ITS
Institut
Teknologi
Sepuluh Nopember

TESIS - CI2541

PERBAIKAN ALGORITMA CHARM UNTUK PENGGALIAN FREQUENT CLOSED ITEMSETS

MARDIYANTO
NRP : 5105 201 701

DOSEN PEMBIMBING
Prof. Dr. Ir. Arif Djunaidy, M.Sc.

PROGRAM MAGISTER
BIDANG KEAHLIAN TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2007

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M. Kom)
di
Institut Teknologi Sepuluh Nopember

Oleh :
Mardiyanto
Nrp. 5105 201 701

Tanggal Ujian : 7 Agustus 2007
Periode Wisuda : September 2007

Disetujui oleh :

1. **Prof. Dr. Ir. Arif Djunaidy, M.Sc.** .....
   NIP : 131 633 403

2. **Ir. F.X. Arunanto, M.Sc.** .....
   NIP. 131 285 253

3. **Daniel Oranova Siahaan, S.Kom., M.Sc., P.D.Eng.** .....
   NIP. 132 318 029

4. **Ahmad Saikhu, S.Si., M. Kom.** .....
   NIP. 132 318 030

Direktur Program Pascasarjana

DEPARTEMEN PENDIDIKAN NASIONAL INSTITUT TEKNOLOGI SEPULUH NOPEMBER PROGRAM PASCASARJANA

Prof. Ir. Happy Ratna S., M.Sc., Ph.D.
NIP. 130 541 829

PERBAIKAN ALGORITMA CHARM UNTUK PENGGALIAN FREQUENT CLOSED ITEMSETS

Nama mahasiswa : Mardiyanto
NRP : 5105 201 701
Pembimbing : Prof. Dr. Ir. Arif Djunaidy, M.Sc.

ABSTRAK

Penggalian *frequent closed itemsets* merupakan salah satu bagian penting dari penggalian kaidah asosiasi (*association rule*) karena dapat secara unik menentukan himpunan semua *frequent itemsets* dan *support*nya. Berbagai algoritma penggalian *frequent closed itemsets* telah ditemukan, diantaranya adalah algoritma CHARM dan algoritma DCI_CLOSED. Algoritma CHARM menggunakan format data vertikal *diffset* dan metode *subsumption check* untuk melakukan pemeriksaan duplikasi. Metode *subsumption check* tidak efisien karena memerlukan penyimpanan semua *frequent closed itemsets* sebelumnya di memori utama. Algoritma DCI_CLOSED menggunakan format data vertikal *bitvectors* dan menggunakan metode *order preserving* untuk melakukan pemeriksaan duplikasi. Metode *order preserving* efisien karena tidak memerlukan penyimpanan *frequent closed itemsets* sebelumnya di memori utama. Berdasarkan penelitian dan teori yang berkaitan dengan penggalian *frequent closed itemsets*, belum ada algoritma yang mengintegrasikan penggunaan format data vertikal *diffset* dan pemeriksaan duplikasi tanpa melakukan penyimpanan semua *frequent closed itemsets* sebelumnya. Sehingga ada peluang penelitian untuk merancang perbaikan algoritma CHARM yang lebih efisien penggunaan memorinya.

Metodologi yang digunakan dalam penelitian ini berkaitan dengan perancangan perbaikan algoritma CHARM yang lebih efisien penggunaan memorinya selama proses enumerasi *frequent closed itemsets*. Metode yang digunakan adalah menggabungkan antara metode *subsumption check* pada cabang yang sedang dienumerasi dan metode *order preserving* dalam melakukan pemeriksaan duplikasi. Algoritma yang dirancang kemudian diimplementasikan dengan menggunakan bahasa pemrograman Visual C++ dan diuji dengan berbagai macam jenis data uji coba. Untuk mengukur keberhasilan rancangan algoritma, maka digunakan dua skenario uji coba seperti berikut: perbandingan efisiensi penggunaan memori dari algoritma yang dirancang terhadap algoritma CHARM dan perbandingan karakteristik waktu komputasi algoritma yang dirancang terhadap algoritma CHARM.

Hasil uji coba menunjukkan bahwa perbaikan algoritma CHARM yang telah dilakukan mampu meningkatkan efisiensi penggunaan memori dibandingkan dengan algoritma CHARM untuk nilai minimum *support* yang semakin kecil. Namun demikian, waktu komputasi dari algoritma yang telah diperbaiki tidak memberikan indikasi kuat untuk dikatakan lebih cepat dibandingkan dengan algoritma CHARM.

Kata kunci: *penggalian frequent closed itemset, perbaikan algoritma CHARM, format data vertikal Diffset*

IMPROVEMENT OF CHARM ALGORITHM FOR MINING FREQUENT CLOSED ITEMSETS

By : Mardiyanto
Student Identity Number : 5105 201 701
Supervisor : Prof . Dr. Ir. Arif Djunaidy, M.Sc.

ABSTRACT

Mining frequent closed itemsets is a fundamental part of mining association rule because it can uniquely determine the whole set of frequent itemsets and their support. Some algorithms of mining frequent closed itemsets have been discovered, some of them were CHARM and DCI_CLOSED algorithms. CHARM algorithm used a vertical diffset data format and a subsumption check method to perform duplication checking. The Subsumption check method is not efficient, however, because it needs to keep the whole frequent closed itemsets found previously in the main memory. DCI_CLOSED algorithm used a vertical bitvectors data format and an order preserving method to perform a duplication checking process. The order preserving method is efficient because it does not need to keep frequent closed itemsets as done in CHARM algorithm. Based on the literature survey related to algorithms for mining frequent closed itemsets, there was not algorithm that integrated the use of vertical diffset data format and duplication checking process to avoid the need for keeping the whole previously frequent closed itemsets found. So, there is an opportunity to design an improvement of CHARM algorithm in order to increase its efficiency in using the main memory.

The main idea of this research is to design an improved CHARM algorithm that is capable of reducing the use of main memory during the enumeration process of frequent closed itemsets generation. This is performed by combining the use a subsumption check method during the enumeration process of frequent closed itemsets and the use of an order preserving method during the duplication check of finding frequent closed itemsets. The improved CHARM algorithm is coded using Visual C++ programming language and tested using several types of experimental data. To evaluate the performance of the algorithm that has been implemented, two comparison experimental scenarios are applied. The first scenario is concerned with the comparison of the use of memory between the original and improved CHARM algorithms, while the second one is concerned with the comparison of running time consumed by both algorithms.

Experimental results show that the improved of CHARM algorithm has better memory utilization in compared to the original CHARM algorithm, particularly for low minimum support values. For comparison of running time, however, the improved algorithm does not give strong indication of whether it is better that that of the original algorithm.

Key words: mining frequent closed itemsets, improvement of CHARM algorithm, vertical diffset data format

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

BAB 1

PENDAHULUAN

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Penggalian *frequent closed itemsets* merupakan salah satu bagian penting dari proses penggalian kaidah asosiasi (*association rule*) karena dapat secara unik menentukan himpunan semua *frequent itemsets* dan *support*nya. Dalam penggalian kaidah asosiasi, *frequent closed sets* memiliki properti yang lebih lengkap dibandingkan dengan *maximal sets* yang tidak mempunyai *superset*.

Berbagai penelitian telah dilakukan untuk untuk melakukan penggalian *frequent itemset,* mulai dari yang dilakukan oleh R. Agrawal (1993) dengan algoritma apriori sampai dengan penelitian yang dilakukan oleh M. J. Zaki dengan algoritma CHARM (Zaki, 2005). CHARM menyimpan data pertama kali pada *disk*, tetapi seluruh data hasil antara disimpan dalam memori. Beberapa hal yang melatarbelakangi gagasan ini menjadi realistis adalah :

a. CHARM memecah ruang pencarian *IT-tree* menjadi potongan kecil yang independen satu sama lain dengan mendasarkan pada kesamaan *prefix* klas.
b. Penggunaan format data vertikal *Diffset* mempengaruhi secara ekstrim penggunaan memori yang kecil
c. CHARM menggunakan operasi interseksi himpunan sederhana dan tidak memerlukan struktur data internal yang rumit, dimana pembangkitan kandidat dan penghitungan terjadi dalam satu tahapan yang memerlukan memori dengan ukuran yang kecil.
d. Menggunakan 4 (empat) buah properti dan metode pengurutan 1-*itemset* yang didasarkan pada nilai *support,* sehingga dapat mengurangi tingkat enumerasi dan percabangan

Kombinasi beberapa faktor di atas membuat CHARM merupakan algoritma praktis dan efisien untuk memproses data yang besar. Namun demikian algoritma ini mempunyai dua kelemahan. Pertama, algoritma tidak dapat secara langsung melakukan penggalian *closed itemsets,* karena algoritma harus

melakukan komputasi yang redundan untuk menentukan apakah *frequent itemset* yang dihasilkan *closed* atau tidak. Kedua, algoritma tidak efisien karena menggunakan tabel *hash* untuk menyimpan semua *closed itemsets* yang dihasilkan sebelumnya untuk melakukan pemeriksaan duplikasi (Lucchesse, 2005).

Selanjutnya, Lucchesse dkk, (2005) melakukan penelitian untuk memperoleh algoritma enumerasi *frequent closed itemsets* yang efisien dan cepat, dengan fitur utama seperti berikut ini:

a. Algoritma menggunakan format data vertikal *bitvectors,* sehingga komputasi dapat dilakukan dengan operasi yang sederhana dan cepat (operator *bitwise AND)*
b. Algoritma dalam melakukan penggalian *closed itemsets* tidak harus sesuai dengan urutan leksikografiknya, sehingga tidak diperlukan lagi proses pengurutan *1-itemset frequent*.
c. Algoritma sangat efisien dalam penggunaan memori karena tidak melakukan penyimpanan *frequent closed itemsets* yang diperoleh pada tahap sebelumnya pada memori utama untuk melakukan pemeriksaan duplikasi

Algoritma tersebut di atas dinamakan DCI_CLOSED karena mewarisi algoritma yang diajukan sebelumnya untuk melakukan penggalian *frequent itemsets* (Lucchesse, 2002-2003). DCI_CLOSED mampu melakukan penghematan baik ruang penyimpan maupun waktu komputasi dalam melakukan penggalian *itemset closure* dan perhitungan nilai *support*-nya. Operasi dasar yang digunakan untuk melakukan penggalian *closure* adalah menghitung *support* dengan menggunakan operasi interseksi *bitwise AND tidset*.

Meskipun algoritma DCI_CLOSED mempunyai kelebihan dari segi kecepatan komputasi, tetapi algoritma tersebut memiliki kelemahan dalam hal penggunaan memori karena menggunakan format data vertikal *bitvectors*. Berdasarkan penelitian yang dilakukan oleh M. J. Zaki (Zaki, 2003), penggunaan format data vertikal *bitvectors* yang terkompresi hanya mempunyai rasio kompresi sebesar 2 sampai dengan 3 dibandingkan dengan penggunaan format data vertikal *tidset*. Di lain pihak, penggunaan penggunaan format data vertikal *diffset* dapat mempunyai rasio kompresi sebesar 2 sampai dengan 5 dibandingkan

penggunaan format data vertikal *tidset*. Dengan demikian, penggunaan format data vertikal *diffset* akan memerlukan memori yang lebih kecil dibandingkan dengan penggunaan format data vertikal *bitvectors*.

1.2 Perumusan Masalah

Berdasarkan pada latar belakang yang diuraikan sebelumnya maka masalah utama dari penelitian ini berkaitan dengan upaya untuk mengembangkan algoritma yang lebih efisien penggunaan memorinya dalam melakukan penggalian *frequent closed itemsets*. Algoritma yang dikembangkan mengintegrasikan penggunaan format data vertikal *diffset* dan pemeriksaan duplikasi *frequent closed itemsets* tanpa harus melakukan penyimpanan semua *frequent closed itemsets* yang dihasilkan sebelumnya. Pada penelitian ini tidak dilakukan perbaikan algoritma DCI_CLOSED karena :

- Kinerja algoritma DCI_CLOSED hanya dibandingkan dengan algoritma FP-CLOSE dan CLOSET+ (Lucchesse, 2005), sehingga tidak bisa diketahui perbandingan kinerja algoritma DCI_CLOSED dengan algoritma CHARM.
- Kinerja algoritma CHARM lebih baik dibandingkan dengan algoritma CLOSET+ (Zaki, 2005)

Oleh karena CHARM menggunakan format data vertikal *diffset* , memeriksa duplikasi dengan menyimpan seluruh *frequent closed itemsets* sebelumnya dalam struktur data tabel *hash* (Zaki, 2005), maka ada peluang penelitian untuk melakukan perbaikan terhadap algoritma CHARM. Sesuai dengan masalah penelitian, dalam rangka mendefinisikan lingkup penelitian yang spesifik, maka pertanyaan penelitian (*research question*) dari penelitian ini adalah "bagaimana mengembangkan algoritma penggalian *frequent closed itemsets* yang lebih efisien penggunaan memorinya dibandingkan dengan algoritma CHARM?"

1.3 Tujuan Penelitian

Penelitian ini bertujuan untuk mendesain algoritma penggalian *frequent closed itemsets* yang efisien.

1.4 Batasan Penelitian

Sesuai dengan perumusan masalah yang dijelaskan sebelumnya, penelitian ini dibatasi pada aspek efisiensi penggunaan memori yang digunakan dalam proses penggalian *frequent closed itemsets* dibandingkan dengan penggunaan memori yang diperlukan oleh algoritma CHARM.

1.5 Kontribusi dan Manfaat Penelitian

Kontribusi yang dapat diperoleh dari penelitian ini adalah tersedianya algoritma penggalian *frequent closed itemsets* yang lebih efisien penggunaan memorinya dibandingkan dengan algoritma CHARM. Sedangkan manfaat yang diharapkan dari penelitian ini terkait dengan penerapan algoritma penggalian *frequent closed itemsets* dalam proses penggalian kaidah asosiasi (*association rule*) yang lebih efisien dalam menggunakan memori.

1.6 Sistematika Penulisan Tesis

Penulisan tesis disusun berdasarkan kerangka sebagai berikut :

a. Bab 1 mengenai pendahuluan yang membahas tentang rencana pengerjaan tesis yang meliputi latar belakang, tujuan penelitian, perumusan masalah, asumsi masalah, dan sistimatika penulisan.
b. Bab 2 membahas dasar teori yang menunjang penelitian.
c. Bab 3 membahas tentang desain algoritma dan implementasi aplikasi, tentang bagaimana cara mengimplementasikan desain algoritma yang telah dibangun menjadi sebuah aplikasi.
d. Bab 4 membahas tentang uji coba dan analisis hasil.
e. Bab 5 tentang penutup yang berisi simpulan yang dapat diambil dan saran yang dapat diberikan untuk pengembangan lebih lanjut.

BAB 2

TINJAUAN PUSTAKA

BAB 2
TINJAUAN PUSTAKA

Untuk mempermudah pemahaman tentang algoritma CHARM dan algoritm DCI_CLOSED maka penulisan dilakukan dengan kerangka seperti tersebut di bawah ini :

1. Penggalian *frequent closed itemsets*
2. *Itemset-Tidset Tree* (*IT-tree*) dan kesamaan klas
3. Properti dasar pasangan *itemset-tidset*
4. *Diffset* untuk perhitungan frekuensi cepat
5. Format data vertikal
6. Algoritma CHARM
7. Algoritma DCI_CLOSED

2.1 Penggalian *Frequent Closed Itemsets*

Jika χ dimisalkan sebagai himpunan item dari sebuah transaksi basis data, dimana setiap transaksi terdiri dari sebuah atribut yang menyatakan identifikasi transaksi yang unik (*tid*) dan satu set item. Himpunan semua *tids* yang ada dalam basis data dinyatakan sebagai τ, maka satu set $X \subseteq \chi$ didefinisikan sebagai *itemset* dan sebuah himpunan $Y \subseteq \tau$ didefinisikan sebagai *tidset*. Sebuah *itemset* yang terdiri dari k buah item didefinisikan sebagai k-*itemset*, sedang sebuah himpunan yang terdiri dari satu set *tid* didefinisikan sebagai *tidset*. Untuk penyingkatan penulisan, sebuah item dinotasikan berupa sebuah huruf dan sebuh *tid* dinotasikan berupa sebuah angka. Sebuah k-*itemset* dinotasikan berupa sebuah string yang terdiri dari k huruf dari masing-masing item, sebuah *tidset* dinotasikan berupa sebuah string yang menggabungkan nomor-nomor identifikasi dari transaksi. Sebagai contoh 3-*itemset* {A,C,W} dan *tidset* {2,4,5} berturut-turut dituliskan menjadi ACW dan 245.

Jika t(X) menyatakan himpunan *tidset* dari sebuah *itemset* X, dan i(Y) menyatakan himpunan *itemset* dari sebuah transaksi Y, maka elemen dari t(X) dapat diperoleh dengan menginterseksikan semua himpunan *tidset* dari masing-masing item x ∈ X, yaitu $t(X) = \bigcap_{x \in X} t(x)$; dan elemen dari i(Y) dapat diperoleh dengan menginterseksikan semua himpunan *itemset* dari masing-masing *tid* y ∈ Y , yaitu $i(Y) = \bigcap_{y \in Y} i(y)$. Notasi $t(X) = \bigcap_{x \in X} t(x)$ $i(Y) = \bigcap_{y \in Y} i(y)$ menunjukkan pasangan *itemset-tidset*, atau yang lebih dikenal dengan sebutan *IT-pair*. Untuk memberi gambaran nyata bagaimana menghitung t(X) dan i(Y) maka dipergunakan basis data contoh sebagaimana ditunjukkan pada Tabel 2.1.

Tabel 2.1 Basis Data Contoh

TID	items				
1	A	C	T	W	
2	C	D	W		
3	A	C	T	W	
4	A	C	D	W	
5	A	C	D	T	W
6	C	D	T		

Seperti ditunjukkan pada Tabel 2.1, maka dapat diperoleh t(A) = 1345, t(C)= 123456, t(W)=1245, i(1) = ACTW dan i(2) = CDW, sehingga untuk menghitung t(ACW) dan i(12) dapat dilakukan dengan cara seperti dibawah ini :

$$t(ACW) = t(A) \cap t(C) \cap t(W) = 1345 \cap 123456 \cap 12345 = 1345$$

$$i(12) = i(1) \cap i(2) = ACTW \cap CDW = CW.$$

Support menyatakan jumlah transaksi munculnya *itemset* X dalam suatu basis data (Zaki, 2005), dinyatakan dengan notasi $\sigma(X) = |t(X)|$. Sebuah *itemset* disebut *frequent* jika *support*nya lebih besar atau sama dengan nilai minimum *support(minsup)* atau dapat dinyatakan dengan $\sigma(X) \geq minsup$. *Frequent itemset* disebut *maximal* jika tidak ada satupun *superset itemset* terdekatnya yang *frequent*. Jika $c:P(\chi) \rightarrow P(\chi)$ merupakan operator *closure*, maka dapat didefinisikan $c(X) = i(t(X))$, dimana $X \subseteq \chi$. Sebuah *frequent itemset* X dinamakan *closed* jika dan hanya jika c(X) = X (Zaki, 2005). Dengan kata lain,

frequent itemset X dinamakan *closed* jika tidak ada satupun *superset* $Y \supset X$ dengan $\sigma(X) = \sigma(Y)$. Untuk basis data contoh seperti ditunjukkan pada Tabel 2.1, maka untuk menghitung *closure* AW adalah :

$$c(AW) = i(t(AW)) = i(1345) = ACW$$

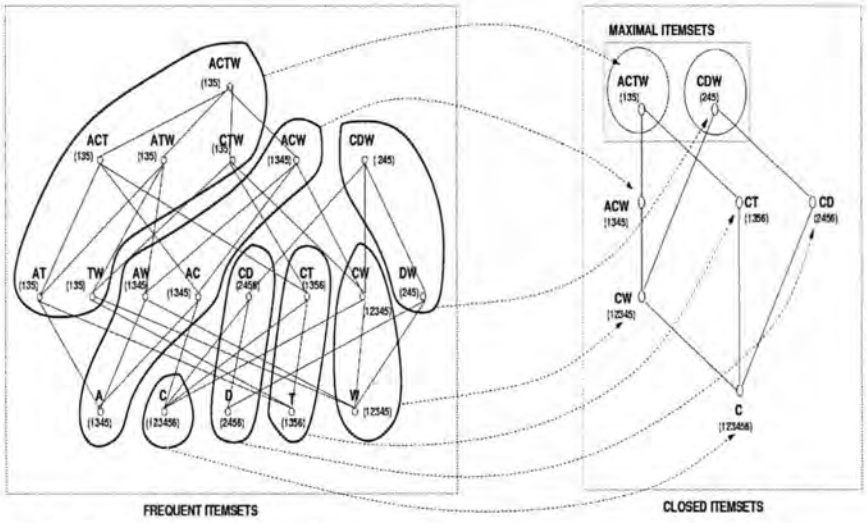
karena dari hasil perhitungan menunjukkan c(AW) = ACW maka AW tidak *closed.* Sedangkan *closure* ACW closed karena c(ACW) = i(t(ACW))= i(1345)=ACW.

