ANALISIS KINERJA ALGORITMA FOLD-GROWTH DAN FP-GROWTH PADA PENGGALIAN POLA ASOSIASI

Rully Soelaiman, Ni Made Arini WP
Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS), Surabaya, 60111
E-mail: rully@its-sby.edu

ABSTRAKSI

Penggalian data sangat penting untuk proses pengumpulan data dan untuk penyimpanan data. Di sisi lain, data mentah yang tersedia sangatlah besar sehingga analisis manual tidak lagi memungkinkan untuk menangani masalah data. Permasalahan utama dari penggalian data adalah munculnya beberapa struktur data baru yang dimaksudkan untuk memperbaiki efisiensi dari penggalian data. Dan tidak satu pun dari struktur data tersebut yang mampu menangani semua masalah penggalian data.

Fold-Growth merupakan salah satu dari metode penggalian pola asosiasi dengan menggunakan struktur data SOTrieIT (Support Ordered-Trie Itemset) dalam proses penggalian itemset yang frequent. SOTrieIT adalah sebuah struktur data yang dapat melakukan ekstraksi 1-itemset dan 2-itemset dari semua transaksi dalam basis data. Dengan menggunakan basis data transaksi yang terdiri dari kode transaksi, dan kode dari barang yang di beli, algoritma ini akan diproses untuk menghasilkan pola asosiasi. Pada penelitian ini, algoritma FOLD-growth akan dibagi dalam empat tahapan utama yaitu, tahapan penggalian 1-itemset frequent dan 2-itemset frequent, tahap pemangkasan item-item yang tidak frequent, membangun FP-tree, dan tahapan penggalian semua itemset frequent.

Berdasarkan uji coba, yang melibatkan dataset sintetik, dapat disimpulkan bahwa secara umum durasi eksekusi dan utilisasi memori Fold-Growth lebih kecil dibandingkan dengan FP-Growth.

Kata Kunci: *FP-growth, SOTrieIT, Fold-Growth, pola asosiasi, penggalian data, struktur data.*

1. PENDAHULUAN

Penggalian data merupakan upaya untuk melakukan ekstraksi pola informasi atau pengetahuan yang menarik dari sejumlah besar data. Pola informasi atau pengetahuan dapat dikatakan menarik apabila pola atau pengetahuan tersebut merupakan pola yang *non-trivial*, implisit, belum diketahui sebelumnya, dan bermanfaat.

Dalam penggalian dan menganalisis pola asosiasi akan ditemukan atribut-atribut yang menunjukkan kondisi dimana atribut-atribut tersebut sering muncul bersamaan dalam suatu data yang diberikan. Penggalian pola asosiasi ini biasanya sering digunakan dalam menganalisis data transaksi. Ada beberapa algoritma yang digunakan dalam upaya untuk memperbaiki kinerja penggalian pola asosiasi tapi, algoritma-algoritma tersebut hanya mampu menangani kondisi tertentu saja dan memiliki kekurangan dalam kondisi lainnya. Sedangkan, dalam penggalian data efektivitas dan efisiensi sangatlah diperlukan.

Dalam pemelitian ini akan dilakukan analisis kinerja algoritma *FP-growth* dan algoritma *Fold-growth* yang menggunakan struktur data *SOTrieIT* pada penggalian pola transaksi. Definisi permasalahan, algoritma *Fold-growth* dan *FP-growth*, serta struktur data *SOTrieIT* akan dijelaskan di bagian 2. Bagian 3 akan menjelaskan hasil beberapa uji coba dan analisis eksperimen. Bagian 4 akan menyajikan simpulan dan kemungkinan pengembangan.

2. METODE

Di bagian ini algoritma *FOLD-growth* dan *FP-growth* akan dijabarkan, serta penjelasan mengenai struktur data *SOTrieIT*.

