidos em memória. Esse custo é proibitivo em alguns (muitos) casos
 - Tanto a contagem do suporte quanto a poda do Apriori podem ser consideravelmente caras, dependendo da implementação

Apriori

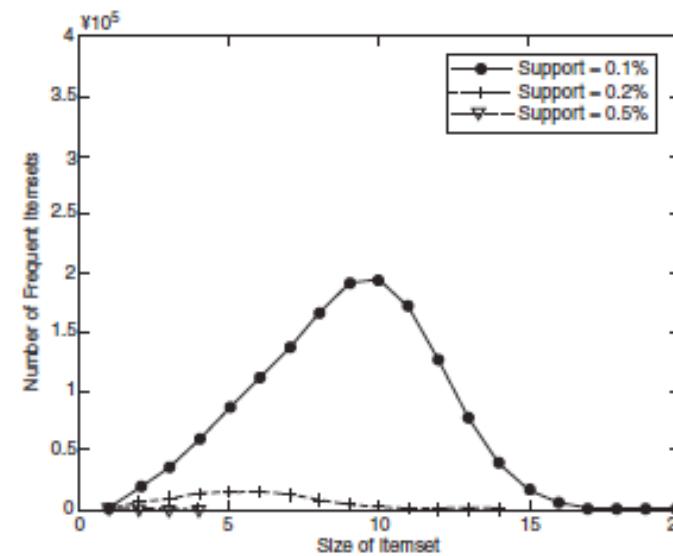
- O custo de memória é inerente à abordagem, e não podemos fazer muita coisa para melhorá-lo
- O custo da contagem e verificação pode ser atenuado, usando estruturas de dados mais ‘sofisticadas’
- Existem duas abordagens mais comuns:
 - Usar uma árvore hash
 - Usar uma árvore de prefixos (Trie)
- Na primeira abordagem, cada nó folha armazena um conjunto de candidatos/conjuntos frequentes
 - O número de comparações é reduzido
- Na segunda abordagem, os nós da Trie armazenam os candidatos/conjuntos frequentes. Os subconjuntos das transações são usadas para indexá-la e atualizar o suporte

Apriori

- A redução do suporte mínimo tem um impacto muito grande no custo computacional do algoritmo
 - O tamanho dos candidatos aumenta -> Mais candidatos são avaliados em cada nível -> o tamanho dos conjuntos frequentes aumenta -> mais níveis são explorados



(a) Number of candidate itemsets.



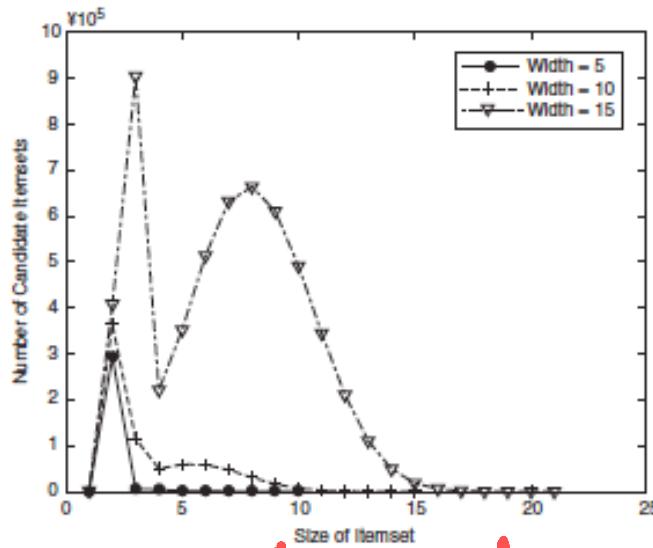
(b) Number of frequent itemsets.