Tabel 2.2 Frequent Itemsets

Support	*Itemset*							
100% (6)	C							
83% (5)	W	CW						
67% (4)	A	D	T	AC	AW	CD	CT	ACW
50% (3)	AT	DW	TW	ACT	ATW	CDW	CTW	ACTW

Sebagaimana ditunjukkan pada Tabel 2.1 ada 5 item unik yang digunakan, X = {A,C,D,T,W} dan 6 buah transaksi, χ={1,2,3,4,5,6}. Tabel 2.2 menunjukkan hasil enumerasi seluruh *frequent itemsets* dengan *minsup* = 50%. Terdapat 19 *frequent itemsets* yang berisi paling sedikit tiga transaksi dengan *minsup* = 50%. Dari 19 *frequent itemsets*, hanya ada 7 *closed itemsets dan 2* maksimal *frequent itemsets*, diperoleh dengan cara menghilangkan semua *itemset* yang mempunyai *tidset* sama. Gambar 2.2 menunjukkan *lattice* dari *frequent, closed* dan maksimal *itemset.*

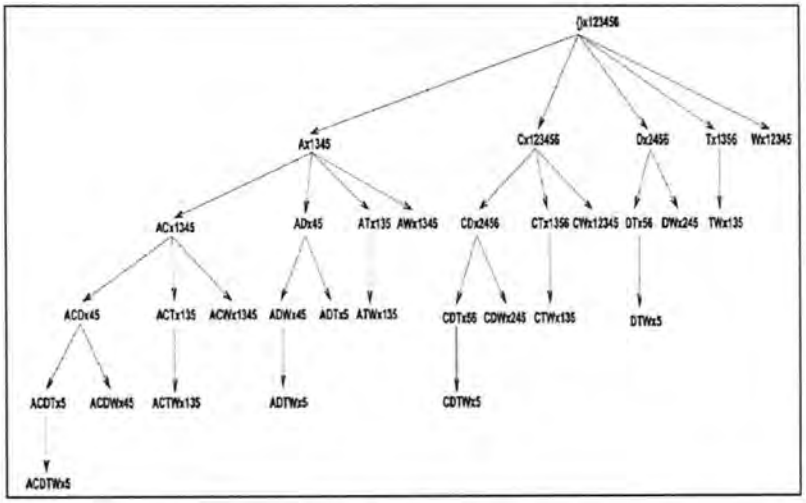
Jika F menunjukkan himpunan *frequent itemsets*, *C* himpunan *closed itemset* dan M himpunan *maximal itemset*, maka hubungan antara M, C dan F dapat dinyatakan dengan $M \subseteq C \subseteq F$ seperti ditunjukkan pada Gambar 2.1 Secara teoritis pada kondisi terburuk, ada $2^{|x|}$ *frequent itemsets* dan *frequent closed itemsets* dengan *x* adalah 1-*itemset frequent.*



Gambar 2.1 Frequent, Closed, Maximal Itemset

2.2 *Itemset-Tidset Tree* (*IT-tree*) dan Kesamaan Klas

Jika X merupakan himpunan *items*, dan p menyatakan *prefix*, maka dapat didefinisikan fungsi p(X,k) = X[1:k] dengan k panjang *prefix* dari X. Kesamaan relasi θ_k *itemsets* pada sebuah *prefix* dapat dinyatakan sebagai berikut : $\forall X, Y \subseteq \chi$, $X=\theta_k Y \Leftrightarrow p(X,k) = p(Y,k)$, dua *itemsets* dalam klas yang sama jika keduanya membagi *prefix* dengan panjang k. CHARM melakukan pencarian himpunan *frequent closed* dengan ruang pencarian *IT-tree* baru seperti ditunjukkan dalam Gambar 2.2.



Gambar 2.2 IT-Tree: Itemset-Tidset Search Tree

Setiap simpul dalam *IT-tree* disajikan dengan pasangan *itemset-tidset*, X x t(X). Semua simpul *children* X masuk ke dalam klas yang sama karena membagi *prefix* X. Jika P menyatakan simpul *parent (prefix)* dan tiap l_i menyatakan item tunggal, maka kesamaan klas [P] = $\{l_1,l_2,\ldots,l_n\}$ dapat menyajikan simpul Pl_i x t(Pl_i). Sebagai contoh seperti ditunjukkan pada Gambar 2.2, *root tree* dengan kesamaan klas []={A,C,D,T,W}, *child* yang paling kiri dari *root* adalah klas [A] menjadi sebuah *prefx* dengan himpunan {C,D,T,W}. Tiap anggota klas menyajikan satu *child* dari simpul *parent*. Sebuah klas menyajikan *items* yang *prefix*nya dapat diperluas untuk mendapatkan simpul *frequent* baru. Oleh karena itu tidak ada *subtree prefix infrequent* yang harus diperiksa. Kekuatan pendekatan kesamaan klas adalah memecah ruang pencarian asli menjadi sub permasalahan yang tidak saling berhubungan. Untuk banyak *subtree root* pada simpul X, dapat diperlakukan sebagai problem baru yang lengkap. *Subtree* dapat menampilkan satu persatu pola dibawahnya dan menyederhanakan *prefix*nya dengan *item* X dan sebagainya.

Setiap klas *prefix* dapat melakukan interseksi *tidset* semua pasangan elemen dalam klas dan meneliti *minsup*nya. Hasil *frequent itemset*nya adalah klas dirinya sendiri, dengan elemen yang dapat diperluas secara rekursif. Klas *itemset* dengan *prefix* P, [P] = $\{l_1,l_2,\ldots,l_n\}$, dapat melakukan interseksi t(l_i) dengan semua t(l_j) dengan j > i untuk memperoleh perluasan klas *frequent* baru, [Pl_i] = $\{l_j \mid j > i$ dan $\sigma(Pl_il_j) \geq$ *minsup*. Sebagaimana ditunjukkan pada Gambar 2.2, *root null* []={A,C,D,T,W} dengan *minsup*=50%, diperoleh *klas* [A] = {C,T,W}, klas [C]={D,T,W}, klas[D]={W} dan klas [W]={}. Klas [A] tidak berisi D karena AD tidak frequent. Algoritma 2.1, memberikan deskripsi *pseudocode* eksplorasi *depth first* (DFS) pada *IT-tree* untuk semua pola *frequent*. CHARM memperbaiki skema enumerasi dasar dengan menggunakan konsep *framework* yang dinamakan *IT-tree*, sehingga dapat menghemat penggunaan memori selama proses enumerasi *frequent itemsets*.

Algoritma 2.1

Enumerate_Frequent([P]):

```
1.  for all li ∈ [P] do
2.    [Pi] =0;
3.    for all j ∈ [P], with j > i do
4.      I = Pli lj;
5.      T = t(Pli) ∩ t(Plj);
6.      if |T| ≥ minsup then
7.        [Pi] = [Pi] ∪ { I x T }
8.      end if
9.    if (DFS) then Enumerate_Frequent ([Pi]); delete [Pi];
10. if (BFS) then
11.   for all [Pi] do Enumerate_Frequent ([Pi]);delete [Pi];
```

2.3 Properti Dasar Pasangan *Itemset-Tidset*

Dua simpul *IT-tree* dinyatakan dengan X_i x $t(X_i)$ dan X_j x $t(X_j)$. Jika didefinisikan $f : P(\chi) \rightarrow N$ merupakan pemetaan satu satu dari *itemsets* integer, maka untuk sebarang *itemsets* X_i dan X_j, dikatakan $X_i \leq_f X_j$ jika $f(X_i) \leq f(X_j)$. Fungsi f menyatakan urutan pada seluruh himpunan *itemsets*. Sebagai contoh jika *f* menunjukkan urutan *lexicographic*, maka *itemset* AC ≤ AD, tetapi jika *f* merupakan urutan *support*nya, maka *itemset* AD ≤ AC jika σ(AD) ≤ σ(AC). CHARM mempunyai 4 (empat) properti dasar *IT-pair* sebagaiman terlihat pada pada Gambar 2.5. Dengan empat properti dasar ini algoritma CHARM dapat melakukan enumerasi himpunan *closed* secara cepat. Sebuah klas [P] dengan anggota $\{l_1,l_2,\ldots,l_n\}$ adalah *prefix* klas P, jika X_i menyatakan *itemset* Pl_i dan t(Xi) menyatakan *tid itemset* X_i maka setiap anggota [P] adalah *IT-pair* X_i x $t(X_i)$.

Jika X_i x $t(X_i)$ dan X_j x $t(X_j)$, menjadi sebarang dua anggota *klas* [P], maka empat properti berikut diperoleh :

Properti 1 Jika $t(X_i)=t(X_j)$, sehingga $c(X_i)=c(X_j)=c(X_i \cup X_j)$

Properti 2 Jika $t(X_i) \subset t(X_j)$, sehingga $c(X_i) \neq c(X_j)$, tetapi $c(X_i)=c(X_i \cup X_j)$

Properti 3 Jika $t(X_i) \supset t(X_j)$, sehingga $c(X_i) \neq c(X_j)$, tetapi $c(X_j)=c(X_i \cup X_j)$

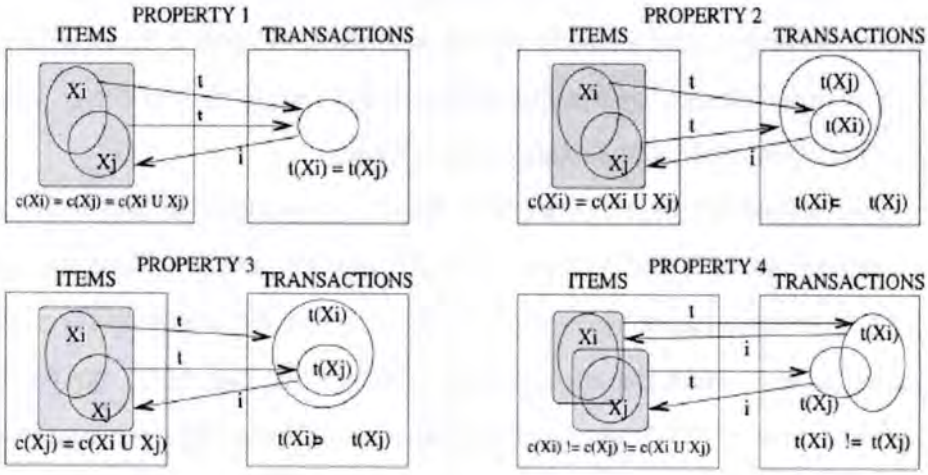
Properti 4 Jika $t(X_i) \neq t(X_j)$, sehingga $c(X_i) \neq c(X_j) \neq c(X_i \cup X_j)$

Pembuktian :

1. Jika $t(X_i) = t(X_j)$ maka $i(t(X_i))=i(t(X_j)$ atau dapat dikatakan $c(X_i) = c(X_j)$. Karena $t(X_i) = t(X_j)$ maka $t(X_i \cup X_j) = t(X_i) \cap t(X_j) = t(X_i)$, sehingga :

   $$i(t(X_i \cup X_j)) = i(t(X_i))$$

   akibatnya $c(X_i \cup X_j) = c(X_i)$. Properti ini menunjukkan bahwa dapat digantikan setiap kehadiran X_i dengan $X_i \cup X_j$ dan dapat dihapus elemen X_j dalam proses enumerasi selanjutnya karena *closure*nya sama dengan *closure* $X_i \cup X_j$
2. Jika $t(X_i) \subset t(X_j)$ maka $t(X_i \cup X_j) = t(X_i) \cap t(X_j) = t(X_i) \neq t(X_j)$ dan *closure*nya adalah $c(X_i \cup X_j) = c(X_i) \neq c(X_j)$. Properti 2 ini menunjukkan bahwa dapat digantikan setiap kehadiran X_i dengan $X_i \cup X_j$ karena keduanya mempunyai *closure* yang sama. Tetapi karena $c(X_i) \neq c(X_j)$ yang berarti membangkitkan *closure* yang berbeda sehingga tidak dapat dihilangkan elemen X_j dari klas [P]
3. Sama dengan kasus no 2 diatas.
4. Jika $t(X_i) \neq t(X_j)$ maka $t(X_i \cup X_j) = t(X_i) \cap t(X_j) \neq t(X_i) \neq t(X_j)$, menghasilkan $c(X_i \cup X_j) \neq c(X_i) \neq c(X_j)$. Tidak ada elemen klas yang dihapus, kedua X_i dan X_j mempengaruhi *closure* berbeda (begitu juga *subset* lainnya)



Gambar 2.3 Basic Properti : *Itemset* dan *tidset*

2.4 *Diffset* untuk Perhitungan Frekuensi Cepat

Algoritma CHARM menggunakan format data vertikal. Kelebihan utama penggunaan format data vertikal dibandingkan penggunaan format data horisontal adalah:

a. *Support* dapat dihitung lebih mudah dan lebih cepat dengan melakukan interseksi pada *tidset* yang diperlukan, sebaliknya penggunaan format data horisontal memerlukan *hash-tree* yang rumit.
b. Ada pemangkasan otomatis informasi yang tidak relevan ketika proses interseksi.

Penggunaan format data vertikal dengan kardinalitas *tidset* sangat panjang memerlukan waktu interkseksi awal dan memori antara *tidset* pola *frequent* yang dibangkitkan menjadi sangat besar. Oleh karena itu diperlukan kompresi dan penyimpanan hasil sementara pada *disk*. Khusus pada *dataset* yang rapat, penggunaan format data vertikal *tidset* akan kehilangan kelebihannya. Untuk itu disajikan format data vertikal baru yang dinamakan *diffset* (Zaki, 2003). *Diffset* menjaga track perbedaan *tid* pola kandidat dari pola *frequent parent*-nya. Perbedaan ini dipropagasi dari satu simpul *root* ke seluruh *children*-nya. Hasil penelitian menunjukkan bahwa *diffset* secara drastis mengurangi ukuran memori yang diperlukan untuk menyimpan hasil sementara (Zaki, 2003). Bahkan dalam domain yang rapat, beberapa algoritma penggalian *frequent closed itemsets* yang menggunakan format data vertikal *diffset*, seluruh pola *set*nya dapat seluruhnya berada di memori utama. *Diffset* merupakan bagian kecil ukuran *tidset* sehingga interseksi *tid* dapat dilakukan dengan sangat effisien.

Jika sebuah klas dengan *prefix* P dan d(X) menunjukkan *diffset* X, maka dalam keadaan normal ketersediaan klas P dengan *prefix* t(P) sama dengan ketersediaan semua anggota klas $t(PX_i)$. Jika PX dan PY menyatakan sebarang dua anggota klas P, maka dapat dinyatakan $t(PX) \subseteq t(P)$ dan $t(PY) \subseteq t(P)$. Untuk memperoleh *support* PXY dapat dilakukan dengan meneliti kardinalitas dari $t(PX) \cap t(PY) = t(PXY)$. Apabila yang tersedia bukan t(PX) tetapi $d(PX) = t(P) - t(X)$ yang menyatakan perbedaan *tids* X dari P, dan $d(PY) = t(P) - t(Y)$ yang

menyatakan perbedaan *tids* Y dari P, maka untuk menghitung *support* PX dapat dihitung dengan cara berikut ini :

$$\sigma(PX) = \sigma(P) - |d(PX)|.$$

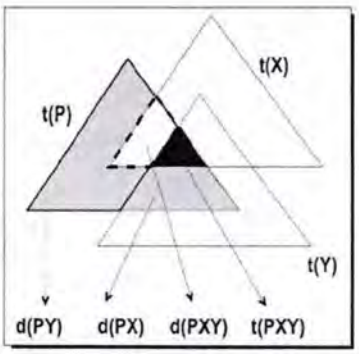
Demikian juga apabila yang tersedia adalah d(PX) dan d(PY) secara rekursif σ(PXY) dapat dihitung sebagai berikut

$$\sigma(PXY) = \sigma(PX) - |d(PXY)|.$$

Dengan hanya menggunakan *diffset,* d(PXY) dapat dihitung dengan cara berikut ini :

$$d(PXY) = t(PX) - t(PY) = t(PX) - t(PY) + t(P) - t(P)$$
$$= (t(P) - t(PY)) - (t(P) - t(PX))$$
$$= d(PY) - d(PX)$$

dengan kata lain, perhitungan d(XY) dapat dihitung dengan menggunakan *tidset* atau *diffset*, apabila yang tersedia adalah *tidset* maka perhitungannya adalah t(PX) – t(PY), sedangkan apabila yang tersedia *diffset* maka perhitungannya menjadi d(PY) – d(PX). Gambar 2.4 menunjukkan perbedaan daerah untuk *tidset* dan *diffset* klas *prefix* yang diberikan. *Tidset* P ditunjukkan dengan segitiga yang ditandai t(P), daerah abu-abu menunjukkan d(PX), sementara daerah dengan garis hitam solid menunjukkan d(PY) merupakan subset dari *tidset prefix* baru PX.