2.1 FP-growth

Berdasarkan [3], penggalian itemset yang frequent dengan menggunakan algoritma FP-Growth akan dilakukan dengan cara membangkitkan struktur data *Tree* atau disebut dengan *FP-Tree*. Metode FP-growth melibatkan tiga tahapan utama yaitu, (a) tahap pembangkitan conditional pattern base, (b) tahap pembangkitan conditional FP-tree, dan (c) tahap pencarian frequent itemset. Algoritma FP-growth dapat dilihat pada Gambar 1.

a) Tahap Pembangkitan Conditional Pattern Base

Conditional pattern base merupakan subdatabase yang berisikan *prefix path* (himpunan item terurut yang mengawali k-itemsets), dan *suffix pattern* (k-itemsets). Misalnya, sebuah itemset yang telah terurut berdasarkan support descending order $\{I_6, I_3, I_1, I_{13}, I_{16}\}$, apabila I_{16} merupakan suffix pattern maka, I_6, I_3, I_1, I_{13} adalah prefix path. Pembangkitan conditional pattern base didapatkan melalui FP-tree yang telah dibangun berdasarkan sebuah basis data transaksi.

b) Tahap Pembangkitan Conditional FP-tree

Pada tahap ini, support count untuk tiap item pada setiap conditional pattern base akan dijumlahkan, kemudian item yang memiliki jumlah support count lebih besar sama dengan min_sup akan dibangkitkan menjadi sebuah tree yang disebut dengan *conditional FP-tree*.

c) Tahap Pencarian Frequent Itemset

Pada tahap ini, apabila *Conditional Fp-tree* merupakan *single path*, maka akan didapatkan frequent itemsets dengan melakukan kombinasi item untuk setiap *Conditional Fp-tree.* Jika bukan single path maka, akan dilakukan pembangkitan FP-Growth secara rekursif.

```
Input: FP-tree Tree
Output: R_t sekumpulan lengkap pola
        frequent
Method: FP-growth(Tree,null)
Procedure: FP-growth(Tree,α)
{
```

```
01:if Tree mengandung single path P;
02:then untuk tiap kombinasi (dinotasikan
   β) dari node-node dalam path P do
03:bangkitkan pola β α dengan support =
   minimum support dari       node-node
   dalam β;
04:else untuk tiap ai dalam header dari
   Tree do {
05:bangkitkan pola
06:bangun β= ai α dengan
   support = ai .support
07:if Tree β=Ø
08:then panggil FP-growth(Tree,β) }
}
```

Gambar 1. Algoritma FP-growth

2.2 Fold-growth

Algoritma FOLD-Growth merupakan hasil gabungan dari algoritma FOLDARM dan FP-Growth. Pada algoritma FOLD-growth ini, diharapkan dapat menggabungkan keuntungan dari algoritma FOLDARM yang memiliki kinerja cepat pada saat ukuran itemset frequent maksimum (k_{max}) adalah kecil atau $k_{max} \leq 10$ dengan keuntungan dari algoritma FP-Growth yang memiliki kinerja yang cepat pada saat $k_{max} > 10$. Algoritma FOLD-growth ditunjukkan pada gambar 2.

Dalam algoritma FOLD-growth ini dibagi menjadi empat tahapan utama yaitu (a) penggalian L_1 dan L_2 dengan menggunakan SOTrieIT, (b) pemangkasan item-item yang tidak frequent, (c) membangun FP-tree menggunakan transaksi-transaksi yang telah dipangkas dan, (d) penggalian itemset frequent dengan algoritma FP-growth.

```
1: Menggunakan SOTrie untuk menemukan L1
   dan L2 dengan cepat
2: if L1 = 0 ∨ L2 = 0 then
3:    algoritma dihentikan
4:  end if
5:  for transaction T ∈ D do
6:  Hilangkan item yang tidak memberikan
   kontribusi pada Lk dimana k > 2,
   menggunakan L1 dan L2
7: Urutkan item berdasarkan support terbesar
8: Bangun/Ubah FP-tree dengan T yang telah
   dipangkas dan diurutkan
9: end for
10: Jalankan algoritma FP-growth pada FP-
   tree yang dibangun
```

Gambar 2. Algoritma Fold-growth

Tahap Penggalian L_1 dan L_2 dengan Menggunakan SOTrieIT

Pada tahap ini, dilakukan scan basis data sebanyak satu kali untuk membaca transaksi-transaksi yang ada dalam basis data. Untuk setiap transaksi akan dibangkitkan semua kemungkinan-kemungkinan *1-itemset* dan *2-itemset* yang kemudian dicatat dalam SOTrieIT.