Apriori

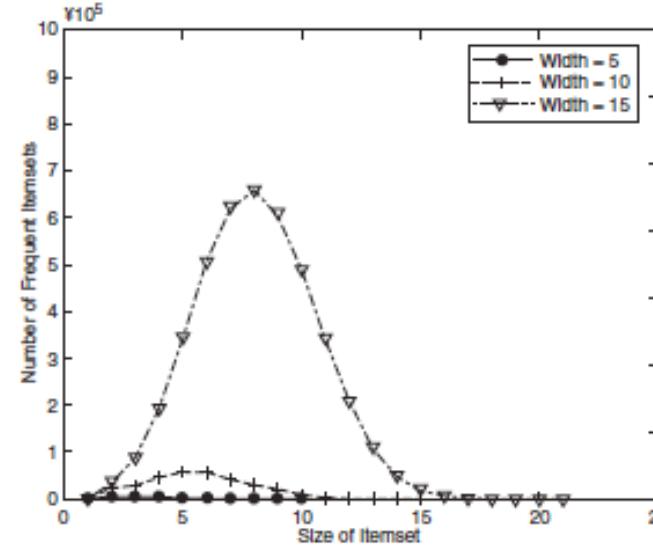
trie o árvore com elementos
para calcular o suporte

$$\begin{matrix} & a_0 \dots a_{995} \\ 3 & (1000) \\ 3 & \end{matrix}$$

- A densidade da base de dados também tem muito impacto no custo
 - Transações passam a ter mais itens
 - Isso tem duas implicações: tamanho médio dos itemsets aumentam; mais subconjuntos são gerados durante a contagem do suporte $\binom{|t|}{k}$ → transações mais longas
→ custoso



(a) Number of candidate itemsets.



(b) Number of Frequent Itemsets.

↳ um menor mā vai estar na
trie para ser abolido → suporte

↳ Implementação rápida
↳ Problema suporte
↳ gera muita saída que mā vai precisar

↳ triagem funciona relativamente bem

↳ problema → transações longas.

Eclat (Equivalence Class Transformation)

- Dadas as deficiências do Apriori, M. Zaki propôs, em 2000, o algoritmo Equivalence Class Transformation (Eclat)
- A proposta do algoritmo é ‘eliminar’ a necessidade de passadas no conjunto de dados para computar o suporte
- Para isso, ele parte de uma representação vertical dos dados, e se baseia no fato de que a cobertura da união de dois itemsets é a interseção de suas coberturas

A cobertura da união é a intersecção das coberturas.



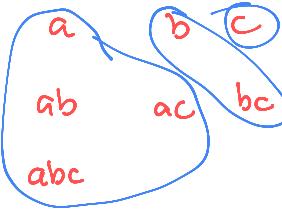
Prof. Mohammed J. Zaki
Rensselaer Polytechnic Institute

Eclat

- Ou seja, a ideia central do algoritmo tentar manter os tidsets em memória principal para computar o suporte dos itemsets através de interseções desses conjuntos
- Contudo, todos os tidsets podem não caber na memória principal. Assim, é necessário algum mecanismo que possibilite a divisão do espaço de busca em subproblemas independentes que caibam na memória
- A divisão é feita conforme uma relação de equivalência estabelecida sobre os candidatos

Eclat

→ Particionamos itens → onde pendentes.



A partição de b não
vai ter os pais ou os pais
de b e vai ser menor
para esquerda.

Avalia de direita
para esquerda.

- Seja $p: P(I) \times \mathbb{N} \rightarrow P(I)$ uma função prefixo. $p(X, k) = X[1:k]$.
- A relação $\theta_k \subseteq P(I) \times P(I)$, $A \theta_k B \equiv p(A, k) = p(B, k)$, é uma relação de equivalência
- Dessa forma, ela induz uma partição dos conjuntos de itens em classes de equivalência, onde todos os elementos compartilham um certo prefixo
- Por exemplo, todos os conjuntos que contêm o item Muesli pertencem à classe de equivalência $[Muesli]_{\theta_1}$
- Intuitivamente, essas classes servem como projeções do conjunto de dados, em que somente as transações contendo aquele prefixo são consideradas *faz projeções que cabem em memória e atuam*

Eclat

→ nos preços.

- Durante a busca em profundidade, o algoritmo partitiona os conjuntos de itens conforme a relação de equivalência e o nível da árvore
- O particionamento pode ser encerrado tão logo os tidsets caibam na memória e as interseções possam ser computadas facilmente
- Contudo, a estratégia pode ser usada durante toda a execução do algoritmo
- O cálculo do suporte no algoritmo se restringe a calcular o tamanho do tidset

Eclat

Dois formas de usar o Eclat:

- ↳ a ordem que cominha os elementos
- ↳ e a forma de particionar sem preio.