Gambar 2.4 *Diffset Prefix* P

Jika $m(X_i)$ dan $m(X_j)$ masing-masing menunjukkan jumlah ketidaksesuaian *diffset* $d(X_i)$ dan $d(X_j)$, maka ada 4 properti yang harus dipertimbangkan :

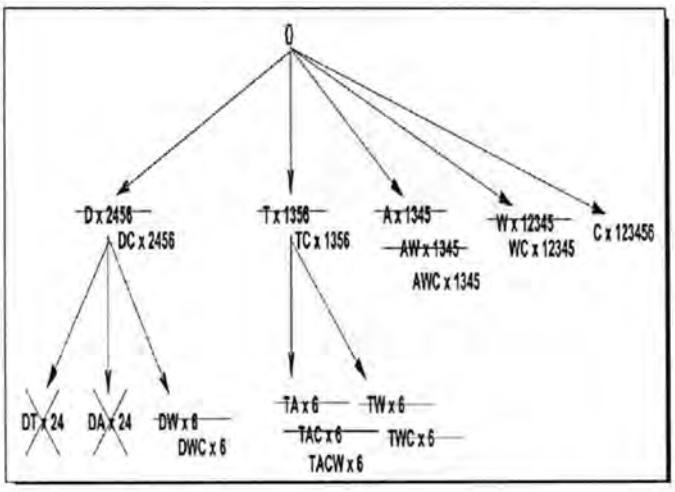
a. Properti 1. $m(X_i) = 0$ dan $m(X_j) = 0$, lalu $d(X_i) = d(X_j)$ atau $t(X_i) = t(X_j)$

b. Properti 2, $m(X_i) > 0$ dan $m(X_j) = 0$, lalu $d(X_i) \supset d(X_j)$ atau $t(X_i) \subset t(X_j)$

c. Properti 3. $m(X_i) = 0$ dan $m(X_j) > 0$, lalu $d(X_i) \subset d(X_j)$ atau $t(X_i) \supset t(X_j)$

d. Properti 4. $m(X_i) > 0$ dan $m(X_j) > 0$, lalu $d(X_i) \neq d(X_j)$ atau $t(X_i) \neq t(X_j)$

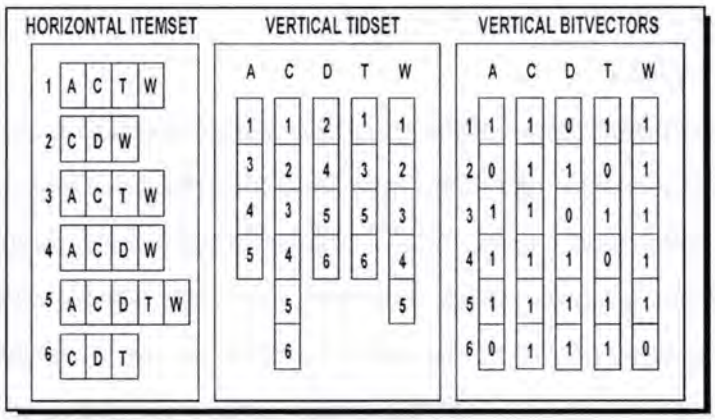
Apabila dianggap basis data awal disimpan dengan format data vertikal *tidset*, dan proses selanjutnya digunakan format data vertikal *diffset*, algoritma CHARM melakukan pengujian *support*, *subset,* kesamaan dan ketidaksamaan secara bersama-sama pada saat menghitung perbedaanya sendiri. Gambar 2.5 menunjukkan pencarian *closed set* menggunakan *diffset* pengganti *tidset*. Penggalian *frequent closed itemsets* persis sama seandainya menggunakan format data vertikal *tidset*, klas *root* tetap menggunakan *tidset* sedangkan untuk klas lainnya menggunakan *diffset*. Seperti contoh yang ditunjukkan pada Gambar 2.5 sebuah *IT-pair* TAWC x 6 menyatakan bahwa perbedaan *tid* TAWC terhadap *parent*-nya TC x 1356 hanya pada *tid* 6, sehingga dapat disimpulkan bahwa *IT-pair* sesungguhnya adalah TAWC x 135



Gambar 2.5 Proses Pencarian Menggunakan *Diffset*

2.5 Format Data Vertikal

Gambar 2.6 menunjukkan beberapa format data yang digunakan dalam penggalian *frequent closed itemsets*. Pada pendekatan horisontal tradisional, setiap transaksi mempunyai sebuah *tid* yang bersesuain. Sebaliknya format vertikal memelihara setiap item *tidset*-nya. Apriori, MaxMiner dan DepthProject merupakan beberapa algoritma yang menggunakan format data horisontal, algoritma CHARM menggunakan format data vertikal *diffset* (Zaki, 2005), ECLAT, PARTITION, VIPER dan MAFIA menggunakan format data vertikal *bitvectors* terkompresi sebagai pengganti *tidset*.



Gambar 2.6 Format Data

Penggunaan format data vertikal lebih efisien dibandingkan penggunaan format data horisontal karena pola *frequent itemsets* dapat dihitung secara langsung dengan interseksi *tidset*. Pembangkitan kandidat dan penghitungan *support* dengan menggunakan format data vertikal dapat dilakukan dalam satu tahap, sebaliknya penggunaan format data horisontal memerlukan *hash/search tree* yang rumit.

Secara umum terdapat dua format data vertikal yaitu :

a. Vertikal *Diffset/Tidset*

Format data vertikal dengan menggunakan *tidset* memiliki kelemahan apabila kardinalitasnya sangat besar. Penggunaan format data ini menjadi tidak efisien karena berawal dengan waktu interseksi yang sangat besar, selain itu ukuran memori pembangkitan *tidset* antara untuk pola *frequent* dapat juga menjadi sangat besar, sehingga memerlukan kompresi data dan penyimpanan hasil sementara di *disk*. Khusus untuk *dataset* yang rapat dengan karakteristik

frequent item yang tinggi dan mempunyai pola yang sangat banyak, penggunaan format data vertikal *tidset* kehilangan kelebihannya.

Format data vertikal *diffset* hanya menyimpan perbedaan *tid* dari setiap pola kandidat yang berasal dari pola *frequent* pembangkitan. *Diffset* dapat mengurangi ukuran memori yang diperlukan untuk menyimpan hasil antara secara drastis. Basis data yang disimpan dalam format *diffset* dapat juga mengurangi ukuran basis data keseluruhan. Sehingga pada ruang lingkup data yang rapat, seluruh pola yang sedang dibangkitkan dapat seluruhnya berada di memori utama. Karena *diffset* merupakan bagian kecil dari ukuran *tidset*, operasi interseksi dilakukan lebih cepat.

b. Vertikal *bitvectors*

Penggunaan format data vertikal *bitvectors* memerlukan satu bit untuk setiap transaksi di basis data. Jika item i muncul di transaksi ke-j, maka bit j dari dari *bitvectors* untuk item i diberi nilai 1, jika sebaliknya diberi nilai 0. Sehingga setiap item menyimpan semua informasi *tidset*-nya. Rasio kompresi format data vertikal *bitvectors* hanya berkisar 2 sampai dengan 3, sedangkan apabila menggunakan format data vertikal *diffset* dapat mempunyai rasio kompresi sebesar 2 sampai dengan 5 (Zaki, 2003). Interseksi dengan menggunakan format data vertikal *bitvectors* dapat dilakukan dengan cepat dengan melakukan operasi *bitwise AND*.

2.6 Algoritma CHARM

CHARM merupakan algoritma yang efisien untuk mengenumerasi himpunan semua *frequent closed itemsets* yang dapat secara langsung membentuk struktur data *lattice*. Beberapa ide inovatif diterapkan pada pengembangan CHARM, yaitu :

a. Secara bersama-sama memeriksa baik ruang *itemset* maupun ruang transaksi yang disebut dengan ruang pencarian *IT-tree*(*itemset tidset tree*) sebagaimana dijelaskan dalam Sub Bab 2.2

b. Menggunakan metode pencarian yang sangat efisien karena banyak tingkat *IT-tree* yang dilewati sehingga secara cepat dapat mengidentifikasi *frequent closed itemsets*.

c. Menggunakan pendekatan *hash-based* cepat dan pendekatan *intersection-based* untuk menghapus *non closed itemsets* selama pemeriksaan *subsumption*.

d. Menggunakan format data vertikal *diffset* (Zaki, 2003), sehingga secara drastis dapat mengurangi jumlah memori yang diperlukan untuk menyimpan hasil antara.

CHARM menggunakan *IT-tree* sebagai metode pencarian baru. Properti yang dimiliki pada *IT-pair* dapat secara cepat mengumpulkan *itemset closure*, karena tidak harus melakukan enumerasi banyak *subset* yang mungkin.

```
CHARM (D, min_sup):
1.  [∅] = {l_i × t(l_i) : l_i ∈ I ∧ σ(l_i) ≥ min_sup}
2.  CHARM-EXTEND ([∅], C = ∅)
3.  return C //all closed sets

CHARM-EXTEND ([P], C):
4.  for each l_i × t(l_i) in [P]
5.     P_i = P ∪ l_i and [P_i] = ∅
6.     for each l_j × t(l_j) in [P], with j > i
7.        X = l_j and Y = t(l_i) ∩ t(l_j)
8.        CHARM-PROPERTY(X × Y, l_i, l_j, P_i, [P_i], [P])
9.     SUBSUMPTION-CHECK(C, P_i)
10.    CHARM-EXTEND ([P_i], C)
11.    delete [P_i]

CHARM-PROPERTY (X × Y, l_i, l_j, P_i, [P_i], [P]):
12. if (σ(X) ≥ minsup) then
13.    if t(l_i) = t(l_j) then //Property 1
14.       Remove l_j from [P]
15.       P_i = P_i ∪ l_j
16.    else if t(X_i) ⊂ t(X_j) then //Property 2
17.       P_i = P_i ∪ l_j
18.    else if t(X_i) ⊃ t(X_j) then //Property 3
19.       Remove l_j from [P]
20.       Add X × Y to [P_i] //use ordering f
21.    else if t(X_i) ≠ t(X_j) then //Property 4
22.       Add X × Y to [P_i] //use ordering f
```

Gambar 2.7 Algoritma CHARM

Pseudocode algoritma CHARM seperti ditunjukkan pada Gambar 2.8, dapat dijelaskan seperti berikut :

a. Inisialisasi klas [P], simpul diperiksa untuk 1-*itemset frequent* dan *tidset*-nya (l_i x t(l_1), $l_1 \in \chi$) pada baris 1. Elemen dalam [P] diurut sesuai dengan total orde f berdasarkan *support*nya. Perhitungan utama dilakukan pada prosedur CHARM-EXTEND, dimana pada prosedur tersebut dihasilkan himpunan *frequent closed itemsets C*.

b. Prosedur CHARM-EXTEND bertanggung jawab untuk mempertimbangkan kombinasi *IT-pair* l_i x t(l_1) yang dilakukan di baris 6 diurut sesuai dengan

jumlah orde f. Setiap l_i membangkitkan sebuah *prefix* baru, $P_i = P \cup l_i$, dengan klas $[P_i]$, yang awalnya kosong (baris 5).

c. Pada baris 7, dua *IT-pair* dikombinasikan menghasilkan *pair* baru X x Y, dimana $X = l_j$ dan $Y = t(l_i) \cap t(l_j)$.

d. Baris 8 menguji empat properti *IT-pair* yang digunakan dengan memanggil prosedur CHARM-PROPERTY. Sebagai catatan, pada rutin ini memungkinkan terjadinya perubahan klas [P] saat itu dengan menghapus *IT-pair* yang telah di *subsumed* oleh *pair* lainnya., menambah *IT-pair* terbaru yang dibangkitkan pada klas $[P_i]$ baru dan dapat juga mengubah *prefix* P_i pada Properti 1 dan 2.

e. Lalu menambah *itemset* Pi dalam himpunan *closed itemsets* C (baris 9), menunjukkan bahwa Pi bukan *subsumed* oleh *closed set* yang ditemukan sebelumnya.

f. Apabila semua l_j telah diproses, secara rekursif dilakukan eksplorasi klas $[P_i]$ baru dengan DFS (baris 10). Setelah semua *closed itemset* yang berisi P_i telah dibangkitkan, dilakukan proses berikutnya (*unpruned*) *IT-pair* pada [P} (baris 11).

Pengurutan elemen *itemset* didasarkan pada *support*-nya. Tujuannya adalah untuk meningkatkan peluang elemen yang dihapus dari klas [P], sehingga peluang terjadinya properti 1 dan 2 lebih besar dibandingkan dengan dua properti lainnya. Properti 1, *closure* dari dua *itemset*nya sama sehingga dapat dihapus l_j dari [P] dan mengganti P_i dengan $P_i \cup l_j$, sedangkan properti 2, masih dapat diganti P_i dengan $P_i \cup l_j$. Kedua properti ini tidak menambah klas [Pi] baru, sehingga apabila kejadian 1 dan 2 lebih banyak terjadi maka lebih sedikit tingkat pencarian dilakukan. Apabila lebih banyak kejadian muncul properti 3 dan 4 menyebabkan penambahan himpunan simpul baru sehinga memerlukan penambahan tingkat pemrosesan. Karena diinginkan $t(l_i) = t(l_j)$ (Properti 1) atau $t(l_i) \subset t(l_j)$ (Properti 2), maka 1-*itemset frequent* diurut sesuai dengan urutan naik *support*nya. Pada tingkat *root IT-tree*, CHARM menggunakan urutan simpul berdasarkan bobot *support*nya. Jika $x,y \in \chi$ menggambarkan bobot sebuah item x maka dapat dinyatakan $w(x) = \sum_{xy \in F2} \sigma(xy)$, merupakan jumlah *support frequent*

2 *itemset* yang berisi item x pada tingkat *root*. Untuk tingkat selanjutnya, elemen ditambahkan dengan urutan *suppor*nya untuk tiap klas [P_i] baru (baris 20 dan 22) sehingga pengurutan kembali diterapkan secara rekursif pada setiap simpul *tree*.

2.7 Algoritma DCI_CLOSED

Algortima DCI_CLOSED menggunakan format data vertikal *bitvectors* dan pemeriksaan duplikasi dilakukan dengan metode *order preserving*. Algoritma ini efisien dalam hal pemeriksaan duplikasi karena algoritma tidak perlu menyimpan seluruh *frequent closed itemsets* hasil proses enumerasi sebelumnya. Dengan metode ini proses penggalian *frequent closed itemsets* dimungkinkan untuk dilakukan secara paralel. *Pseudocode* Algoritma DCI_CLOSED$_d$ seperti ditunjukkan pada Gambar 2.8

Algorithm 1 DCI_CLOSED pseudocode

```
 1: procedure DCI_CLOSED_d (CLOSED_SET, PRE_SET, POST_SET)
 2:   while POST_SET ≠ ∅ do
 3:     i ← min_≺(POST_SET)
 4:     POST_SET ← POST_SET \ i
 5:     new_gen ← CLOSED_SET ∪ i                ▷ Build a new generator
 6:     if supp(new_gen) ≥ min_supp and
  :       ¬is_dup(new_gen, PRE_SET) then
  :                  ▷ if new_gen is both frequent and order preserving
 7:       CLOSED_SET_New ← new_gen
 8:       POST_SET_New ← ∅
 9:       for all j ∈ POST_SET do        ▷ Compute closure of new_gen
10:         if g(new_gen) ⊆ g(j) then
11:           CLOSED_SET_New ← CLOSED_SET_New ∪ j
12:         else
13:           POST_SET_New ← POST_SET_New ∪ j
14:         end if
15:       end for
16:       Write Out CLOSED_SET_New and its support
17:       DCI_CLOSED_d(CLOSED_SET_New, PRE_SET, POST_SET_New)
18:       PRE_SET ← PRE_SET ∪ i
19:     end if
20:   end while
21: end procedure
  :
  :
22: function is_dup(new_gen, PRE_SET)              ▷ Duplicate check
23:   for all j ∈PRE_SET do
24:     if g(new_gen) ⊆ g(j) then
25:       return TRUE                  ▷ new_gen is not order preserving
26:     end if
27:   end for
28:   return FALSE
29: end function
```

Gambar 2.8 Algoritma DCI_CLOSED$_d$

Jika T menyatakan himpunan transaksi yang ada dalam basis data D dan I menyatakan *itemset* maka $T \subseteq D$ dan $I \subseteq \chi$, menjadi *subset* semua transaksi dan item-item yang muncul di D. Konsep *closed itemset* didasarkan pada dua fungsi *f* dan *g*

$$f(T) = \{ i \in \chi \mid \forall t \in T, i \in t \}$$

$$g(I) = \{ t \in \chi \mid \forall i \in I, i \in t \}$$

fungsi *f* menyatakan himpunan item-item yang berada di semua transaksi T, sementara fungsi *g* menyatakan himpunan transaksi yang berisi *itemset* I.

Sebuah *itemset* I dikatakan *closed* jika dan hanya jika :

$$c(I) = f(g(I)) = f \circ g(I) = I$$

Dimana fungsi komposit c = f o g disebut operator *Galois* atau operator *closure*

Untuk menghitung *closure* pembangkitan X, diterapkan operator Galois c. Penerapan c memerlukan interseksi semua transaksi termasuk X. Cara lain untuk memperoleh closure ini disarankan dengan *lemma* berikut :

Lemma 1 : diberikan *itemset* X dan sebuah item $i \in \chi$, $g(X) \subseteq g(i) \Leftrightarrow i \in c(X)$

Dari lemma 1, dapat disimpulkan bahwa jika $g(X) \subseteq g(i)$ maka $i \in c(X)$. Oleh karena itu dengan melakukan pemeriksaan *inclusion* untuk semua item dalam χ yang tidak termasuk didalamnya X, dapat secara *incremental* menghitung c(X)

Untuk melakukan penggalian semua *frequent closed itemsets* agar tidak terjadi duplikasi, dilakukan dengan pemeriksaan *order preserving generators* (Lucchesse, 2005). Diperkenalkan definisi berikut :

Definisi 1 : sebuah generator $X = Y \cup i$, dimana Y adalah *closed itemset* dan $i \notin Y$, dikatakan *closed* jika ada $c(X) = X$ atau $i \prec (c(X) \setminus X)$

Definisi 2 : diberikan sebuah generator *gen*= $Y \cup i$, dimana Y adalah *closed itemsets* dan $i \notin Y$, didefinisikan *pre-set*(*gen*) sebagai berikut :

$$pre\text{-}set(gen) = \{ j \mid j \in \chi, j \notin gen, \text{dan } j \prec i \}$$

Lemma 2 memberikan gambaran cara untuk memeriksa properti *order preserving* dari *gen* dengan mempertimbangkan *tidlist* $g(j)$, untuk semua $j \in pre\text{-}set(gen)$

Lemma 2: Diberikan $gen = Y \cup i$ menjadi sebuah generator dimana Y adalah *closed itemsets* dan $i \notin Y$. Jika $\exists j \in pre\text{-}set(gen)$ sedemikian sehingga $g(gen) \subseteq g(j)$ sehingga *gen* bukan *order preserving*. Jika sebuah generator bukan

merupakan *order preserving* maka dapat dilakukan *pruning* terhadapnya sehingga dengan metode ini menjadi efisien karena tidak memerlukan penyimpanan *closed itemset* sebelumnya untuk melakukan pemeriksaan duplikasi.

Berdasarkan Gambar 2.8 maka penjelasan lebih rincinya sebagai berikut :

a. Prosedur mempunyai tiga input parameter : sebuah *closed itemset* CLOSED_SET, dan dua himpunan item, *PRE_SET* dan POST_SET. Outputnya adalah semua *closed itemsets* dengan menganalisa semua pembangkitan yang sah.
b. *Dataset* D pertamakali di-*scan* untuk menentukan *frequent single item* $\mathcal{F}_1 \subseteq \chi$ dan membuat *dataset* vertikal *bitwise* $\mathcal{VD}$ yang berisi *tidlist* g(i), $\forall i \in \mathcal{F}_1$. Prosedur lalu dipanggil dengan argumen yang diberikan : CLOSED_SET = c(0), *PRE_SET* = 0, dan POST_SET = $\mathcal{F}_1 \setminus$ c(0)
c. Prosedur membuat semua kemungkinan pembangkitan, dengan cara memperluas CLOSED_SET dengan berbagai item $i \in$ POST_SET (baris 2 – 5). Item i dipilih sesuai dengan urutan $\prec$ (baris 3)
d. Pembangkitan yang tidak *frequent* dan tidak *order preserving* dihapus (baris 6) tanpa menghitung *closure*-nya. Item $i \in$ POST_SET yang telah digunakan untuk memperoleh pembangkitan yang tidak sah tidak dipertimbangkan lagi dalam pemanggilan *recursive* selanjutnya.
e. Menghitung *closure* pembangkitan yang *valid* (baris 7 – 15)
f. Pada akhir proses, sebuah *closed set* baru ($CLOSED\_SET_{New} \leftarrow$ c(new_gen)) diperoleh (baris 16). Dari *closed set* baru ini, pembangkitan baru dapat dibuat, dengan secara *recursive* memanggil prosedur $DCI\_CLOSED_d()$ (baris 17)

Berdasarkan pada tinjauan pustaka yang telah diuraikan sebelumnya maka beberapa hal yang berkaitan dengan kelebihan dan kelemahan algoritma CHARM dan algoritma DCI_CLOSED adalah sebagai berikut :

a. Algoritma CHARM masih memiliki dua kelemahan. Pertama, algoritma tidak dapat secara langsung melakukan penggalian *closed itemsets,* karena algoritma harus melakukan komputasi yang redundan untuk menentukan apakah *frequent itemset* yang dihasilkan *closed* atau tidak. Kedua, algoritma tidak efisien karena menggunakan tabel *hash* untuk menyimpan semua *closed*

itemsets yang dihasilkan sebelumnya untuk melakukan pemeriksaan duplikasi (Lucchesse, 2005).

b. Algoritma DCI_CLOSED menggunakan format data vertical *bitvectors* dalam proses perhitungan *closure*. Sesuai dengan penelitian yang dilakukan sebelumnya (Zaki, 2003), penggunaan format data vertikal *bitvectors* yang terkompresi hanya mempunyai rasio kompresi sebesar 2 sampai dengan 3 dibandingkan dengan penggunaan format data vertikal *tidset*. Di lain pihak, penggunaan penggunaan format data vertikal *diffset* dapat mempunyai rasio kompresi sebesar 2 sampai dengan 5 dibandingkan penggunaan format data vertikal *tidset*. Penggunaan format data vertikal *bitvectors* merupakan kelemahan yang dimiliki algoritma DCI_CLOSED, sedangkan kelebihannya adalah proses pemeriksaan duplikasi dilakukan tanpa harus menyimpan *closed itemset* sebelumnya.