Tahap Pemangkasan Item-item yang Tidak Frequent

Dalam tahap ini, akan dilakukan pemangkasan pada setiap transaksi yang ada dalam basis data dengan menggunakan L_1 dan L_2. Untuk setiap transaksi *T*, pada itemset L_k yang terdapat dalam transaksi tersebut dimana panjang *k* lebih dari 2, akan dilakukan pengecekan dengan menggunakan L_1 dan L_2. Sehingga, untuk item-item yang dianggap tidak frequent akan dilakukan pemangkasan. Sebuah item dikatakan tidak frequent apabila nilai support count-nya kurang dari batas minimum support yang telah ditentukan oleh pengguna. Setelah dilakukan pemangkasan terhadap transaksi *T*, maka item-item yang terdapat dalam transaksi tersebut akan diurutkan berdasarkan nilai support count yang paling besar. Dengan menggunakan L_1 dan L_2 yang didapatkan melalui SOTrieIT, maka akan dihasilkan *Ordered Frequent Items* yang telah dipangkas.

Tahap Pembangunan FP-tree

Pada tahapan ini, akan dilakukan pembangunan FP-Tree dengan menggunakan data transaksi *T* yang telah dipangkas dan diurutkan berdasarkan nilai support count. Dengan perolehan *Frequent Items* setelah dipangkas dan diurutkan, maka akan dibangun FP-Tree.

Tahap Penggalian Itemset Frequent

Setelah tahap pembangunan FP-tree selesai, dilanjutkan dengan tahap penggalian itemset frequent dengan menggunakan algoritma FP-growth pada FP-tree tersebut.

Tabel 1. Basis data transaksi

TID	*Item*
1	AC
2	BC
3	ABC
4	ABCD

Himpunan pola asosiasi yang ditemukan berdasarkan basis data transaksi pada tabel 1 dan dengan minimum support dan minimum confidence adalah 50% dalam proses penggalian rekursif di atas, dapat dilihat pada Gambar 3.

```
Pola asosiasi {1} → {3}
       Support: 0.75 Confidence: 1
Pola asosiasi {3} → {1}
       Support: 0.75 Confidence: 0.75
Pola asosiasi {2} → {3}
       Support: 0.75 Confidence: 1
Pola asosiasi {3} → {2}
       Support: 0.75 Confidence: 0.75
```

Gambar 3. Pola Asosiasi berdasarkan tabel 1

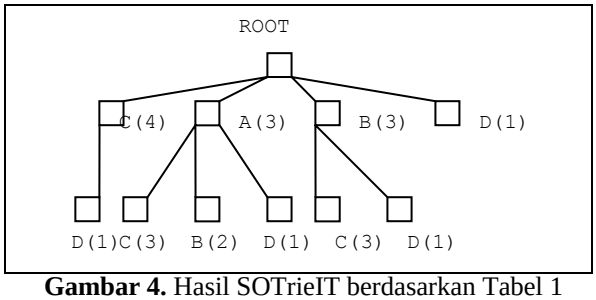
2.3 Struktur Data SOTrieIT

SOTrieIT memiliki dua tingkatan dari node-node *Tree* yang menyatakan bahwa tiap node *w* memiliki sebuah label *l* yang dinyatakan sebagai item dan sebuah notasi *j* yang menyimpan nilai support count yang berhubungan. Setiap node pohon terhubung pada beberapa item yang terdapat dalam itemset *I* (dinotasikan dengan $a_i \in I$) maka, untuk w_i mengacu pada node yang memiliki hubungan dengan $a_i \in I$.

Himpunan SOTrieIT dimungkinkan memiliki *parent node* yang berbeda-beda seperti w_1 , w_2 ,..., w_N, yang dibangun dari sebuah basis data untuk menyimpan *support count* dari semua 1-itemset dan 2-itemset. Maka, digunakan node khusus yang dinamakan **ROOT** untuk menghubungkan semua SOTrie bersama-sama. Hasil SOTrieIT berdasarkan

 ISSN: 1907-5022

basis data transaksi pada tabel 1 dapat dilihat pada Gambar 4.



Gambar 4. Hasil SOTrieIT berdasarkan Tabel 1

3. UJI COBA

Uji coba ini meliputi uji coba kebenaran dan uji coba kinerja. Semua eksperimen dilakukan pada sebuah PC dengan prosesor Intel ® Pentium ® 4 CPU 2.80 GHz, memori 512 MB RAM, VGA Card NVIDIA GeForce4 MX 440 dengan AGP8X. Sistem operasi yang digunakan adalah Microsoft Windows 2000 Professional (5.0, Build 2195). Kedua algoritma diimplementasikan menggunakan bahasa pemprograman Java.