ALGORITHM 8.3. Algorithm ECLAT

```
// Initial Call:  $\mathcal{F} \leftarrow \emptyset, P \leftarrow \{\langle i, t(i) \rangle \mid i \in \mathcal{I}, |t(i)| \geq \text{minsup}\}$ 
ECLAT ( $P, \text{minsup}, \mathcal{F}$ ):
1 foreach  $\langle X_a, t(X_a) \rangle \in P$  do
2    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, \text{sup}(X_a))\}$ 
3    $P_a \leftarrow \emptyset$ 
4   foreach  $\langle X_b, t(X_b) \rangle \in P$ , with  $X_b > X_a$  do
5      $X_{ab} = X_a \cup X_b$   $(k+1)$ 
6      $t(X_{ab}) = t(X_a) \cap t(X_b)$ 
7     if  $\text{sup}(X_{ab}) \geq \text{minsup}$  then
8        $P_a \leftarrow P_a \cup \{(X_{ab}, t(X_{ab}))\}$ 
9   if  $P_a \neq \emptyset$  then ECLAT ( $P_a, \text{minsup}, \mathcal{F}$ )
```

subconjunto
→ base de
itens no vertical

$(i, t(i))$
 $(a, t(a))$
 $(b, t(b))$

se um quero
 $ab \rightarrow$ intersecção de $t(a) \cup t(b)$

o que é?

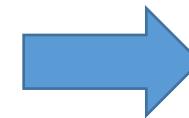
deixar aqui
combinacão
com itens parce
grande.

Figura retirada de Zaki e Meira (2014)

Representações de conjuntos de dados

Suponha o minimo = 3

TID	Muesli	Oats	Milk	Yoghurt	Biscuits	Tea
1	1	0	1	1	0	1
2	0	1	1	0	0	0
3	0	0	1	0	1	1
4	1	0	0	1	0	0
5	0	1	1	0	0	1
6	1	0	1	0	0	1



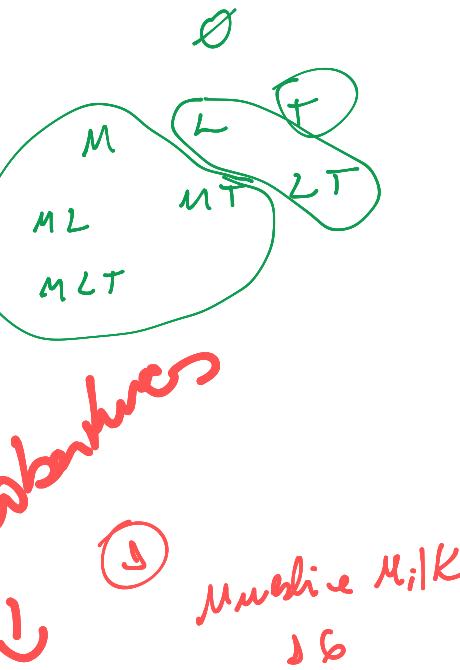
$[\emptyset]_{\theta_0}$

x	c(x)
Muesli	146
Milk	12356
Tea	1356

② Milk e Tea
1356

Muesli Tea
146

Muesli Milk Tea
12356



Eclat

- O custo computacional do algoritmo está diretamente relacionado ao tamanho dos tidsets
 - O tempo de execução depende do cálculo da interseção dos tidsets
- O custo de espaço também é dependente do tamanho. Quanto mais denso o conjunto de dados, mais largos serão os tidsets
- Há duas formas de se implementar o algoritmo:
 - Usando vetores de bits
 - Usando vetores de Ids
- Os vetores de bits são interessantes para o cálculo do suporte
 - Eles facilitam a computação da interseção e o cálculo do tamanho do tidset pode ser feito usando uma tabela auxiliar (palavras de 16bits mapeadas para valores)
- Contudo, se o conjunto for esparsa, isso representará um desperdício muito grande de espaço. Então vetores de Ids se tornam mais interessantes.
 - As computações de interseção são feitas como na função merge do mergesort

interseção de dois vetores intensivo



Prof. Karam Gouda
Benha University, Egito

Diffssets e dEclat

- Percebendo o problema de se manter os tidsets em memória, Zaki e Gouda propuseram em 2001 (o artigo só foi publicado em 2003) uma solução alternativa
- Eles propuseram armazenar as diferenças entre os tidsets dos membros de uma classe e dos prefixos que a definem
- Eles chamaram esse conjunto de **diffset**
- Formalmente, para um prefixo P e um itemset PX, o diffset de X,
 $d(PX) = c(P) - c(PX)$
- Seriam armazenados, portanto, o suporte do itemset e seu diffset