Dengan memperhatikan kelebihan dan kelemahan yang dimiliki oleh algoritma CHARM dan algoritma DCI_CLOSED sehingga ada peluang penelitian untuk melakukan perbaikan algoritma CHARM dalam melakukan penggalian *frequent closed itemsets*. Pada penelitian ini tidak dilakukan perbaikan algoritma DCI_CLOSED karena :

- Kinerja algoritma DCI_CLOSED hanya dibandingkan dengan algoritma FP-CLOSE dan CLOSET+ (Lucchesse, 2005), sehingga tidak bisa diketahui perbandingan kinerja algoritma DCI_CLOSED dengan algoritma CHARM.
- Kinerja algoritma CHARM lebih baik dibandingkan dengan algoritma CLOSET+ (Zaki, 2005)

Algoritma yang dikembangkan mengintegrasikan format data vertikal *diffset* dan pemeriksaan duplikasi tanpa harus menyimpan semua *frequent closed itemsets* hasil enumerasi sebelumnya.

BAB 3

PERANCANGAN ALGORITMA

BAB 3
PERANCANGAN ALGORITMA

Penggalian *frequent closed itemsets* merupakan salah satu bidang penelitian yang menarik dalam pencarian kaidah asosiasi (*association rule*) karena dapat secara unik menentukan himpunan semua *frequent closed itemsets* dan *support*nya. Telah banyak penelitian dilakukan untuk melakukan enumerasi *frequent closed itemsets*, salah satunya adalah CHARM (Zaki, 2005) yang dianggap sebagai algoritma efisien untuk mengenumerasi himpunan semua *frequent closed itemsets* karena :

a. Dalam melakukan enumerasi menggunakan format data vertikal *diffset* yang mempunyai kompresi lebih besar dibandingkan menggunakan format data vertikal *bitvectors* (Zaki, 2003)
b. Menggunakan 4 (empat) buah properti dan metode pengurutan 1-*itemset* yang didasarkan pada nilai *support,* sehingga dapat mengurangi tingkat enumerasi dan percabangan.

Untuk mempermudah pemahaman tentang 4 (empat) buah properti yang dimiliki oleh algoritma CHARM, maka berikut ini ditulis kembali 4 (empat) buah properti tersebut. Jika $m(X_i)$ menyatakan jumlah ketidaksesuaian *diffset* $d(X_i)$ dan X_i menyatakan *itemset* X_i, demikian juga jika dinyatakan $m(X_j)$ adalah ketidaksesuaian *diffset* $d(X_j)$ dan X_j menyatakan *itemset* X_j . 4 (empat) buah properti dapat digunakan seperti tersebut di bawah ini :

a. Properti 1. $m(X_i) = 0$ dan $m(X_j) = 0$, lalu $d(X_i) = d(X_j)$ atau $t(X_i) = t(X_j)$
b. Properti 2, $m(X_i) > 0$ dan $m(X_j) = 0$, lalu $d(X_i) \supset d(X_j)$ atau $t(X_i) \subset t(X_j)$
c. Properti 3. $m(X_i) = 0$ dan $m(X_j) > 0$, lalu $d(X_i) \subset d(X_j)$ atau $t(X_i) \supset t(X_j)$
d. Properti 4. $m(X_i) > 0$ dan $m(X_j) > 0$, lalu $d(X_i) \neq d(X_j)$ atau $t(X_i) \neq t(X_j)$

Dari empat buah properti di atas dapat disimpulkan bahwa properti 1 dan properti 2 tidak menambah kesamaan klas baru. Untuk itu CHARM menggunakan metode pengurutan 1-*itemset* yang didasarkan pada nilai *support*nya sehingga

peluang terjadinya properti 1 dan properti 2 lebih besar dibandingkan dengan dua properti lainnya.

3.1 Analisis Kompleksitas Algoritma CHARM

Dalam melakukan enumerasi *frequent closed itemsets,* CHARM nampak kurang efisien karena masih memiliki dua kelemahan. Pertama, algoritma tidak dapat secara langsung melakukan penggalian *frequent closed itemsets* karena harus melakukan komputasi yang redundan. Kedua, algoritma kurang efisien dalam penggunaan memori karena menggunakan tabel *hash* untuk menyimpan semua *frequent closed itemsets* sebelumnya dalam melakukan pemeriksaan duplikasi. Untuk keperluan analisis algoritma, maka berikut ini dituliskan kembali algoritma CHARM secara lengkap seperti ditunjukkan dalam Algoritma 3.1.

Algoritma 3.1 Algoritma CHARM

CHARM (D, min_sup):
1. $[0] = \{l_i \times t(l_i) : l_i \in \chi \wedge \sigma(l_i) \geq minsup\}$
2. CHARM-EXTEND ([0], C=ϕ)
3. return C // *all closed sets*

CHARM-EXTEND([P], C):
4. **for each** $l_i \times t(l_i)$ **in** [P]
5. Pi = P $\cup$ l_i dan $[P_i] = \phi$
6. **for each** $l_j \times t(l_j)$ in [P], with j > I
7. X = l_j dan Y = $t(l_i) \cap t(l_j)$
8. **CHARM-PROPERTY**(X x Y, l_i, l_j, P_i, $[P_i]$, [P])
9. **end for**
10. **SUBSUMPTION-CHECK** (C,P_i)
11. **CHARM-EXTEND** ($[P_i]$,C)
12. **Hapus** $[P_i]$
13. **end for**

CHARM-PROPERTY(X x Y, l_i, l_j, P_i, $[P_i]$, [P]):
14. **if** ($\sigma(X) \geq minsup$) **then**
15. **if** $t(l_i) = t(l_j)$ **then** // Properti 1 (equal)
16. **Hapus** l_j dari [P]
17. $P_i = P_i \cup l_j$
18. **else if** $t(X_i) \subset t(X_j)$ **then** // Properti 2 (subset)
19. $P_i = P_i \cup l_j$
20. **else if** $t(X_i) \supset t(X_j)$ **then** // Properti 3 (superset)
21. **Hapus** l_j dari [P]
22. **Tambahkan** X x Y to $[P_i]$
23. **else if** $t(X_i) \neq t(X_j)$ **then** // Properti 4 (not equal)
24. **Tambahkan** X x Y to $[P_i]$
25. **end if**
26. **end if**

Untuk memudahkan analisis kompleksitas algoritma, maka Algoritma 3.1 dibagi menjadi 3 prosedur, yaitu :

a. Prosedur CHARM untuk melakukan proses penyimpanan semua hasil *frequent closed itemsets*,
b. Prosedur CHARM-EXTEND untuk melakukan proses yang rekursif untuk untuk setiap kesamaan klas dan melakukan pemeriksaan duplikasi kandidat *frequent closed itemsets*.
c. Prosedur CHARM-PROPERTY yang berfungsi untuk melakukan pemeriksaan properti dari setiap kombinasi dalam CHARM-EXTEND.

3.1.1 Prosedur CHARM

Seperti ditunjukkan pada Algoritma 3.1, prosedur pertama mempunyai fungsi menyimpan hasil proses enumerasi *frequent closed itemset* yang dilakukan oleh prosedur CHARM-EXTEND. Proses yang dilakukan adalah menyeleksi 1-*itemset* basis data. Apabila 1-*itemset* tersebut *frequent,* maka 1-*itemset* tersebut menjadi anggota dari himpunan *itemset*. Proses yang terjadi pada baris 1 dapat menghemat proses yang dilakukan CHARM-EXTEND, karena hanya mengenumerasi 1-*itemset* yang *frequent*. Secara lengkap prosedur CHARM dapat dijelaskan sebagai berikut :

a. Baris 1 adalah proses untuk mengenumerasi semua 1 *itemset* yang mempunyai nilai *support* lebih besar dari *minsup*. Baris ini bertujuan untuk menghilangkan 1-*itemset* yang tidak *frequent* sehingga dapat mengurangi proses enumerasi.
b. Baris 2 merupakan proses utama dari algoritma karena pada proses ini enumerasi *frequent closed itemsets* dimulai. Dengan nilai awal kesamaan klas *null* [] dan himpunan *frequent closed set* C diset sama dengan nol.
c. Baris 3 merupakan hasil dari proses CHARM-EXTEND, dimana C berisi semua *frequent closed set*.

Proses utama dari prosedur CHARM terletak pada baris ke-2, yaitu proses CHARM-EXTEND, sehingga kompleksitasnya sangat tergantung pada baris ke-2. Analisis kompleksitas prosedur CHARM dijelaskan pada Sub Bab 3.1.2.

3.1.2 Prosedur CHARM-EXTEND

Prosedur kedua sebagaimana ditunjukkan pada Algoritma 4.1 adalah prosedur CHARM-EXTEND, bertugas untuk melakukan enumerasi *frequent closed itemsets*. Prosedur bermula dari CHARM-EXTEND ([P],C), dengan kesamaan klas P sama dengan *null* dan *frequent closed set* C diset sama dengan nol, dimulai dari baris ke-4 s.d baris ke-13 dilakukan secara rekursif selama anggota kesamaan klas lebih dari satu. Secara lengkap penjelasan dari prosedur CHARM-EXTEND adalah sebagai berikut :

a. Baris 4, untuk setiap item elemen dari [P] lakukan langkah 5 sampai dengan langkah 13.
b. Baris 5, proses penggabungan, diawali dari *itemset* ke-i elemen [P] dengan kesamaan klas [P] ($P_i = P \cup l_i$). Penggabungan ini menjadi klas baru [P_i] dengan anggota = 0
c. Baris 6, untuk setiap anggota kesamaan klas [P] dengan posisi lebih besar dari i lakukan langkah 7 sampai dengan langkah 8.
d. Baris 7, interseksi *tidset* l_i dan *tidset* l_j
e. Baris 8, melakukan pemeriksaan properti elemen baris 7, apabila *support* dari l_j lebih besar dari *minsup* maka lakukan langkah 15 sampai dengan 25 (CHARM-PROPERTY), apabila lebih kecil maka langsung ke langkah 10 yaitu SUBSUMPTION-CHECK.
f. Baris 10, algoritma melakukan proses pemeriksaan duplikasi terhadap P_i dengan anggota *frequent closed itemsets* C. Pemeriksaan dilakukan dengan menggunakan Algoritma 4.2 yaitu menggunakan tabel *hash*, *hash key* merupakan penjumlahan *tidset* untuk setiap kandidat *frequent closed itemsets*. Pemeriksaan dilakukan dengan cara untuk *hash key* yang sama apakah *support* kandidat sama dengan *support* di tabel *hash* atau apakah kandidat merupakan *subset* himpunan *closed set*nya(C), apabila tidak sama tambahkan kandidat *closed set* pada C.
g. Baris 11, perluasan dilakukan apabila anggota kesamaan klas [P_i] jumlahnya lebih dari 1, apabila tidak maka hanya dilakukan langkah 5, 10 dan 12
h. Baris 12, melakukan penghapusan kesamaan klas [P_i] karena anggota kelasnya sudah tidak ada (sama dengan nol).

Algoritma 3.2 Subsumption-Check

1. $h(P) = \sum_{T \in t(P)} T$
2. **for** all Y ∈ HASHTABLE [h(P) do
3. **if** $\sigma(Y) \neq \sigma(P)$ atau $P \not\subset Y$
4. $C = C \cup P$
5. **end if**
6. **end for**

Penggunaan tabel *hash* pada pada proses SUBSUMPTION-CHECK (baris 10) untuk melakukan pemeriksaan duplikasi kurang efisien karena tabel *hash* menyimpan semua *frequent closed itemsets* yang telah dienumerasi sebelumnya. Memori yang diperlukan akan semakin besar apabila jumlah *frequent closed itemsets* yang ditemukan membesar, atau dengan kata lain memori yang dibutuhkan ≈ jumlah C, dengan C adalah himpunan semua *frequent closed itemsets*. Karena SUBSUMPTION-CHECK menggunakan tabel *hash* maka kompleksitas terbaiknya ≈ $O(1)$ sedangkan kompleksitas terjeleknya (*worst case*) sesuai dengan jumlah *hash key* yang sama pada tabel *hash*.

Kompleksitas prosedur CHARM-EXTEND dipengaruhi oleh hasil dari prosedur CHARM-PROPERTY. Kompleksitas terbaik (*best case*) terjadi apabila properti yang muncul dari setiap enumerasi adalah properti 1 sehingga bisa digantikan setiap kehadiran X_i dengan $X_i \cup X_j$ dan dapat dihapus elemen X_j. Sedangkan kompleksitas terjelek (*worst case*) terjadi apabila properti yang muncul adalah properti 4 sehingga tidak ada elemen yang bisa dihapus dan menimbulkan kesamaan klas baru.

Seperti ditunjukkan Algoritma 3.1, kompleksitas algoritma tergantung pada jumlah kesamaan klas dan anggotanya. Untuk tiap kesamaan klas akan diproses secara rekursif selama anggota kelasnya lebih dari 1 (satu). Fungsi rekursif yang digunakan pada prosedur CHARM-EXTEND seperti ditunjukkan pada persamaan berikut ini :

$$M(n) = \begin{cases} M(n-1) + M(n-1) + n - 1 & n > 2 \\ 1 & n = 2 \end{cases} \qquad (3.1)$$

Berdasarkan persamaan 3.1, analisis kompleksitas terbaik (*best case*) dari prosedur CHARM-EXTEND adalah apabila properti yang muncul dari setiap enumerasi adalah properti 1. Pada kondisi ini dapat digantikan setiap kehadiran

X_i dengan $X_i \cup X_j$ dan dapat dihapus elemen X_j, sehingga kompleksitas untuk himpunan *itemset* yang berjumlah n adalah n-1 karena hanya melakukan enumerasi sebanyak n-1 kali.

$Cbest = n - 1 \approx O(n)$

Sedangkan kompleksitas terjelek (*worst case*) terjadi apabila setiap enumerasi adalah properti 4, yang berarti menambah kesamaan klas baru. Berdasarkan persamaan 4.1 analisisnya adalah sebagai berikut :

$M(n) = 2M(n-1) + n-1$ diganti $M(n-1) = 2M(n-2) + n-2$

$M(n) = 2[2M(n-2) + (n-2)] + n-1$

$= 2^2M(n-2) + 2n-4 + n-1$

$= 2^2M(n-2) + 3n - 5$ diganti $M(n-2) = 2M(n-3) + n-3$

$M(n) = 2^2[M(n-3) + (n-3)] + 3n - 5$

$= 2^3M(n-3) + 4n - 12 + 3n - 5$

$= 2^3M(n-3) + 7n - 17$

$= 2^3M(n-3) + (2^3-1)n - [(2^3-1)(3-1) + 3]$

Sehingga polanya adalah :

$M(n) = 2^iM(n-i) + (2^i-1)n - [(2^i-1)(i-1) + i]$ diganti $i = n - 2$

$= 2^{n-2}M(n-(n-2)) + (2^{n-2}-1)n - [(2^{n-2}-1)(n-2-1) + n-2]$

$= 2^{n-2}M(2) + (2^{n-2}-1)n - [(2^{n-2}-1)(n-3) + n-2]$

$= 2^{n-2} + n2^{n-2}-n - [n2^{n-2}-n-3*2^{n-2} + 3 + n - 2]$

$= 2^{n-2} + n2^{n-2} - n - [n2^{n-2}-3*2^{n-2} + 1]$

$= 2^{n-2} + n2^{n-2} - n - n2^{n-2} + 3*2^{n-2} - 1$

$= 4*2^{n-2} - n - 1$

$= 2^n - n - 1$

Sehingga $Cworst = 2^n - n - 1 \approx O(2^n)$

3.1.3 Prosedur CHARM-PROPERTY

Baris 8 adalah prosedur CHARM-PROPERTY, prosedur ini efisien karena berdasarkan properti yang dimiliki dapat mengurangi tingkat enumerasi dan percabangan. *Itemset* yang tidak *frequent* tidak dipertimbangkan dalam proses selanjutnya, sedangkan *itemset* yang *frequent* dilakukan pemeriksaan propertinya (baris 15 sampai dengan baris 24). CHARM melakukan pengurutan 1-*itemset* berdasarkan *support*nya, dengan cara ini maka peluang kejadian munculnya properti 1 dan properti 2 lebih besar dibandingkan dengan peluang kejadian dua properti lainnya, sehingga dapat mengurangi proses rekursif yang dilakukan pada

bagian kedua (CHARM-EXTEND). Detail dari bagian ketiga algoritma adalah sebagai berikut :

a. Baris 15 – 17, apabila *tidset* l_i = *tidset* l_j. Properti 1 diterapkan dengan menghapus l_j dari anggota kesamaan klas [P] dan menggabungkan l_j dengan P_i, sehingga dapat mengurangi proses percabangan.
b. Baris 18-19, apabila *tidset* X_i merupakan *subset* dari *tidset* X_j. Properti 2 diterapkan dengan menggabungkan l_j dengan P_i, maka tidak menimbulkan kesamaan klas baru sehingga dapat mengurangi tingkat enumerasi.
c. Baris 20-22, apabila *tidset* X_i merupakan *superset* dari *tidset* X_j. Properti 3 diterapkan dengan menghapus l_j dari anggota kesamaan klas [P] dan menambahkan X x Y dalam anggota kesamaan klas [P_i] dan diurut sesuai *support*nya.
d. Baris 23 – 24, apabila *tidset* X_i tidak sama dengan *tidset* X_j. Properti 4 diterapkan dengan menambahkan X x Y dalam anggota kesamaan klas [Pi] dan diurut sesuai *support*nya

Kompleksitas prosedur CHARM-PROPERTY adalah $O(1)$ karena hanya melakukan sekali pemeriksaan untuk setiap kombinasi yang dilakukan pada prosedur CHARM-EXTEND

3.1.4 Kinerja Algoritma CHARM berdasarkan Basis Data Contoh

Untuk memberi gambaran yang nyata tentang algoritma CHARM maka dipergunakan basis data contoh sebagaimana ditunjukkan pada Tabel 3.1.

Tabel 3.1 Basis Data Contoh

TID	items				
1	A	B	C	D	E
2	A	B	E		
3	A	C	E		
4	A	D	E		
5	B	C	D	E	
6	B	C	D	E	
7	D	E			

Basis data pada Tabel 3.1 harus diubah terlebih dahulu menjadi format data vertikal seperti ditunjukkan pada Tabel 3.2.

Tabel 3.2 Format Data Vertikal Basis Data Contoh

A	B	C	D	E
1	1	1	1	1
2	2	3	4	2
3	5	5	5	3
4	6	6	6	4
			7	5
				6
				7

Untuk basis data contoh seperti pada Tabel 3.2 dengan *minsup* 10% maka enumerasi *frequent closed itemsets* algoritma CHARM adalah sebagai berikut :

a. Dimulai klas *root* [] = {A x 1234, B x 1256, C x 1356, D x 14567, E x 1234567} pada baris 1. Baris 4 pertama kali diproses P_i= A , dengan [A] = 0.
b. Baris 6 untuk setiap l_j (B,C,D,E) dihitung interseksi *tidset* l_i dan l_j. AB,AC, AD *frequent* dan mempunyai properti 4 sehingga timbul cabang dan tingkat baru AB x 12, AC x 13 dan AD x 14 sedangkan AE mempunyai properti 2 sehingga digantikan semua simpul dan cabang yang berisi A dengan AE.
c. Simpul yang terbentuk adalah ABE x 12, ACE x 13 dan ADE x 14. *Subsumption check* AE, karena C (*closed set*) masih kosong sehingga AE ditambahkan pada C.
d. CHARM-EXTEND ABE x 12, simpul yang terjadi adalah ABCE x 1 dan ABDE x 1 karena dua simpul ini mempunyai *tidset* yang sama sehingga dapat dilakukan penggabungan (properti 1). Hanya ada satu simpul yang terbentuk

yaitu ABCDE x 1. Kemudian lakukan *subsumption check* simpul ABE x 12 dan tidak ada duplikasi sehingga ABE ditambahkan pada C.

e. CHARM-EXTEND ABCDE x 1, karena hanya ada satu *itemset* maka langsung ke langkah 10 SUBSUMPTION-CHECK dan tidak ada duplikasi sehinga ABCDE ditambahkan pada C. Karena tidak bisa diperluas lagi maka hapus kesamaan klas ABCDE kemudian hapus kesamaan klas *parent*nya secara rekursif.