Uji Coba Kebenaran

Uji coba kebenaran dilakukan untuk menguji validitas sistem aplikasi. Basis data transaksi pada Tabel 1 digunakan untuk uji coba ini. Setelah dilakukan uji coba ternyata pola yang dihasilkan oleh kedua algoritma sama dengan pola asosiasi dalam gambar 3 dan paper [4].

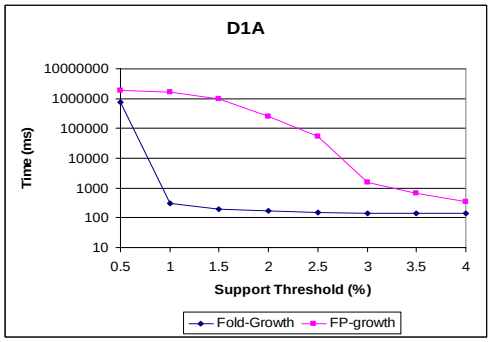
Uji Coba Kinerja

Uji coba kinerja dilakukan untuk menguji kehandalan kinerja algoritma *Fold-growth* dibandingkan *FP-growth* pada sejumlah basis data transaksi sintetik dari berbagai ukuran dan distribusi data. Pengujian ini dilakukan berdasarkan dari segi kecepatan, jumlah node yang dibangun.

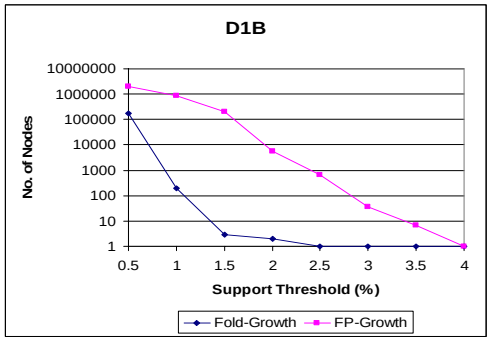
Pengujian pertama dilakukan terhadap himpunan data T25I20N10KD100K, yang terdiri dari 100.000 transaksi dengan 10.000 item yang berbeda. Rata-rata jumlah item dalam sebuah transaksi adalah 25 dan rata-rata jumlah panjang *Large Itemset* adalah 20.

Gambar 5 menunjukkan durasi eksekusi dari alogritma Fold-growth dan FP-growth untuk dukungan minimum 0.5% hingga 4%. Berdasarkan grafik tersebut, *Fold-growth* selalu berjalan lebih cepat daripada *FP-growth*.

Gambar 6 menunjukkan jumlah node yang dihasilkan dari kedua algoritma untuk dukungan minimum 0.5% hingga 4%. Berdasarkan grafik tersebut, dapat dilihat bahwa algoritma Fold-growth menghasilkan node lebih sedikit dibandingkan FP-growth.



Gambar 5. Durasi eksekusi T25I20N10KD100K (min_support 0.5%-4%)



Gambar 6. Jumlah node T25I20N10KD100K (min_support 0.5%-4%)

Hal ini dikarenakan algoritma Fold-growth telah melalui proses pemangkasan terlebih dahulu dan menjadi lebih efisien dalam hal penyimpanan data.

Pengujian kedua dilakukan terhadap himpunan data *T25I20N75D1K*, yang terdiri dari 1000 transaksi dengan 75 item yang berbeda. Rata-rata jumlah item dalam sebuah transaksi adalah 25 dan rata-rata jumlah panjang *Large Itemset* adalah 20.

Gambar 7 menunjukkan durasi eksekusi dari kedua algoritma untuk Dataset *T25I20N75D1K* untuk minimum support 25% hingga 50%. Berdasarkan grafik tersebut, *Fold-growth* selalu berjalan lebih cepat daripada *FP-growth*.

Gambar 8 menunjukkan jumlah node yang dihasilkan dari kedua algoritma untuk data set *T25I20N75D1K* dengan minimum support 25% hingga 50%. Berdasarkan grafik tersebut, dapat dilihat bahwa algoritma Fold-growth menghasilkan node lebih sedikit dibandingkan FP-growth. Hal ini dikarenakan algoritma Fold-growth telah melalui proses pemangkasan terlebih dahulu dan menjadi lebih efisien dalam hal penyimpanan data.