Diffsets e dEclat

Calculo do suporte

- O fato é que, se somente os diffsets são armazenados, o suporte não é mais obtido como a cardinalidade desse conjunto
- Dessa forma, como calcular o suporte de um itemset PXY obtido a partir de outros dois PX e PY, usando somente seus diffsets?
- O suporte de PX é calculado pela diferença entre o suporte de P e o diffset de PX, $\text{sup}(PX) = \text{sup}(P) - |d(PX)|$
 - Portanto, $\text{sup}(PXY) = \text{sup}(PX) - |d(PXY)|$
- A solução passa por computar o diffset de PXY. Porém, temos somente os diffsets de PX e PY

Support do pair

Diffssets e dEclat

$$\begin{aligned} & (c(P_X) \cap \overline{c(P_Y)}) \cup (c(P) \cap \overline{c(P)}) \\ & (c(P_X) \cup c(P)) \cap (c(P_X) \cup \overline{c(P)}) \cap (\overline{c(P_Y)} \cup c(P)) \cap (\overline{c(P_Y)} \cup \overline{c(P)}) \\ & \overbrace{(c(P_X) \cup \overline{c(P)})}^{\text{d}(P_X)} \equiv \overbrace{\frac{(c(P_X) \cap c(P))}{c(P) - c(P_X)}}^{\text{d}(P)} \end{aligned}$$

- Por definição, $d(PXY) = c(PX) - c(PXY) = c(PX) - c(PY)$
- Podemos adicionar, ao conjunto acima, o conjunto vazio ($c(P) - c(P)$) sem alterá-lo
- Logo, $d(PXY) = c(PX) - c(PY) + c(P) - c(P) = (c(P) - c(PY)) - (c(P) - c(PX)) = d(PY) - d(PX)$
- Em outras palavras, podemos usar os diffsets dos conjuntos base para calcular o diffset do novo candidato
- A variante do Eclat que usa diffsets ficou conhecida como dEclat

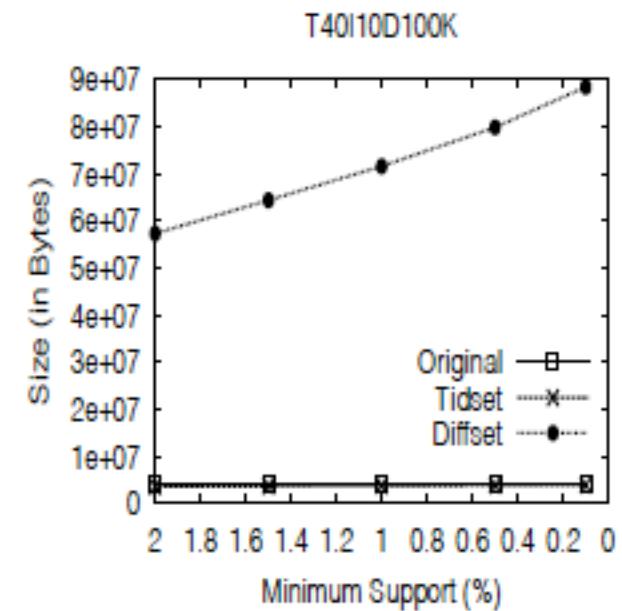
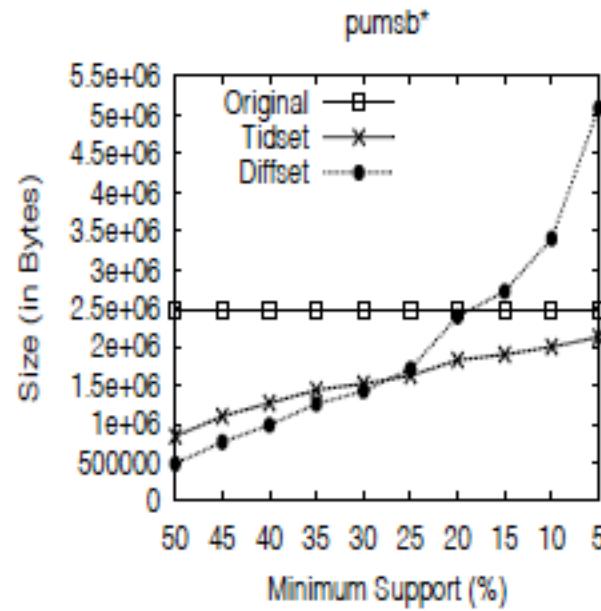
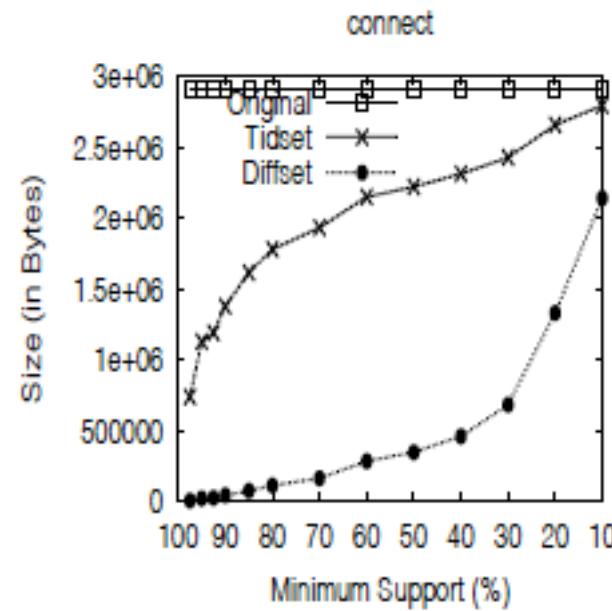
Diffssets e dEclat