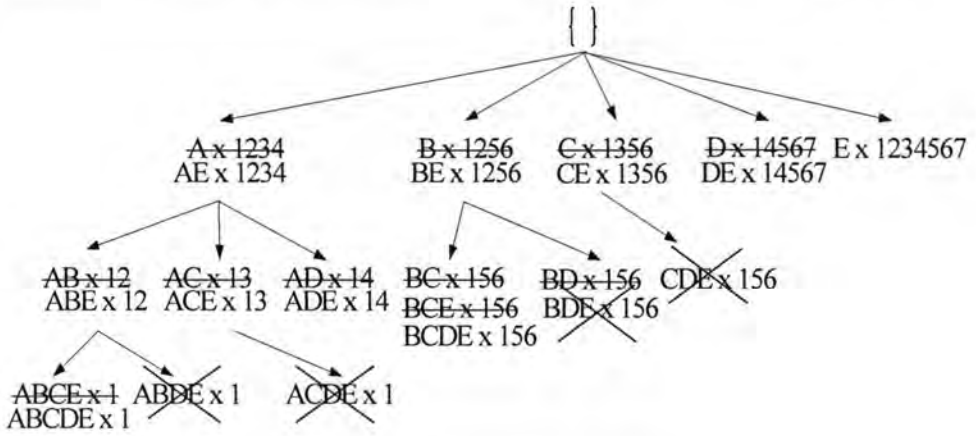
f. Setelah itu dilanjutkan dengan B dikombinasikan dengan C, D dan E. BC dan BD mempunyai properti 4 maka timbul cabang baru BC x 156 dan BD x 156, sedangkan B dengan E mempunyai properti 2 sehingga digantikan kehadiran B dengan BE, simpul yang terbentuk adalah BE x 1256 , BCE x 156 dan BDE x 156. Lakukan baris 10 SUBSUMPTION-CHECK, BE x 12 tidak ada duplikasi di tabel *hash* sehingga ditambahkan pada C. Selanjutnya CHARM EXTEND BCE dan BDE mempunyai properti 1 sehingga timbul simpul baru BCDE x 156, sedangkan BDE dihapus. Lakukan baris 10 SUBSUMPTION-CHECK, BCDE x 156 tidak ada duplikasi di tabel *hash* sehingga ditambahkan pada C

g. C dengan D dan E. Untuk C dan D terbentuk simpul baru CD x 156, sedangkan C dan E karena mempunyai properti 2 sehingga digantikan kehadiran C dengan CE, simpul yang terbentuk adalah CE x 1356 dan CDE x 156. SUBSUMPTION-CHECK terhadap CE x 1356, tidak terjadi duplikasi di tabel *hash* sehingga ditambahkan pada C. Sedangkan CDE mempunyai *hash key* yang sama dengan BCDE dan CDE merupakan *subset* BCDE sehingga CDE tidak ditambahkan pada C.

h. D dengan E mempunyai properti 2 sehingga D diganti dengan DE lalu lakukan SUBSUMPTION-CHECK karena tidak ada duplikasi tambahkan DE pada C. Terakhir SUBSUMPTION-CHECK E x 1234567, tidak ada duplikasi sehingga ditambahkan pada C.

Untuk basis data contoh pada Tabel 3.1, sebagaimana ditunjukkan pada Gambar 3.1 *frequent closed itemsets* yang terbentuk adalah 10, sehingga tabel *hash* harus menyimpan 10 buah *support* dan *itemset*nya. Simpul yang terbentuk pada algoritma CHARM tidak semuanya *closed* sehingga diperlukan pemeriksaan

duplikasi sebanyak kandidat *frequent closed itemsets* yang terbentuk. Untuk basis data contoh proses pemeriksaan duplikasi dilakukan sebanyak 10 kali.



Gambar 3.1 Proses Pencarian Algoritma CHARM

3.2 Rancangan Perbaikan Algoritma CHARM

Berdasarkan uraian diatas, rancangan perbaikan algoritma CHARM yang dilakukan terdiri dari 2 hal :

a. Menggabungkan proses SUBSUMPTION-CHECK (baris 10 Algoritma 3.1) dan melakukan pemeriksaan *order preserving* (Lucchesse, 2005) sehingga tidak diperlukan lagi tabel *hash* yang menyimpan seluruh *frequent closed itemsets* dari hasil proses enumerasi sebelumnya.
b. Pemeriksaan order preserving pada algoritma CHARM tidak dapat dilakukan secara langsung sebagaimana halnya pada algoritma DCI_CLOSED karena CHARM menggunakan 4 (empat) properti untuk melakukan proses enumerasi. Penggunaan properti ini menyebabkan adanya kemungkinan timbulnya *frequent itemset* yang tidak *closed* pada cabang yang sedang dienumerasi, sehingga pemeriksaan duplikasi tetap harus menggunakan metode *subsumption check*. Perbedaan mendasar metode *subsumption check* algoritma perbaikan CHARM adalah tabel *hash* hanya menyimpan *frequent closed itemsets* pada cabang yang sedang dienumerasi sehingga ukurannya jauh lebih kecil dibandingkan tabel *hash* yang digunakan algoritma CHARM.

Dengan perubahan ini maka proses pemeriksaan duplikasi dapat menghemat penggunaan memori

Untuk itu dirancang algoritma yang bisa mengintegrasikan kelebihan yang ada pada algoritma CHARM dan melakukan pemeriksaan duplikasi dengan metode *order preserving* sebagaimana dapat dilihat pada Algoritma 3.3. Perbaikan algoritma yang dilakukan adalah menggabungkan proses SUBSUMPTION-CHECK dan melakukan proses pemeriksaan duplikasi dengan menggunakan metode *order preserving* (baris 10). Secara detail perbaikan yang dilakukan adalah sebagai berikut :

a. Baris 10, proses pemeriksaan duplikasi dengan menggunakan metode *order preserving* dan *subsubsumption check* pada cabang yang sedang dienumerasi, apabila tidak ada duplikasi maka dilakukan penulisan *frequent closed itemsets* dan *support*nya (baris 11).
b. Untuk bisa melakukan pemeriksaan *order preserving* maka perlu adanya *PRE_SET* yang proses penambahannya dilakukan setelah cabang *frequent 1-itemset* selesai dienumerasi (baris 14). Setelah itu dilakukan penghapusan tabel *hash* (baris 15) dan l_i dari *class* [P] (baris 17).

Untuk melakukan penggalian semua *frequent closed itemsets* agar tidak terjadi duplikasi, dilakukan dengan pemeriksaan *order preserving generators*. berdasarkan *PRE_SET*nya. Jika diberikan sebuah generator *gen*= $Y \cup i$, dimana Y adalah *closed itemsets* dan $i \notin Y$, didefinisikan *pre-set*(*gen*) sebagai berikut :

$$pre\text{-}set(gen) = \{ j \mid j \in \chi, j \notin gen, \text{dan } j \prec i \} \quad (3.2)$$

Untuk memeriksa properti *order preserving* dari *gen* dengan mempertimbangkan *tidlist* $g(j)$, untuk semua $j \in pre\text{-}set(gen)$. Jika $gen = Y \cup i$ menjadi sebuah generator dimana Y adalah *closed itemsets* dan $i \notin Y$ dan jika $\exists j \in pre\text{-}set(gen)$ sedemikian sehingga $g(gen) \subseteq g(j)$, maka *gen* bukan *order preserving*. Jika sebuah generator bukan merupakan *order preserving* maka dapat dilakukan *pruning* terhadapnya. Metode ini menjadi efisien karena tidak memerlukan penyimpanan *closed itemset* sebelumnya untuk melakukan pemeriksaan duplikasi. Jumlah *PRE_SET* yang digunakan maksimal sama dengan jumlah *1-itemset frequent*,

jumlahnya jauh lebih kecil daripada jumlah *frequent closed itemsets* yang terbentuk.

Secara lengkap rancangan perbaikannya ditunjukkan pada Algoritma 3.3.

Algoritma 3.3. Rancangan Perbaikan Algoritma CHARM

CHARM (D, min_sup):
1. $[0] = \{l_i \times t(l_i) : l_i \in \chi \wedge \sigma(l_i) \geq minsup\}$
2. CHARM-EXTEND ([0], C=ϕ)

CHARM-EXTEND([P], C):
3. **for each** $l_i \times t(l_i)$ **in** [P]
4. $P_i = P \cup l_i$ dan $[P_i] = \phi$
5. **for each** $l_j \times t(l_j)$ in [P], with j > i
6. $X = l_j$ dan $Y = t(l_i) \cap t(l_j)$
7. **CHARM-PROPERTY**($X \times Y$, l_i, l_j, P_i, $[P_i]$, [P])
8. **end for**
9. **if !IS_DUP**(P_i,Y)
10. **Tuliskan P_i and *support***
11. **CHARM-EXTEND** ($[P_i]$,C)
12. **Hapus** $[P_i]$
13. **if [P] > 1 and level=root then**
14. ***PRE_SET* = *PRE_SET*** $\cup$ **($t(l_i)$)**
15. **Hapus C**
16. **end if**
17. **Hapus li from [P]**
18. **end for**

CHARM-PROPERTY($X \times Y$, l_i, l_j, P_i, $[P_i]$, [P]):
19. **if** ($\sigma(X) \geq$ min_sup) **then**
20. **if** $t(l_i) = t(l_j)$ **then** // Properti 1 (equal)
21. **Hapus** l_j dari [P]
22. $P_i = P_i \cup l_j$
23. **else if** $t(X_i) \subset t(X_j)$ **then** // Properti 2 (subset)
24. $P_i = P_i \cup l_j$
25. **else if** $t(X_i) \supset t(X_j)$ **then** // Properti 3 (superset)
26. **Hapus** l_j dari [P]
27. **Tambahkan** X x Y to $[P_i]$
28. **else if** $t(X_i) \neq t(X_j)$ **then** // Properti 4 (not equal)
29. **Hapus** X x Y to $[P_i]$
30. **end if**
31. **end if**

Algoritma 3.4. Algoritma Pemeriksaan Duplikasi

IS_DUP(P_i,Y)

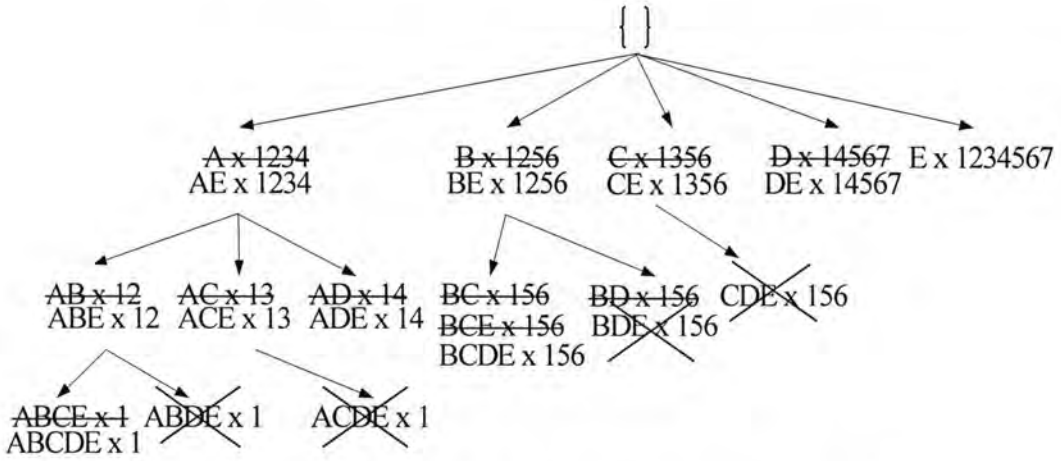
1. subset = false
2. $h(P_i) = \sum_{T \in \sigma(P_i)} T$
3. **for** all Y ∈ HASHTABLE [h(Pi) do
4. **if** $\sigma(Y) \neq \sigma(P_i)$ atau $Pi \not\subseteq Y$
5. $C = C \cup P_i$
6. **else**
7. subset=true
8. **end if**
9. **end for**
10. **if** !subset **then**
11. **while** all j ∈ *PRE_SET* & !subset
12. **if** $g(P_i) \subseteq g(j)$ then
13. subset = true
14. **end if**
15. **end while**
16. **end if**
17. **return** subset

Kompleksitas rancangan perbaikan algoritma seperti ditunjukkan pada Algoritma 3.3 hampir sama dengan kompleksitas Algoritma 3.1, perbedaannya hanya pada proses pemeriksaan duplikasi. Pemeriksaan duplikasi dengan menggunakan Algoritma 3.4 dilakukan dengan menggabungkan metode SUBSUMPTION-CHECK dan metode *order preserving*. SUBSUMPTION-CHECK dilakukan untuk melakukan pemeriksaan duplikasi pada cabang 1-*itemset frequent* yang sedang dienumerasi, sehingga tabel *hash* hanya menyimpan semua C yang sedang dienumerasi pada cabang tersebut. Kompleksitas terbaik dari algoritma ini adalah $O(1)$, apabila duplikasi terjadi pada cabang yang sedang dienumerasi. Kompleksitas terjeleknya (*worst case*) terjadi apabila tidak ada duplikasi pada cabang yang sedang dienumerasi sehingga harus menggabungkan metode SUBSUMPTION-CHECK (baris 1 s.d baris 9) dan metode *order preserving*. Pada kondisi ini jumlah pemeriksaan duplikasi dipengaruhi oleh posisi kandidat tersebut. Apabila suatu kandidat *frequent closed itemsets* berada pada posisi cabang *root* ke-n maka pemeriksaan dilakukan sebanyak n kali ditambah jumlah *hash key* yang sama pada tabel *hash*. Kompleksitasnya adalah $O(n)$.

Tree yang terbentuk sebagaimana ditunjukkan pada Gambar 3.2, perbaikan yang dilakukan adalah :

a. Tabel *hash* yang dipergunakan tidak menyimpan seluruh *frequent closed itemsets* yang terbentuk tetapi hanya menyimpan *frequent closed itemsets* pada cabang yang dienumerasi. Contoh pada saat memeriksa CE maka tabel *hash* yang digunakan adalah kosong, sedangkan pada algoritma CHARM tabel *hash* yang digunakan berisi 7 record.

b. Jumlah *PRE_SET* yang digunakan hanya berisi 4 record, dimana jumlah ini masih lebih kecil dibandingkan jumlah record tabel *hash* (9 record). Pada kondisi terjelek jumlah *PRE_SET* adalah F1 – 1, dengan F1 adalah 1-*itemset frequent*, sedangkan jumlah *frequent closed itemsets* (C) adalah 2^{F1}. Dengan demikian menggunakan metode *order preserving* yang berbasis *PRE_SET* jauh lebih efisien dibandingkan menggunakan metode *subsumption check* yang berbasis *frequent closed itemsets* hasil enumerasi sebelumnya.



Gambar 3.2 Proses Enumerasi dengan Algoritma Perbaikan CHARM

BAB 4

UJI COBA DAN ANALISA HASIL

BAB 4
UJI COBA DAN ANALISIS HASIL

Algoritma perbaikan CHARM (CHARM_NEW) dan algoritma CHARM diuji dengan beberapa basis data *real* dan *sintetik*. Uji coba dan analisis hasil dilakukan sesuai dengan kerangka seperti tersebut di bawah ini :

1. Lingkungan uji coba
2. Data uji coba
3. Skenario uji coba
4. Pelaksanaan uji coba
5. Hasil uji coba
6. Analisis hasil uji coba

4.1 Lingkungan Uji Coba

Pengujian dilakukan dengan menggunakan 2 (dua) unit PC, masing-masing dengan spesifikasi sebagai berikut :

a. Processsor : Intel Pentium IV 3000 Mhz, 1 GB RAM
b. Sistem operasi : Windows XP Pro
c. Bahasa pemrograman : Microsoft Visual C++ 6.0

1 (satu) unit PC digunakan untuk menguji kinerja algoritma CHARM, sedangkan satu unit PC lainnya digunkan untuk menguji kinerja algoritma perbaikan CHARM (CHARM_NEW). Kedua algoritma dikodekan dengan bahasa pemrograman Microsoft Visual C++ 6.0.

4.2 Data Uji Coba

Basis data uji coba di*download* dari http://www.rcs.edu /~zaki/. Basis data Pumsb (Pumsb dan Pumsb*) berisi data sensus. Pumsb* pada dasarnya sama

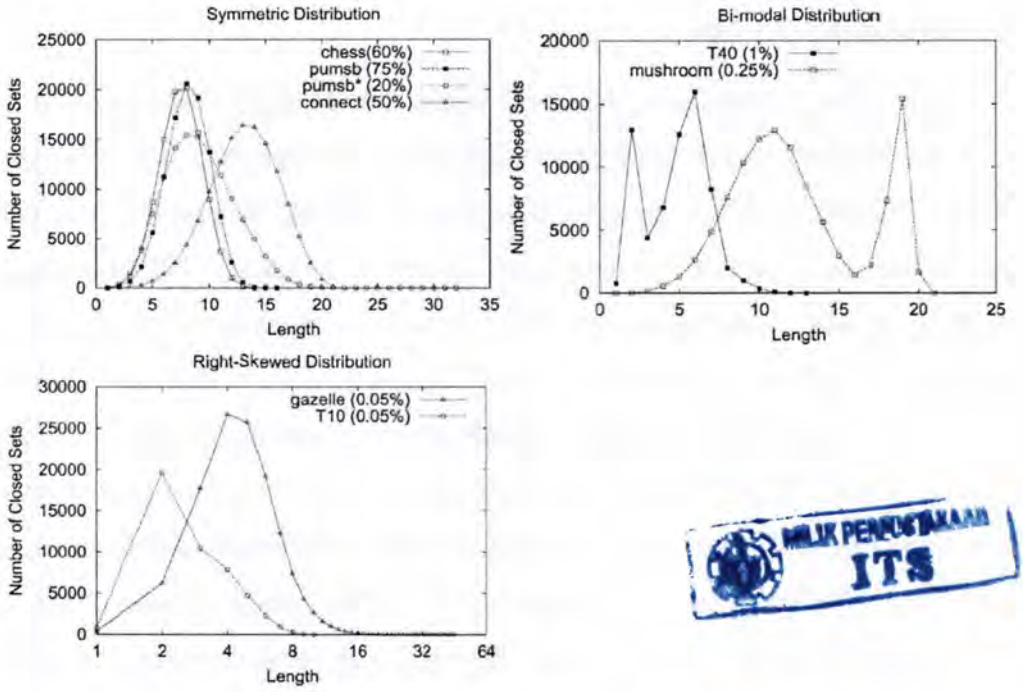
dengan Pumsb yang membedakan adalah Pumsb* tidak mempunyai item dengan minimum *support* 80% atau lebih. Basis data Mushroom berisi karakteristik berbagai spesies jamur. Basis data Connect dan Chess diberikan dari tahapan game masing-masing. Tiga basis data berikutnya berasal dari UC Irvine Machine Learning Database Repository. Basis data *sintetik* (T10 dan T40) menggunakan IBM generator merupakan transaksi mimik pada lingkungan *retailing*. Basis data Gazelle berasal dari *click-stream* data perusahaan dot-com kecil Gazelle.com, retailer *legware* dan *legcare* yang tidak eksis lagi. Umumnya basis data *real* sangat rapat sedang basis data *sintetik* jarang apabila dibandingkan dengan basis data *real*.