Pengujian ketiga dilakukan terhadap himpunan data *T25I20N75D100*, yang terdiri dari 100 transaksi dengan 75 item yang berbeda. Rata-rata jumlah item dalam sebuah transaksi adalah 25 dan rata-rata jumlah jumlah panjang *Large Itemset* yang ditemukan adalah 20.

Gambar 9 menunjukkan durasi eksekusi dari kedua algoritma untuk Dataset *T25I20N75D100* untuk minimum support 35% hingga 80%. Berdasarkan grafik tersebut, *Fold-growth* selalu berjalan lebih cepat daripada *FP-growth*.

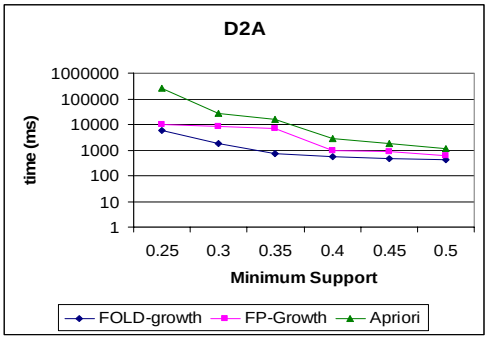
Gambar 10 menunjukkan jumlah node yang dihasilkan dari kedua algoritma untuk data set

ISSN: 1907-5022

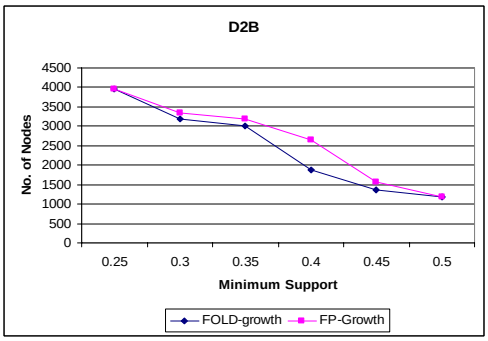
T25I20N75D100 dengan minimum support 35% hingga 80%. Berdasarkan grafik tersebut, dapat dilihat bahwa algoritma Fold-growth menghasilkan node lebih sedikit dibandingkan FP-growth.

Pengujian keempat dilakukan terhadap himpunan data T2I2N32KD640K, yang terdiri dari 640.000 transaksi dengan 32.000 item yang berbeda.

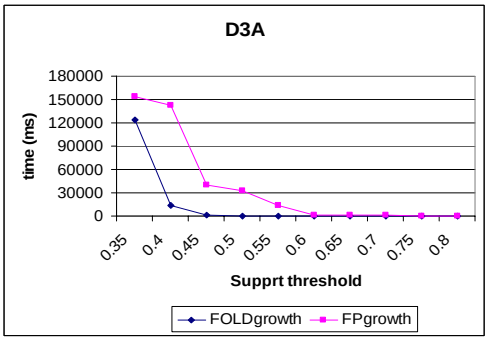
Rata-rata jumlah item dalam sebuah transaksi adalah 2 dan rata-rata jumlah jumlah panjang *Large Itemset* yang ditemukan adalah 2.



Gambar 7. Durasi eksekusi T25I20N75D1K (min_support 25%-50%)



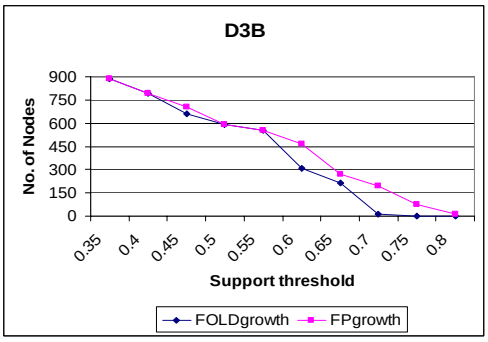
Gambar 8. Jumlah node T25I20N75D1K (min_support 25%-50%)