ALGORITHM 8.4. Algorithm DECLAT

```
// Initial Call:  $\mathcal{F} \leftarrow \emptyset$ ,  
     $P \leftarrow \{\langle i, \mathbf{d}(i), \text{sup}(i) \rangle \mid i \in \mathcal{I}, \mathbf{d}(i) = \mathcal{T} \setminus \mathbf{t}(i), \text{sup}(i) \geq \text{minsup}\}$   
DECLAT ( $P, \text{minsup}, \mathcal{F}$ ):  
1 foreach  $\langle X_a, \mathbf{d}(X_a), \text{sup}(X_a) \rangle \in P$  do  
2    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, \text{sup}(X_a))\}$   
3    $P_a \leftarrow \emptyset$   
4   foreach  $\langle X_b, \mathbf{d}(X_b), \text{sup}(X_b) \rangle \in P$ , with  $X_b > X_a$  do  
5      $X_{ab} = X_a \cup X_b$   
6      $\mathbf{d}(X_{ab}) = \mathbf{d}(X_b) \setminus \mathbf{d}(X_a)$   
7      $\text{sup}(X_{ab}) = \text{sup}(X_a) - |\mathbf{d}(X_{ab})|$   
8     if  $\text{sup}(X_{ab}) \geq \text{minsup}$  then  
9        $P_a \leftarrow P_a \cup \{\langle X_{ab}, \mathbf{d}(X_{ab}), \text{sup}(X_{ab}) \rangle\}$   
10  if  $P_a \neq \emptyset$  then DECLAT ( $P_a, \text{minsup}, \mathcal{F}$ )
```

Diffssets e dEclat

- Essa abordagem se mostrou muito eficiente para conjuntos densos
- Porém, em conjuntos esparsos, o algoritmo original é a melhor opção



Figuras retiradas de Zaki e Gouda (2001)

Leitura

- Seções 8.1, 8.2 (Zaki e Meira)
- Seções 6.1, 6.2 (Introduction to Data Mining)
- Mohammed Javeed Zaki: Scalable Algorithms for Association Mining. IEEE Trans. Knowl. Data Eng. 12(3): 372-390 (2000)
- Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. Technical Report 01-1, Computer Science Dept., Rensselaer Polytechnic Institute (March 2001) 10
- Christian Borgelt. Efficient Implementations of Apriori and Eclat. Workshop of Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL, USA).

Aprendizado Descritivo

Aula 03 – Mineração de itens frequentes: Apriori e Eclat

Professor Renato Vimieiro

DCC/ICEX/UFMG