Tabel 4.1 menunjukkan karakteristik basis data yang digunakan dalam evaluasi. Kolom 1 menyatakan nama basis data, terdiri dari basis data real dan sintetik. Kolom 2 menyatakan jumlah item yang digunakan dalam basis data. Kolom 3 menunjukkan rerata jumlah *itemset* dalam setiap transaksi, dan kolom 4 menunjukkan jumlah record setiap basis data. Kolom 5 menunjukkan jumlah maksimum pola *frequent closed itemsets* pada minimum *support* terendah yang digunakan pada pengujian. Kolom 6 menunjukkan minimum *support* terkecil yang dipergunakan pada waktu pengujian. Sebagai contoh basis data Gazelle, pada minimum *support* 0,01%. *closed pattern* terpanjangnya adalah 154. Pada pola yang panjang, banyak metode penggalian semua *frequent pattern* menjadi tidak praktis (Zaki, 2005). Hasil ini memberikan petunjuk keefektifan CHARM dalam melakukan penggalian *closed pattern* terutama karena adanya perulangan properti 1 dan 2 sebagaimana telah dijelaskan pada BAB 2

Tabel 4.1 Karakteristik Basis Data

Basis Data	Jml Item	Rerata Panjang	Jml Record	Maks. Pattern	Minimum Support
(1)	(2)	(3)	(4)	(5)	(6)
Chess	76	37	3.196	21	25%
Connect	130	43	67.557	29	10%
Mushroom	120	23	8.124	21	0.075%
Pumsb*	7.117	50	49.046	38	10%
Pumsb	7.117	74	49.046	24	50%
Gazelle	498	2.5	59.601	154	0.01%
T10I4D100K	1.000	10	100.000	13	0.01%
T40I10D100K	1.000	40	100.000	20	0.2%

Basis data dikelompokkan sesuai tipe distribusinya (Zaki, 2005) seperti terlihat pada Gambar 4.1. Chess, Pumsb*, Pumsb dan Connect menunjukkan sebuah distribusi yang hampir simetrik dari *closed frequent pattern.* T40 dan Mushroom menunjukkan kecenderungan distribusi *bimodal closed sets.* T40 seperti T10 mempunyai banyak *pattern* pendek dengan panjang 2, tetapi juga mempunyai puncak lain dengan panjang 6. Mushroom mempunyai *pattern* yang lebih panjang apabila dibandingkan basis data T40, puncak kedua terjadi dengan panjang 19. Akhirnya Gazelle dan T10 mempunyai distribusi *right-skewed.* Gazelle cenderung mempunyai banyak *pattern* kecil dengan *right tail* sangat panjang. T10 mempunyai distribusi yang hampir sama dengan mayoritas *closed pattern* berawal dengan panjang 2.



Gambar 4.1 Jumlah *Frequent closed itemsets* dan Distribusi Panjangnya (Zaki, 2005)

4.3 Skenario Uji Coba

Untuk mengetahui kinerja algoritma dan untuk keperluan evaluasi maka dua skenario diterapkan untuk tiap basis data :

a. Membandingkan efisiensi memori antara algoritma yang dikembangkan dengan algoritma CHARM
b. Melihat karakteristik kecepatan komputasi algoritma yang dikembangkan dibandingkan dengan algoritma CHARM

Berdasarkan kedua skenario ini maka uji coba tiap basis data dikelompokkan sesuai dengan tipe distribusinya, sebagaimana telah dijelaskan dalam Sub Bab 4.2. Tujuannya adalah mengetahui pengaruh distribusi panjang *closed pattern* basis data terhadap algoritma CHARM dan algoritma CHARM_NEW

4.4. Pelaksanaan Uji Coba

Uji coba dilaksanakan dengan 2 (dua) unit komputer, masing-masing dengan spesifisikasi seperti yang telah dijelaskan pada Sub Bab 4.1. Satu unit komputer digunakan untuk menguji algoritma CHARM, sedangkan satu unit komputer lainnya digunakan untuk menguji algoritma CHARM_NEW. Tiap basis data diuji dengan berbagai macam nilai minimum *support*. Hasil uji coba algoritma CHARM disimpan dalam sebuah file dengan format Performance_Charm_Hash_NameBasisDataSupport%, sedangkan hasil uji coba algoritma CHARM_NEW disimpan dalam sebuah file dengan format Performance_Charm_Preset_NamaBasisDataSupport%. Beberapa parameter yang dicatat dalam file hasil uji coba algoritma CHARM adalah jumlah *frequent closed itemsets*, jumlah memori yang dipergunakan, waktu proses, rata-rata memori, maksimum *pattern* dan jumlah *frequent* 1 *itemset* (F1). File hasil uji coba algoritma CHARM_NEW isinya hampir sama dengan file hasil uji coba algoritma CHARM, perbedaannya adalah adanya tambahan parameter jumlah *PRE_SET* selama enumerasi.

4.5. Hasil Uji Coba

Sesuai dengan skenario dan pelaksanan uji coba seperti dijelaskan dalam Sub Bab 4.3 dan Sub Bab 4.4, hasil uji coba seperti terlihat pada Sub Bab 4.5.1 sampai dengan Sub Bab 4.5..3

4.5.1 Hasil Uji Coba Basis Data Simetrik

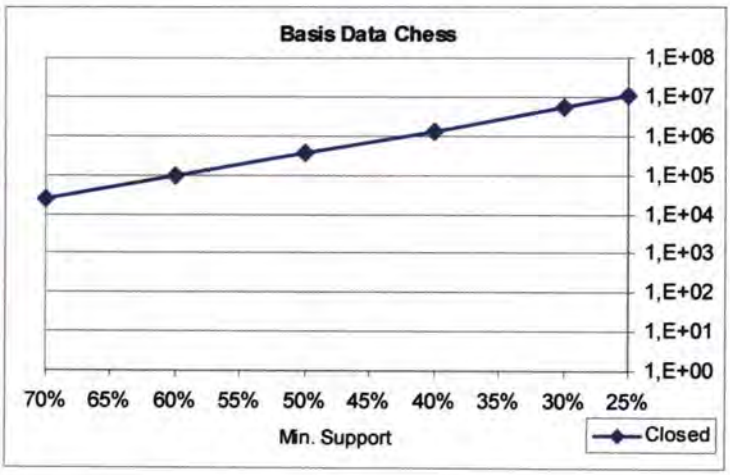
Tabel 4.2 menjelaskan daftar tabel dan nama gambar hasil uji coba basis data simetrik seperti

Tabel 4.2 Daftar Tabel dan Gambar Hasil Uji coba Basis Data Simetrik

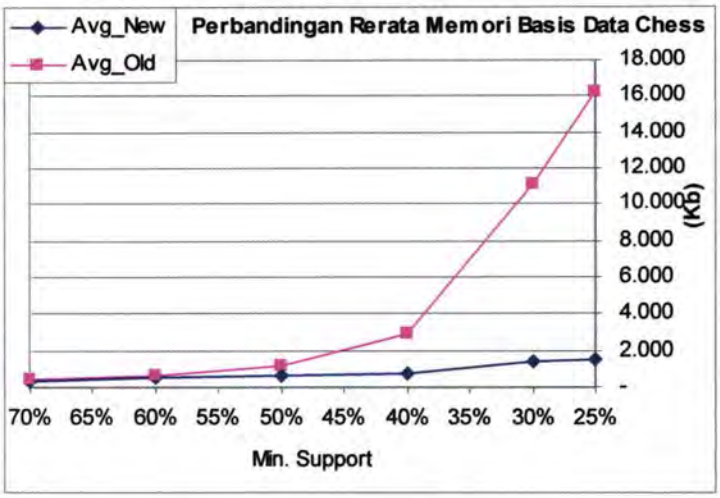
Basis Data	Nama Tabel	Nama Gambar
Chess	Tabel 4.3	Gambar 4.2, Gambar 4.3 dan Gambar 4.4
Pumsb	Tabel 4.4	Gambar 4.5, Gambar 4.6 dan Gambar 4.7
Pumsb*	Tabel 4.5	Gambar 4.8, Gambar 4.9 dan Gambar 4.10
Connect	Tabel 4.6	Gambar 4.11, Gambar 4.12 dan Gambar 4.13

Tabel 4.3 Hasil Uji Coba Basis Data Chess

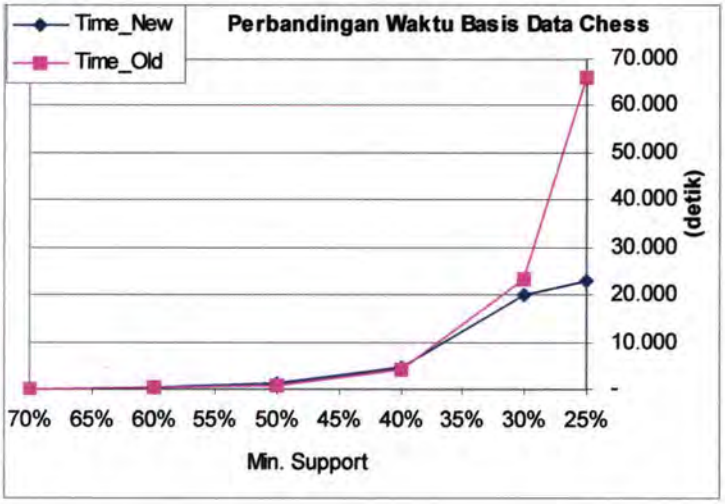
Minsup (%)	Closed	F1	*PRE_SET*	Time (sec)			Avg Memori (Kb)		Efisiensi (%)
				New	Old	%	New	Old	
70	23.892	24	23	57	55	103,64	363	399	9,02
60	98.392	34	33	257	257	100,00	498	628	20,70
50	369.450	37	36	1.069	1.031	103,69	605	1.159	47,80
40	1.361.157	40	39	4.493	4.361	103,03	800	2.947	72,85
30	5.316.467	50	49	20.085	23.539	85,33	1.378	11.084	87,59
25	10.849.724	51	50	22.845	66.006	34,61	1.536	16.213	90,53



Gambar 4.2 Grafik Jumlah *Frequent closed itemsets* Basis Data Chess



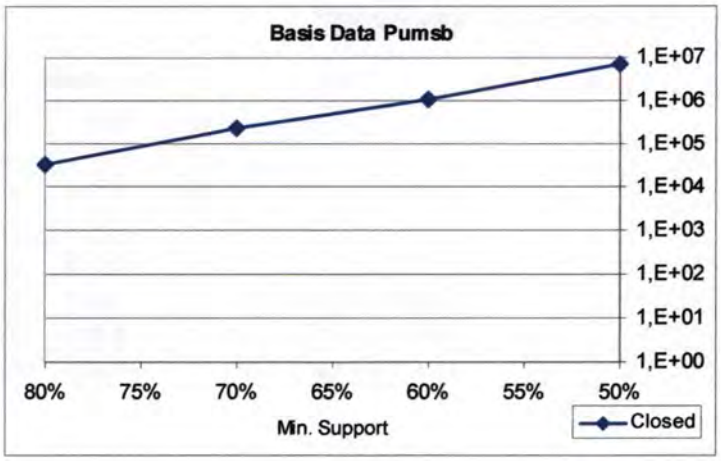
Gambar 4.3 Grafik Perbandingan Rerata Memori Basis Data Chess



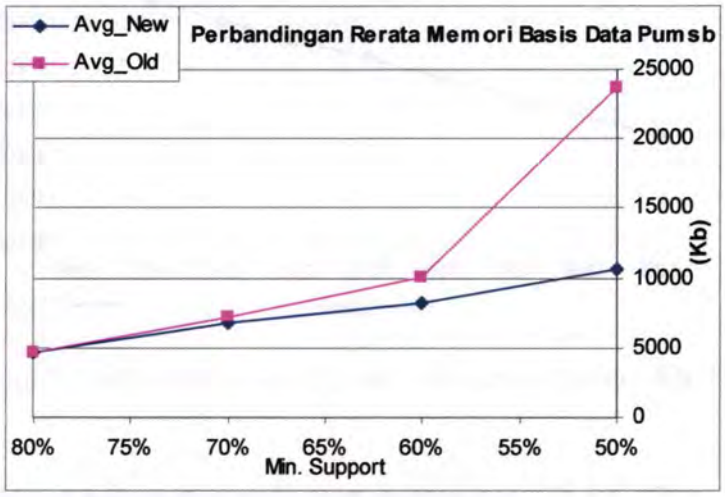
Gambar 4.4 Grafik Perbandingan Waktu Basis Data Chess

Tabel 4.4 Hasil Uji Coba Basis Data Pumsb

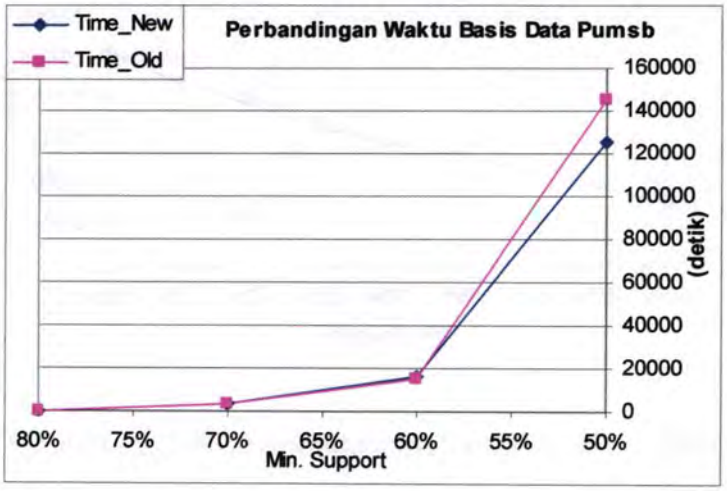
Minsup (%)	Closed	F1	*PRE_SET*	Time (sec)			Avg Memori (Kb)		Efisiensi (%)
				New	Old	%	New	Old	
80	33.295	25	20	335	325	103,08	4.632	4.707	1,6
70	241.196	34	29	3.365	3.283	102,50	6.855	7.290	6,0
60	1.074.627	39	34	16.167	15.807	102,28	8.220	10.077	18,4
50	7.121.264	52	43	125.121	145.348	86,08	10.589	23.725	55,4



Gambar 4.5 Grafik Jumlah *Frequent closed itemsets* Basis Data Pumsb



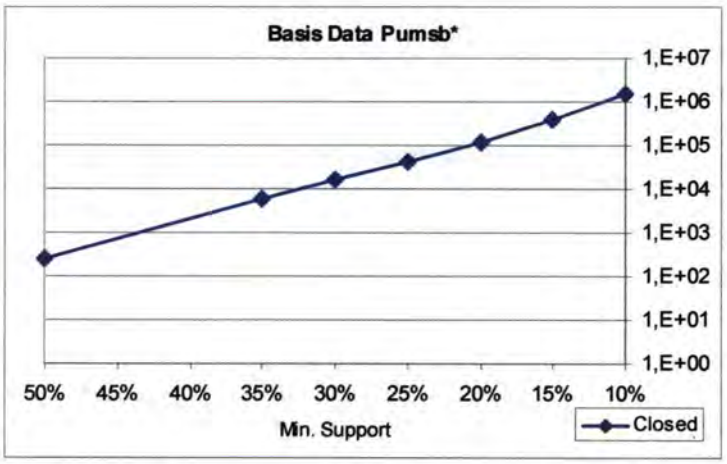
Gambar 4.6 Grafik Perbandingan Rerata Memori Basis Data Pumsb



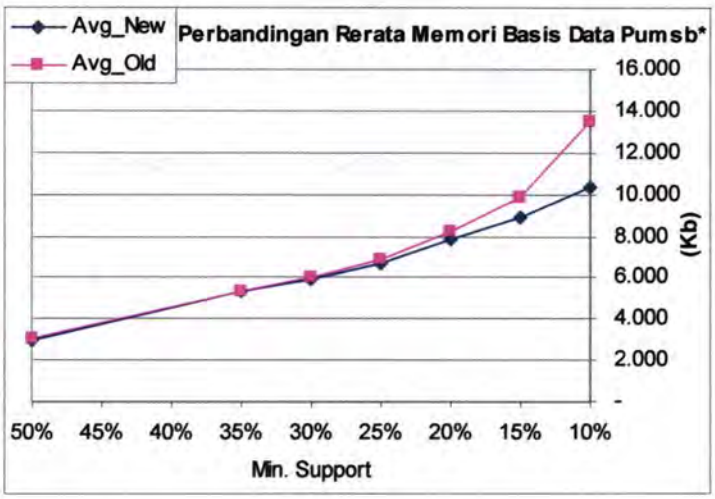
Gambar 4.7 Grafik Perbandingan Waktu Basis Data Pumsb

Tabel 4.5 Hasil Uji Coba Basis Data Pumsb*

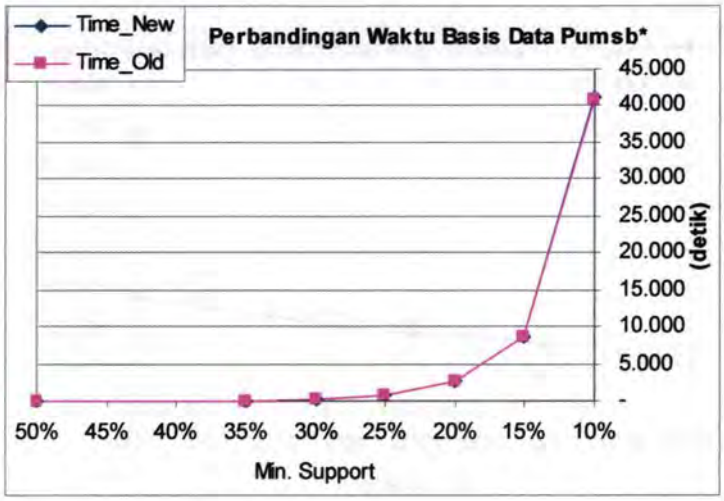
Minsup (%)	Closed	F1	*PRE_SET*	Time (sec)			Avg Memori (Kb)		Efisiensi (%)
				New	Old	%	New	Old	
50	248	27	22	9	9	100,00	3.010	3.072	2,0
35	6.129	55	44	99	97	102,06	5.350	5.369	0,4
30	16.154	60	49	255	249	102,41	5.938	6.046	1,8
25	42.756	68	57	775	759	102,11	6.728	6.907	2,6
20	122.201	86	68	2.611	2.581	101,16	7.859	8.277	5,1
15	382.392	94	76	8.641	8.557	100,98	8.888	9.886	10,1
10	1.512.865	110	92	41.175	40.851	100,79	10.326	13.522	23,6



Gambar 4.8 Grafik Jumlah *Frequent closed itemsets* Basis Data Pumsb*



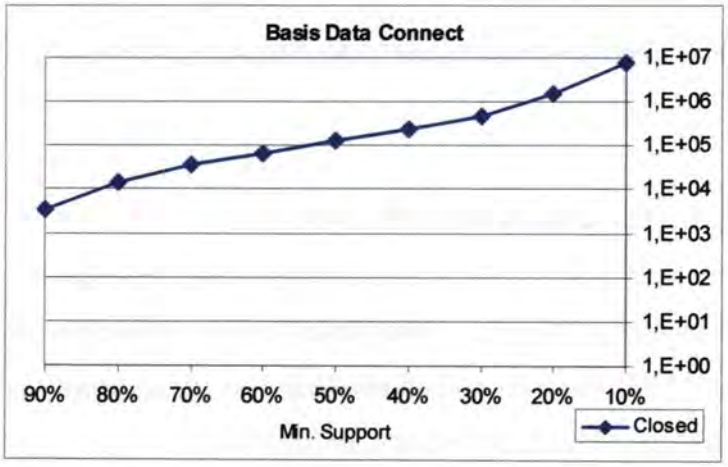
Gambar 4.9 Grafik Perbandingan Rerata Memori Basis Data Pumsb*



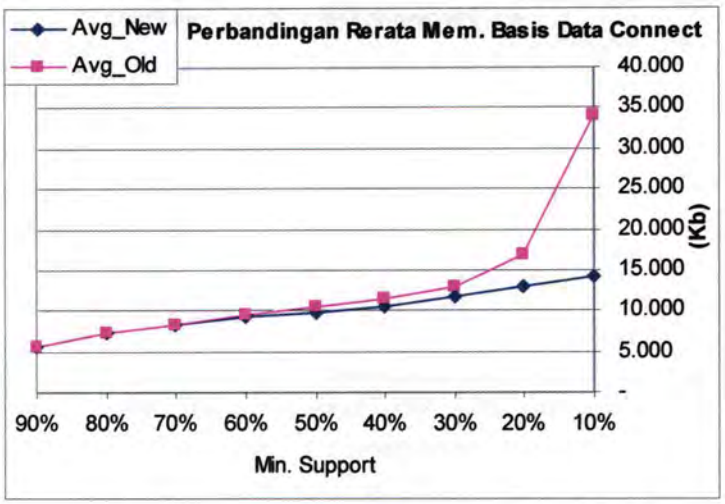
Gambar 4.10 Grafik Perbandingan Waktu Basis Data Pumsb*