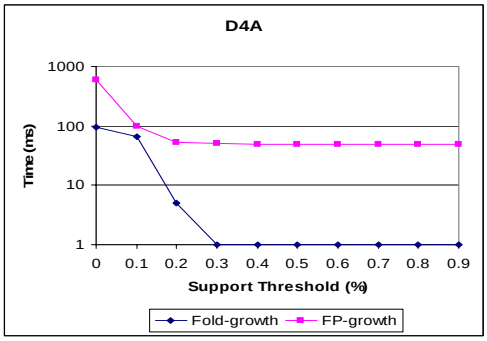
Gambar 9. Durasi eksekusi T25I20N75D100 (min_support 35%-80%)

Gambar 11 menunjukkan durasi eksekusi dari kedua algoritma untuk Dataset T2I2N32KD640K untuk minimum support 25% hingga 50%. Berdasarkan grafik tersebut, *Fold-growth* selalu berjalan lebih cepat daripada *FP-growth*.

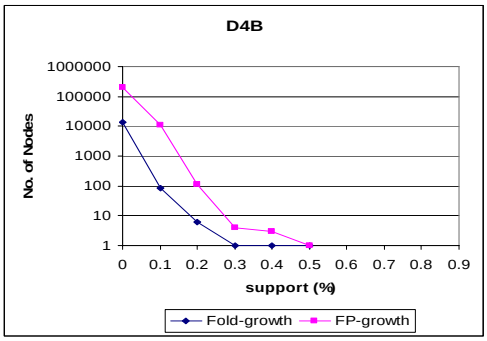
Gambar 12 menunjukkan jumlah node yang dihasilkan dari kedua algoritma untuk data set T2I2N32KD640K dengan minimum support 0% hingga 90%. Berdasarkan grafik tersebut, dapat dilihat bahwa algoritma Fold-growth menghasilkan node lebih sedikit dibandingkan FP-growth.



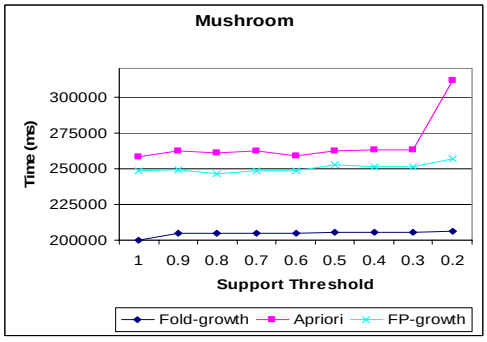
Gambar 10. Jumlah node T25I20N75D100 (min_support 35%-80%)



Gambar 11. Durasi eksekusi T2I2N32KD640K (min_support 0%-90%)



Gambar 12. Jumlah node T2I2N32KD640K (min_support 0%-90%)



Gambar 13. Durasi eksekusi Mushroom (min_support 100%-20%)

Pengujian kelima dilakukan terhadap himpunan data Mushroom, yang merupakan basis data transaksi yang diperoleh melalui *FIMI Repository*. Dataset ini terdiri dari kurang lebih 9000

 ISSN: 1907-5022

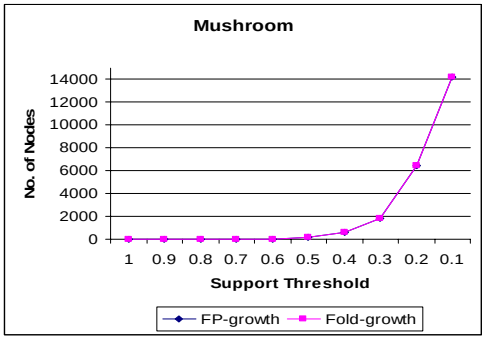
transaksi dengan rata-rata panjang transaksi adalah 25 item.

Gambar 13 menunjukkan durasi eksekusi dari kedua algoritma untuk Dataset *Mushroom* untuk minimum support 20% hingga 100%. Berdasarkan grafik tersebut, *Fold-growth* selalu berjalan lebih cepat daripada *FP-growth*.