Tabel 4.6 Hasil Uji Coba Basis Data Connect

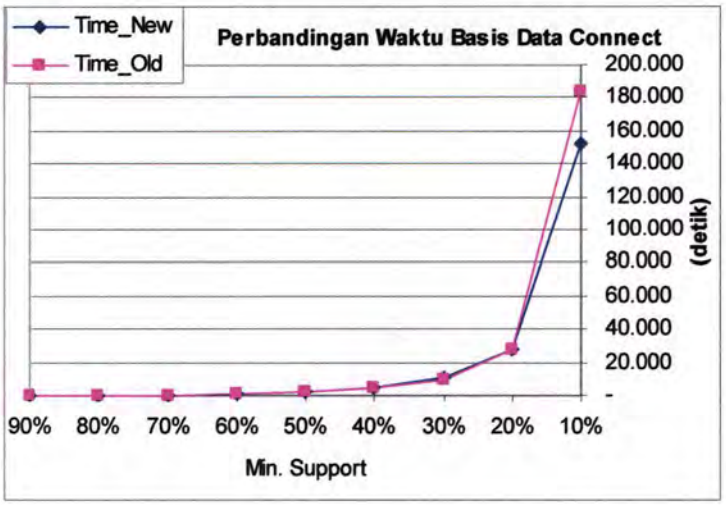
Minsup (%)	Closed	F1	*PRE_SET*	Time (sec)			Avg Memori (Kb)		Efisiensi (%)
				New	Old	%	New	Old	
90	3.486	21	20	25	23	108,70	5.561	5.570	0,16
80	15.107	28	27	143	135	105,93	7.371	7.419	0,65
70	35.875	31	30	451	435	103,68	8.225	8.401	2,09
60	68.343	36	35	1.051	1.019	103,14	9.333	9.664	3,43
50	130.101	38	37	2.301	2.245	102,49	9.895	10.436	5,18
40	239.372	41	40	4.861	4.761	102,10	10.655	11.542	7,68
30	460.356	46	45	10.387	10.235	101,49	11.669	13.102	10,94
20	1.482.861	59	58	28.579	28.117	101,64	12.987	16.844	22,90
10	8.035.411	73	72	152.469	183.928	82,90	14.294	34.080	58,06



Gambar 4.11 Grafik Jumlah *Frequent closed itemsets* Basis Data Connect



Gambar 4.12 Grafik Perbandingan Rerata Memori Basis Data Connect



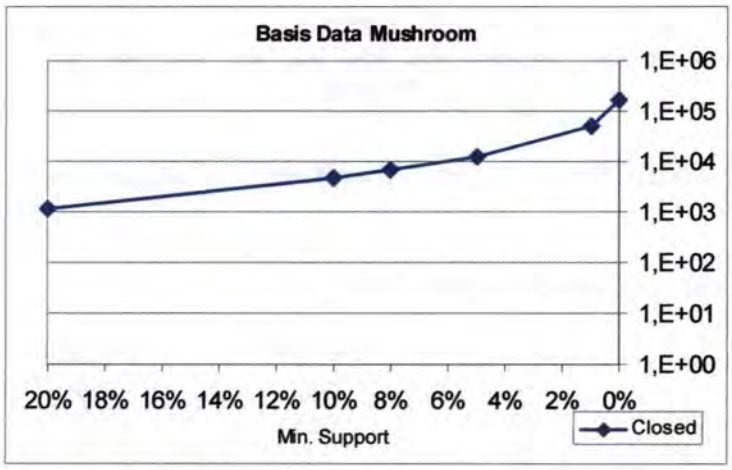
Gambar 4.13 Grafik Perbandingan Waktu Basis Data Connect

4.5.2 Hasil Uji Coba Basis Data Bimodal

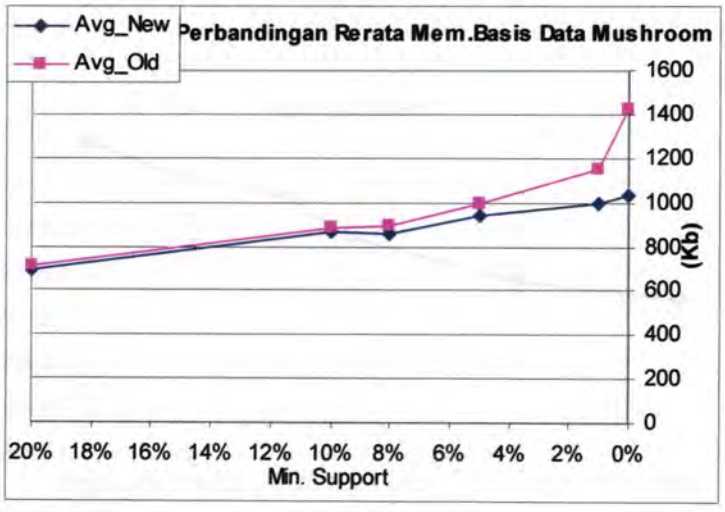
Mushroom dan T40 merupakan basis data bimodal. Hasil uji coba basis data Mushroom sebagaimana terlihat pada Tabel 4.6, Gambar 4.14, Gambar 4.15 dan Gambar 4.16. Hasil uji coba basis data T40 sebagaimana terlihat pada Tabel 4.7, Gambar 4.17, Gambar 4.18 dan Gambar 4.19.

Tabel 4.7 Hasil Uji Coba Basis Data Mushroom

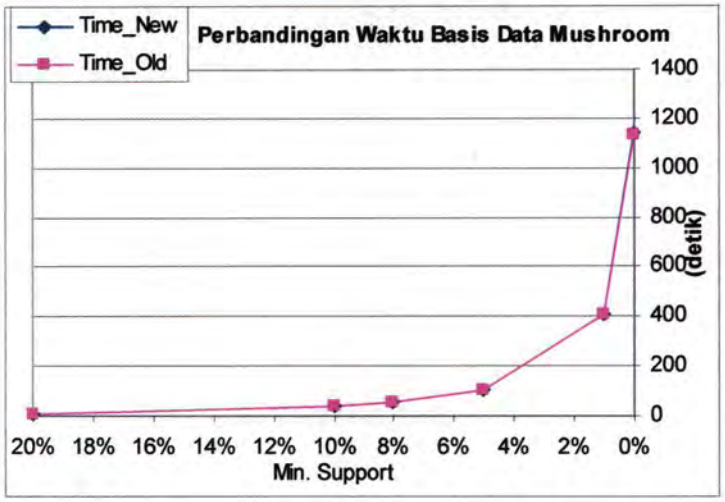
Minsup (%)	Closed	F1	*PRE_SET*	Time (sec)			Avg Memori (Kb)		Efisiensi (%)
				New	Old	%	New	Old	
20	1.197	43	42	11	11	100,00	699	709	1,4
10	4.885	56	55	39	39	100,00	868	887	2,1
8	6.749	58	57	53	53	100,00	863	900	4,1
5	12.843	73	72	105	105	100,00	939	1.001	6,2
1	51.640	96	94	413	411	100,49	997	1.152	13,5
0,0075	164.520	117	110	1.145	1.135	100,88	1.029	1.429	28,0



Gambar 4.14 Grafik Jumlah *Frequent closed itemsets* Basis Data Mushroom



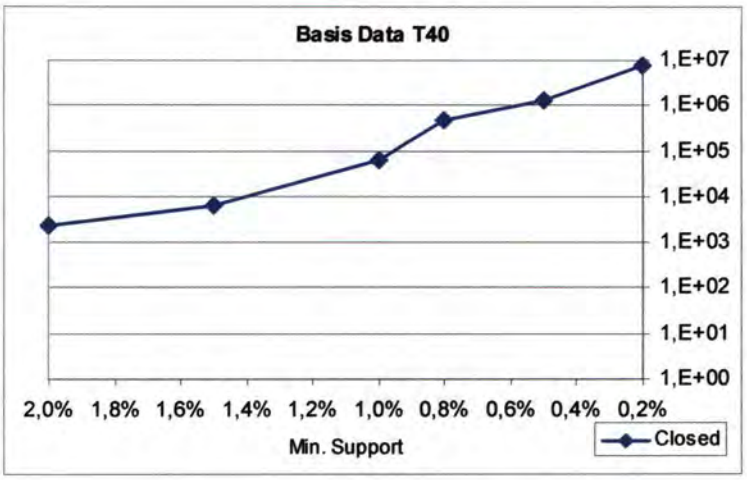
Gambar 4.15 Grafik Perbandingan Rerata Memori Basis Data Mushroom



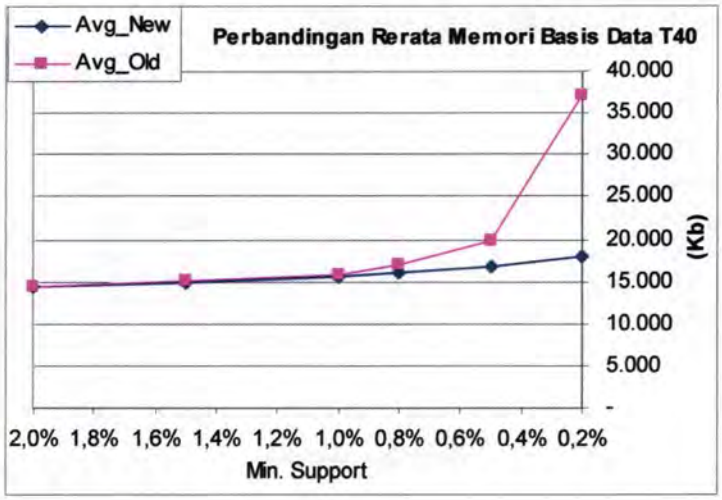
Gambar 4.16 Grafik Perbandingan Waktu Basis Data Mushroom

Tabel 4.8 Hasil Uji Coba Basis Data T40

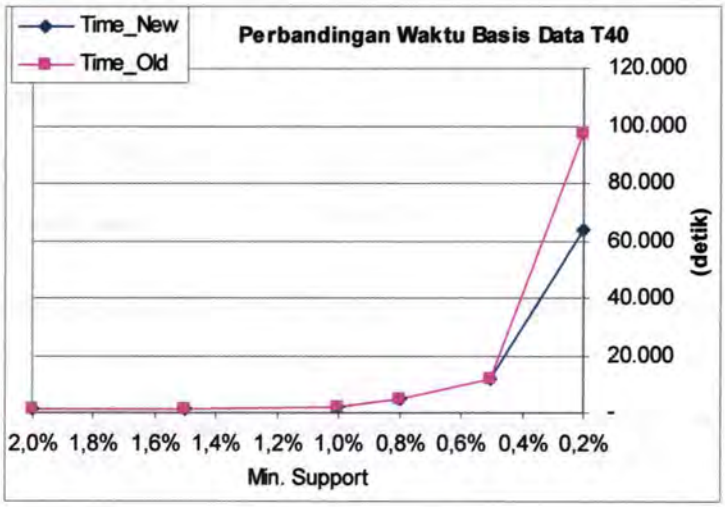
Minsup (%)	Closed	F1	*PRE_SET*	Time (sec)			Avg Memori (Kb)		Efisiensi (%)
				New	Old	%	New	Old	
2	2.293	610	609	1.085	1.085	100,00	14.417	14.426	0,1
1,5	6.539	682	681	1.373	1.369	100,29	15.014	15.053	0,3
1	65.236	755	754	2.359	2.343	100,68	15.699	15.951	1,6
0,8	477.738	783	782	5.001	4.877	102,54	16.032	17.125	6,4
0,5	1.275.939	838	837	11.763	11.645	101,01	16.694	19.886	16,1
0,2	7.629.970	906	905	63.707	97.894	65,08	17.985	37.198	51,7



Gambar 4.17 Grafik Jumlah *Frequent closed itemsets* Basis Data T40



Gambar 4.18 Grafik Perbandingan Rerata Memori Basis Data T40



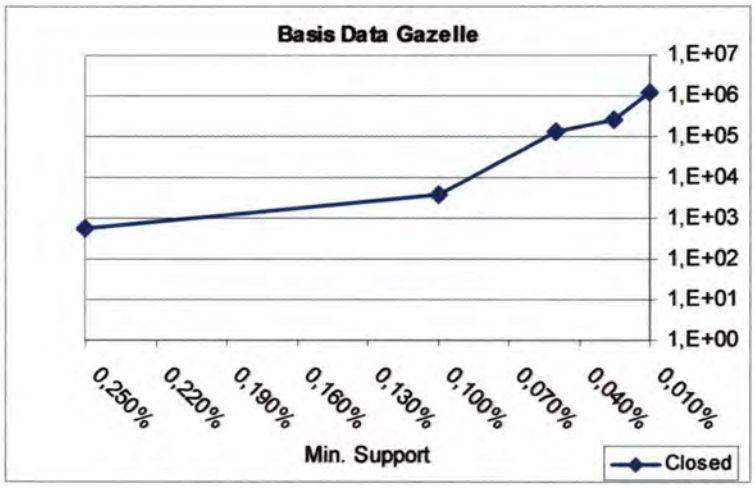
Gambar 4.19 Grafik Perbandingan Waktu Basis Data T40

4.5.3 Hasil Uji Coba Basis Data Right Skewed

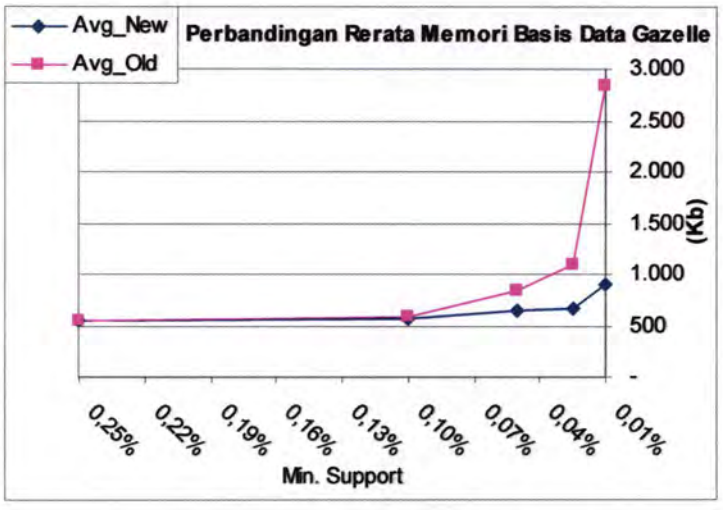
Basis data yang bertipe bimodal adalah Gazelle dan T10. Hasil uji coba basis data Gazelle sebagaiman terlihat pada Tabel 4.8, Gambar 4.20, Gambar 4.21 dan Gambar 4.22. Hasil uji coba basis data T10 sebagaimana terlihat pada Tabel 4.9, Gambar 4.23, Gambar 4.24 dan Gambar 4.25.

Tabel 4.9 Hasil Uji Coba Basis Data Gazelle

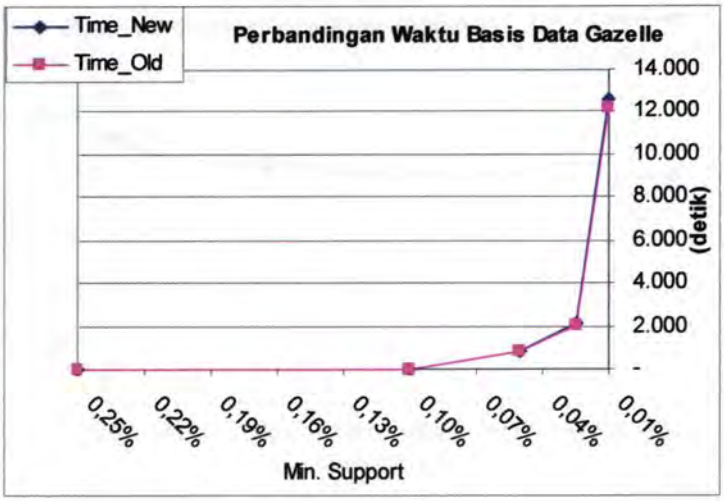
Minsup (%)	Closed	F1	*PRE_SET*	Time (sec)			Avg Memori (Kb)		Efisiensi (%)
				New	Old	%	New	Old	
0,500	201	150	149	11	11	100,00	521	523	0,4
0,250	554	247	246	19	19	100,00	552	553	0,2
0,100	3.974	343	342	35	35	100,00	570	580	1,7
0,050	127.131	374	373	855	817	104,65	648	849	23,7
0,025	266.911	408	407	2.175	2.089	104,12	674	1.089	38,1
0,010	1.240.700	451	450	12.593	12.215	103,09	911	2.852	68,1



Gambar 4.20 Grafik Jumlah *Frequent closed itemsets* Basis Data Gazelle



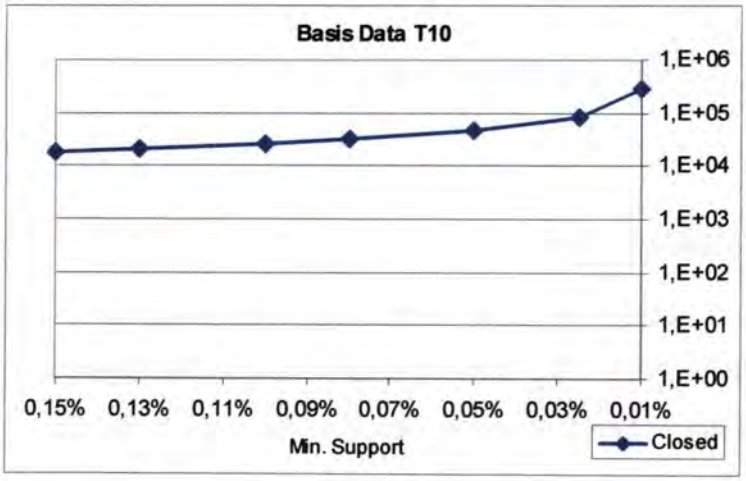
Gambar 4.21 Grafik Perbandingan Rerata Memori Basis Data Gazelle



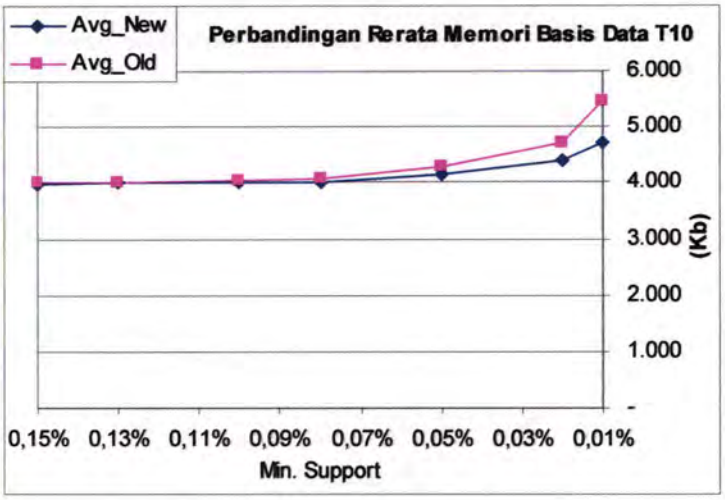
Gambar 4.22 Grafik Perbandingan Waktu Basis Data Gazelle

Tabel 4.10 Hasil Uji Coba Basis Data T10

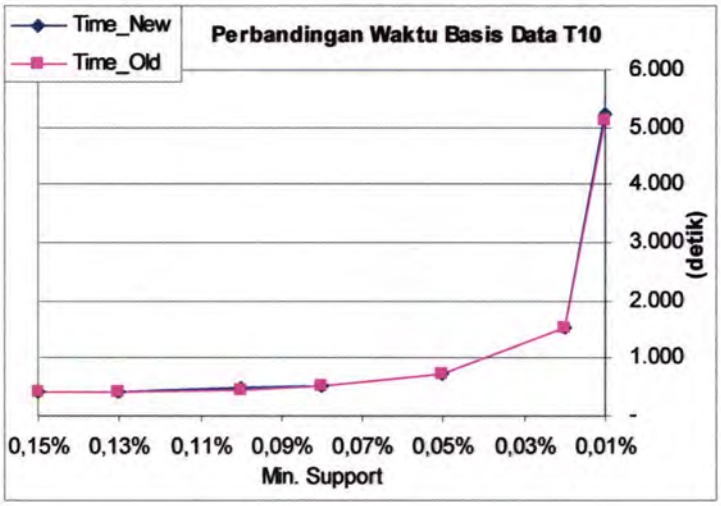
Minsup (%)	Closed	F1	*PRE_SET*	Time (sec)			Avg Memori (Kb)		Efisiensi (%)
				New	Old	%	New	Old	
0,15	18.650	767	766	417	409	101,96	3.992	4.008	0,4
0,13	21.106	776	775	431	425	101,41	3.996	4.021	0,6
0,10	26.574	797	796	471	463	101,73	4.011	4.050	1,0
0,08	32.288	814	813	525	515	101,94	4.026	4.093	1,6
0,05	46.314	839	838	735	727	101,10	4.144	4.279	3,2
0,025	81.401	856	855	1.551	1.527	101,57	4.390	4.715	6,9
0,01	283.397	867	866	5.243	5.141	101,98	4.721	5.458	13,5



Gambar 4.23 Grafik Jumlah *Frequent closed itemsets* Basis Data T10



Gambar 4.24 Grafik Perbandingan Rerata Memori Basis Data T10



Gambar 4.25 Grafik Perbandingan Waktu Basis Data T10

4.6 Analisis Hasil Uji Coba

Sesuai dengan skenario, pelaksanaan dan hasil uji coba sebagaimana telah dijelaskan dalam Sub Bab 4.2 sampai dengan Sub Bab 4.4, maka analisis hasil uji coba dikelompokkan berdasarkan tipe distribusi basis data.

4.6.1 Analisis Hasil Uji Coba Basis Data Simetrik

Pengujian dilakukan pada basis data Chess, Pumsb, Connect dan Pumsb*. Gambar 4.2, Gambar 4.5, Gambar 4.8 dan Gambar 4.11 menunjukkan jumlah *frequent closed itemsets* tiap basis data simetrik dengan berbagai macam nilai minimum *support*. Hasil uji coba menunjukkan bahwa jumlah *frequent closed itemsets* hasil enumarasi tidak tergantung jumlah record basis data. Misalnya basis data Chess dengan jumlah record sebanyak 3196, pada minimum *support* 25% jumlah *frequent closed itemsetsnya* bisa mencapai 10 juta, sedangkan Pumsb* dengan jumlah record 49046 pada minimum *support* yang sama jumlah *frequent closed itemsetsnya* hanya sebesar $\pm$ 42 ribu. Sebagaimana dapat dilihat pada Gambar 4.3, Gambar 4.6, Gambar 4.9 dan Gambar 4.12, algoritma CHARM_NEW menggunakan memori yang lebih efisien dibandingkan memori yang digunakan algoritma CHARM terutama untuk nilai minimum *support* yang makin kecil. Seperti ditunjukkan pada Tabel 4.3, basis data Chess pada minimum *support* terbesar efisiensi memori algoritma CHARM_NEW sebesar 9,02%, sedangkan pada minimum *support* terkecil efisensinya sebesar 90,53%. Basis data Pumsb sebagaimana terlihat pada Tabel 4.4, efisiensi memori algoritma CHARM_NEW sebesar 1,6% pada minimum *support* terbesar, sedangkan pada minimum *support* terkecil efisiensinya sebesar 55,4%. Tabel 4.6 menunjukkan hasil uji coba basis data Connect, efisiensi memori algoritma CHARM_NEW sebesar 0,2% pada minimum *support* terbesar, sedangkan pada minimum *support* terkecil efisiensinya sebesar 58,1%. Sedangkan basis data Pumsb* sebagaimana ditunjukkan pada Tabel 4.5 efisiensi memori algoritma CHARM_NEW sebesar 0,4% pada minimum *support* terbesar, sedangkan pada minimum *support* terkecil efisiensinya sebesar 23,6%. Efisiensi memori terbesar algoritma CHARM_NEW ditunjukkan basis data Chess, sedangkan efisiensi memori terkecil ditunjukkan basis data Pumsb*. Jumlah record basis data Chess hanya sebanyak 3.196, tetapi jumlah *frequent closed itemsets* hasil enumerasi lebih dari 10 juta pada minimum *support* 25%. Sedangkan jumlah record basis data Pumsb* sebanyak 49.046, tetapi *frequent closed itemsets* pada minimum *support* terkecil hanya sebanyak $\pm$ 1,5 juta. Penggunaan memori algoritma CHARM dipengaruhi jumlah *frequent closed*

*itemsets*nya, sehingga apabila *frequent closed itemsets* semakin besar maka memori yang dibutuhkan juga semakin besar. Berbeda dengan algoritma CHARM_NEW dimana penggunaan memori sangat dipengaruhi oleh jumlah *PRE_SET*. Pada minimum *support* yang makin kecil *frequent closed itemsets* bertambah hampir mendekati fungsi eksponensial, sedangkan *PRE_SET* bertambah mendekati fungsi linier sehingga algoritma CHARM_NEW lebih efisien penggunaan memorinya dibandingkan algoritma CHARM terutama untuk nilai minimum *support* yang makin kecil. Hasil uji coba basis data simetrik menunjukkan bahwa efisiensi memori algoritma CHARM_NEW semakin besar apabila jumlah record basis data semakin kecil dengan jumlah *frequent closed itemsets* semakin besar.

Penggunaan waktu proses enumerasi *frequent closed itemsets* pada basis data simetrik dapat dilihat pada Gambar 4.4, Gambar 4.7, Gambar 4.10 dan Gambar 4.11. Kecuali basis data Pumsb*, pada minimum *support* terbesar waktu proses algoritma CHARM_NEW maksimum sebesar 108,70% dari waktu proses algoritma CHARM, sedangkan pada minimum *support* terkecil waktu proses algoritma CHARM_NEW maksimum sebesar 86,08% dari waktu proses algoritma CHARM. Kinerja algoritma CHARM_NEW terutama untuk minimum *support* yang makin kecil cenderung makin baik. Untuk basis data Chess, Connect dan Pumsb waktu proses algoritma CHARM_NEW lebih baik bila dibandingkan dengan waktu proses algoritma CHARM terutama untuk nilai minimum *support* terkecil yang digunakan dalam pengujian. Hal ini disebabkan pada minimum *support* tersebut jumlah *frequent closed itemsets* yang ditemukan adalah lebih dari 5 juta, sehingga algoritma CHARM memerlukan *virtual* memori yang jauh lebih besar bila dibandingkan algoritma CHARM_NEW. Dengan besarnya kebutuhan *virtual memori* pada algoritma CHARM menyebabkan proses pemeriksaan duplikasi menjadi lebih lama dibandingkan dengan algoritma CHARM_NEW. Waktu yang digunakan algoritma CHARM_NEW pada basis data Pumsb* sedikit lebih besar bila dibandingkan dengan algoritma CHARM, hal ini dikarenakan *frequent closed itemsets* yang terbentuk pada minimum *support* terkecil pengujian hanya berjumlah maksimum $\pm$ 1,5 juta. Perbedaan penggunaan *virtual* memori algoritma CHARM tidak terlalu besar dibandingkan dengan penggunaan *virtual*

memori algoritma CHARM_NEW sehingga pemeriksaan duplikasinya lebih baik bila dibandingkan dengan algoritma CHARM_NEW.

4.6.2 Analisis Hasil Uji Coba Basis Data Bimodal

Dua basis data dengan distribusi bimodal yaitu Mushroom dan T40, jumlah *frequent closed itemsets* basis data bimodal seperti ditunjukkan pada Gambar 4.14 dan Gambar 4.17. Seperti ditunjukkan pada Gambar 4.15 dan Gambar 4.18, algoritma CHARM_NEW menggunakan memori yang lebih efisien apabila dibandingkan algoritma CHARM. Seperti ditunjukkan pada Tabel 4.6, basis data Mushroom pada minimum *support* terbesar efisiensi memori algoritma CHARM_NEW sebesar 1,4%, sedangkan pada minimum *support* terkecil efisensinya sebesar 28%. Sedangkan basis data T40 sebagaimana terlihat pada Tabel 4.7, efisiensi memori algoritma CHARM_NEW sebesar 0,1% pada minimum *support* terbesar, sedangkan pada minimum *support* terkecil efisiensinya sebesar 51,7%. Basis data Mushroom dengan jumlah record 8.124 pada minimum *support* terkecil efisensi memori algoritma CHARM_NEW hanya sebesar 28%. Hal ini disebabkan jumlah *frequent closed itemsets*nya hanya sebesar 164.520, sehingga memori yang digunakan algoritma CHARM yang berbasis untuk *frequent closed itemsets* perbedaanya tidak terlalu besar bila dibandingkan dengan memori yang digunakan oleh algoritma CHARM_NEW yang berbasis *PRE_SET*. Sedangkan untuk basis data T40, efisiensi memori algoritma CHARM_NEW pada minimum *support* terkecil sebesar 51,7% karena jumlah *frequent closed itemsets*nya sebesar 7 juta. Jumlah *frequent closed itemsets* yang besar menyebabkan penggunaan memori algoritma CHARM juga besar karena pemeriksaan duplikasinya berdasarkan pada jumlah *frequent closed itemsets*. Pada kondisi ini memori yang digunakan algoritma CHARM_NEW perubahannya tidak siginifikan karena algoritma CHARM_NEW berbasis *PRE_SET*. Hasil uji coba pada basis data bimodal menunjukkan bahwa algoritma CHARM_NEW lebih efisien penggunaan memorinya dibandingkan dengan algoritma CHARM terutama untuk nilai minimum *support* yang makin kecil.

Waktu proses kedua algoritma ditunjukkan pada Gambar 4.16 dan Gambar 4.19. Waktu proses algoritma CHARM_NEW secara keseluruhan perbedaannya tidak terlalu signifikan. Waktu proses algoritma CHARM_NEW basis data T40 lebih baik apabila dibandingkan dengan waktu proses algoritma CHARM untuk nilai minimum support 0,2%. Hal ini disebabkan pada minimum *support* tersebut jumlah *frequent closed itemsets* yang ditemukan adalah lebih dari 7 juta. Algoritma CHARM harus menyimpan *frequent closed itemsets* sebelumnya sehingga algoritma CHARM memerlukan *virtual* memori yang lebih besar apabila dibandingkan dengan algoritma CHARM_NEW. Algoritma CHARM hanya menyimpan *PRE_SET* sebelumnya, sehingga jumlah *frequent itemset closed* yang makin besar tidak mempengaruhi secara siginifikan waktu prosesnya. Proses enumerasi dengan *virtual* memori yang besar menyebabkan waktu proses algoritma CHARM lebih lama bila dibandingkan dengan algoritma CHARM_NEW.

4.6.3 Analisis Hasil Uji Coba Basis Data Right Skewed

Basis data Gazelle dan T10, mempunyai sejumlah besar *closed pattern* sangat pendek. Jumlah *frequent closed itemsets* yang terbentuk seperti ditunjukkan pada Gambar 4.20 dan Gambar 4.23. Seperti ditunjukkan pada Tabel 4.8, jumlah *frequent closed itemsets* basis data Gazelle sebanyak $\pm$ 1,2 juta pada minimum *support* 0,01%. Efisiensi memori algoritma CHARM_NEW sebesar 68,1%, karena *closed pattern* basis data Gazelle hanya ditemukan pada nilai minimum *support* yang yang sangat kecil ($\leq$ 0,5%) sehingga *tidset* untuk semua *closed pattern* akan kecil ($\leq$ 298). Kondisi ini menyebabkan memori yang digunakan algoritma CHARM_NEW menjadi lebih efisien bila dibandingkan memori yang dipergunakan oleh algoritma CHARM yang berbasis *frequent closed itemsets* yang ditemukan. Basis data T10 seperti ditunjukkan pada Tabel 4.9 efisiensi yang bisa dicapai jauh lebih kecil bila dibandingkan efisiensi yang bisa dicapai basis data Gazelle. Efisiensi memori algoritma CHARM_NEW hanya sebesar 13,5% untuk nilai minimum *support* 0,01%. Hal ini dikarenakan pada nilai minimum *support* 0,01% jumlah *frequent closed itemsets*nya hanya sebesar

283 ribu. Jumlah ini jauh lebih kecil apabila dibandingkan jumlah *frequent closed itemsets* basis data Gazelle pada minimum *support* terkecil. Hasil uji coba basis data *right skewed* seperti terlihat pada Gambar 4.20 dan Gambar 4.24 menunjukkan bahwa algoritma CHARM_NEW lebih efisien penggunaan memorinya apabila dibandingkan algoritma CHARM terutama untuk nilai minimum *support* yang makin kecil.

Seperti ditunjukkan pada Gambar 4.22 dan Gambar 4.25 waktu proses enumerasi *frequent closed itemsets* algoritma CHARM_NEW perbedaannya tidak terlalu signifikan apabila dibandingkan dengan algoritma CHARM untuk berbagai macam nilai minimum *support*. Waktu proses algoritma CHARM_NEW tidak ada yang lebih baik apabila dibandingkan dengan waktu proses algoritma CHARM. Hal ini disebabkan untuk semua nilai minimum *support* jumlah *frequent closed itemsets* yang ditemukan maksimum hanya sebesar 1,3 juta. Perbedaan penggunaan *virtua*l memori antara kedua algoritma tidak terlalu signifikan sehingga proses enumerasi lebih banyak dilakukan memori utama. Dengan demikian waktu proses algoritma CHARM sedikit lebih baik apabila dibandingkan dengan waktu proses algoritma CHARM_NEW.

Hasil uji coba keseluruhan menunjukkan bahwa algoritma CHARM_NEW lebih efisien penggunaan memorinya dibandingkan dengan algoritma CHARM terutama untuk *minimum support* yang makin kecil. Efisiensi memori tidak dipengaruhi oleh tipe distribusi basis data tetapi efisiensi memori dipengaruhi karakteristik basis datanya. Semakin kecil jumlah record basis data dengan jumlah *frequent closed itemsets* hasil enumerasi makin besar maka efisiensi memorinya semakin besar. Penggunaan memori terbesar algoritma CHARM_NEW apabila kombinasi semua 1-*itemset frequent* (F1) mempunyai properti 2 atau 4, sehingga tidak ada penghapusan F1 dalam proses enumerasi. *PRE_SET* yang digunakan sebanding dengan | F1 - 1 | x l dengan l adalah jumlah rerata *tid*. Algoritma CHARM menyimpan semua *closed set* yang telah dienumarasi sebelumnya dalam tabel *hash* untuk melakukan pemeriksaan duplikasi. *Frequent closed itemsets* (C) yang tumbuh secara eksponensial menyebabkan kinerja *hash* akan turun. Seperti ditunjukkan pada Tabel 4.3 pertumbuhan C untuk basis data Pumsb berkembang secara eksponensial

sedangkan *PRE_SET* berkembang secara linier untuk nilai *minimum support* yang makin kecil. Kebutuhan memori algoritma CHARM akan bergerak secara eksponensial, sedangkan kebutuhan memori yang dipergunakan algoritma CHARM_NEW berkembang secara linier seperti ditunjukkan pada Gambar 4.6.

Berdasarkan analisis kompleksitas pemeriksaan duplikasi sebagaimana telah dijelaskan pada Sub Bab 3.2, kompleksitas pemeriksaan duplikasi algoritma CHARM_NEW lebih besar apabila dibandingkan dengan kompleksitas algoritma CHARM. Meskipun demikian dari hasil uji coba seperti terlihat pada Gambar 4.7 menunjukkan bahwa waktu proses algoritma CHARM_NEW menunjukkan kinerja yang makin baik apabila dibandingkan algoritma CHARM terutama untuk nilai minimum *support* yang makin kecil. Seperti ditunjukkan pada Tabel 4.3, *PRE_SET* berkembang secara linier sedangkan *frequent closed itemsets* (C) berkembang secara eksponensial, sehingga algoritma CHARM_NEW yang berbasis *PRE_SET* menggunakan *virtual* memori yang lebih kecil apabila dibandingkan algoritma CHARM yang berbasis C. Tetapi apabila jumlah C hasil enumerasi lebih kecil dari 5 juta maka kecepatan proses algoritma CHARM lebih cepat dibandingkan dengan algoritma CHARM_NEW, karena penggunaan *virtual* memori algoritma CHARM tidak berbeda jauh dengan algoritma CHARM_NEW, sehingga proses pemeriksaan duplikasi algoritma CHARM lebih banyak dilakukan di memori utama.

BAB 5

KESIMPULAN DAN PENGEMBANGAN LEBIH LANJUT

BAB 5
KESIMPULAN DAN PENGEMBANGAN LEBIH LANJUT

5.1 Kesimpulan

Berdasarkan hasil uji coba dan analisis hasil, maka dapat disimpulkan beberapa hal seperti berikut :

1. Algoritma CHARM_NEW yang dikembangkan dalam penelitian ini menunjukkan penggunaan memori yang lebih efisien dibandingkan dengan algoritma CHARM, terutama untuk nilai minimum *support* yang rendah.
2. Efisiensi penggunaan memori dipengaruhi oleh karakteristik basis data yang diujicobakan. Untuk ini, semakin kecil jumlah record basis data dengan jumlah *frequent closed itemsets* hasil enumerasi yang semakin besar memberikan hasil efisiensi penggunaan memori yang semakin besar.
3. Dari segi kecepatan proses, algoritma CHARM_NEW tidak memberikan indikasi yang kuat untuk dikatakan lebih cepat dibandingkan dengan algoritma CHARM.

5.2 Pengembangan Lebih Lanjut

Seperti dijelaskan sebelumnya bahwa kecepatan proses dari algoritma CHARM_NEW tidak memberikan indikasi yang kuat untuk dikatakan lebih cepat dibandingkan dengan algoritma CHARM. Hal ini dikarenakan perbaikan algoritma CHARM hanya difokuskan pada tujuan untuk menghemat penggunaan memori. Hal ini memberikan peluang untuk dilakukan pengembangan lebih lanjut terhadap algoritma CHARM_NEW dari segi kecepatan proses. Salah satu yang dapat diperbaiki terkait dengan prosedur yang dipakai untuk melakukan pemeriksaan duplikasi kandidat *frequent closed itemsets*, yang dalam algoritma CHARM_NEW dilakukan dengan penggabungan sederhana antara metode pemeriksaan duplikasi yang digunakan oleh CHARM dan metode *order preserving* yang digunakan dalam algoritma DCI_CLOSED.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

Levitin A., (2003), *Introduction to The Design & Analysis of Algorithms*, Addison Wesley, New York.

Lucchesse C., Orlando S. and Perego R., (2005), "Fast and Memory Efficient Mining of Frequent Closed Itemsets ," *IEEE Trans. On Knowledge and Data Eng.*

Lucchesse C., Orlando S. and Perego R., (2002), "Adaptive and resource aware mining of frequent sets ," *In Proc. Of the IEEE Int. Conference on Data Mining,* Maebashi, Japan.

Lucchesse C., Orlando S. and Perego R., (2003), "Kdci: a multi-strategy algorithm for mining frequent sets ," *In Proc. Of the 2003 Workshop on Frequent Itemset Mining Implementations,* Melbourne, Florida, USA.

Pang Ning Tan, Steinbach M. and Kumar V., (2006), *Introduction to Data Mining*, Addison Wesley, New York.

Zaki M.J. and Hsiao C.-J., (2005), "Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure ," *IEEE Trans. On Knowledge and Data Eng.*, Vol. 17, no. 4, pp. 462-478.

Zaki M.J. and Gouda K., (2003), "Fast Vertical Mining Using Diffsets ," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining.*