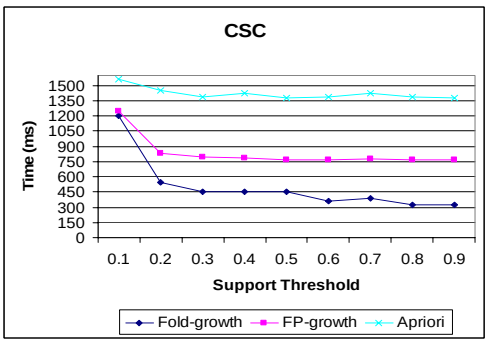
Gambar 14 menunjukkan jumlah node yang dihasilkan dari kedua algoritma untuk data set *Mushroom* dengan minimum support 20% hingga 100%. Berdasarkan grafik tersebut, dapat dilihat bahwa algoritma Fold-growth menghasilkan node yang sama dengan FP-growth.

Pengujian keenam dilakukan terhadap himpunan data CSC, yang merupakan basis data transaksi yang diperoleh melalui *ARMiner Project*. Dataset ini terdiri dari kurang lebih 300 transaksi dengan rata-rata panjang transaksi adalah 25 item.

Gambar 15 menunjukkan durasi eksekusi dari kedua algoritma untuk Dataset *CSC* untuk minimum support 10% hingga 100%. Berdasarkan grafik tersebut, *Fold-growth* selalu berjalan lebih cepat daripada *FP-growth*.



Gambar 14. Jumlah node Mushroom (min_support 100%-20%)



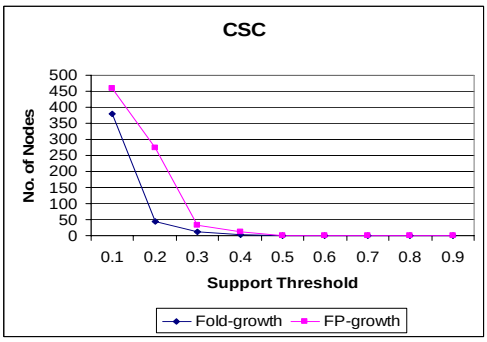
Gambar 15. Durasi eksekusi CSC (min_support 10%-100%)

Gambar 16 menunjukkan jumlah node yang dihasilkan dari kedua algoritma untuk data set *CSC* dengan minimum support 10% hingga 100%. Berdasarkan grafik tersebut, dapat dilihat bahwa algoritma Fold-growth menghasilkan node lebih sedikit dibandingkan FP-growth. Hal ini dikarenakan algoritma Fold-growth telah melalui proses pemangkasan terlebih dahulu dan menjadi lebih efisien dalam hal penyimpanan data.

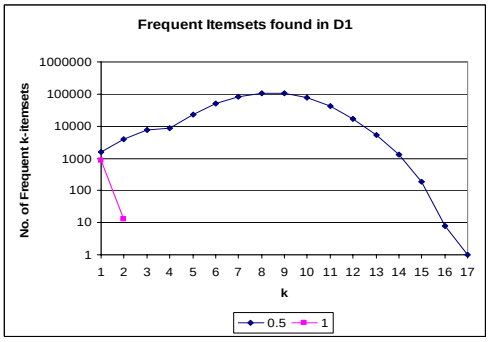
Uji Coba Frequent Itemset Dengan *k* Tertentu

Berdasarkan dengan hasil pengujian dengan menggunakan dataset sintetik mapun dengan menggunakan dataset asli (didapatkan melalui *FIMI Repository*), ternyata dihasilkan sejumlah itemset frequent yang bervariasi. Jumlah dan panjang dari itemset frequent yang dihasilkan bergantung pada ukuran datasets dan juga bergantung pada nilai minimum support yang ditentukan oleh user.

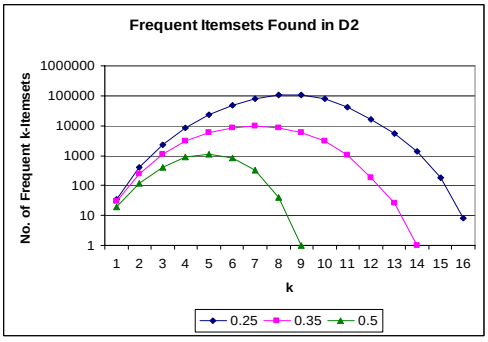
Pengujian pertama, untuk dataset T25I20N10KD100K (D1) dengan nilai minimum support adalah 0.5 % dan 1 % maka, untuk algoritma FOLD-growth, FP-growth, dan Apriori akan ditemukan frequent itemset seperti pada gambar 17.



Gambar 16. Jumlah CSC (min_support 10%-100%)



Gambar 17. Frequent Itemset yang Ditemukan di Dataset D1 (min_support 0.5%, 1%)



Gambar 18. Frequent Itemset yang Ditemukan di Dataset D2 (min_support 25%, 35%, dan 50%)

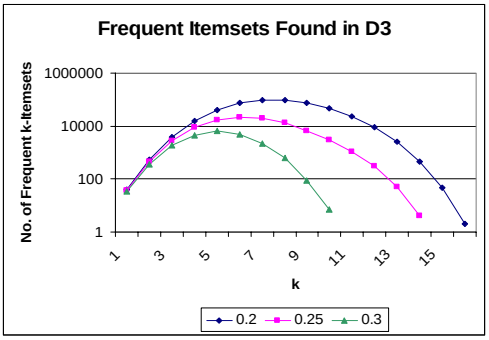
Pengujian kedua, untuk dataset T25I20N75D1K (D2) dengan minimum support adalah 25 %, 35 %, dan 50 % maka, untuk algoritma FOLD-growth, FP-growth, dan Apriori akan ditemukan frequent itemset seperti pada gambar 18.

Pengujian ketiga, untuk dataset T25I20N75D100 (D3) dengan minimum support adalah 20 %, 25 %, dan 30 % maka, untuk algoritma FOLD-growth, FP-growth, dan Apriori akan ditemukan frequent itemset seperti pada gambar 19.

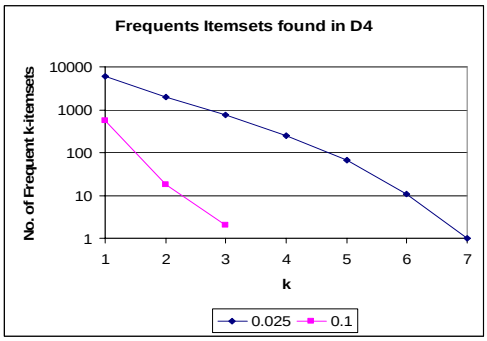
ISSN: 1907-5022

Pengujian keempat, untuk dataset T2I2N32KD640K (D4) dengan minimum support adalah 0.025%, dan 0.1% maka, untuk algoritma FOLD-growth, FP-growth, dan Apriori akan ditemukan frequent itemset seperti pada gambar 20.

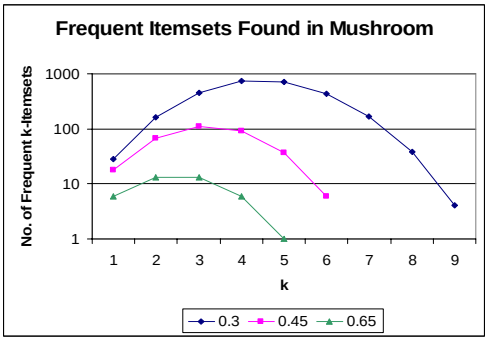
Pengujian kelima, untuk dataset Mushroom dengan minimum support adalah 30%, 45% dan 65% maka, untuk algoritma FOLD-growth, FP-growth, dan Apriori akan ditemukan frequent itemset seperti pada gambar 21.



Gambar 19. Frequent Itemset yang Ditemukan di Dataset D3 (min_support 20 %, 25 %, dan 30 %)



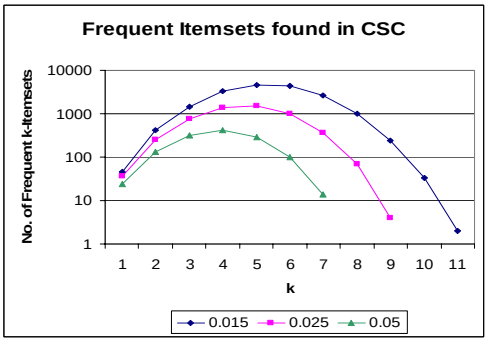
Gambar 20. Frequent Itemset yang Ditemukan di Dataset D4 (min_support 0.0 25 %, dan 0.1 %)



Gambar 21. Frequent Itemset yang Ditemukan di Dataset Mushroom (min_support 30 %, 45%, dan 65 %)

Pengujian keenam, untuk dataset CSC dengan minimum support adalah 1.5 %, 2.5 % dan 5 % maka, untuk algoritma FOLD-growth, FP-growth, dan Apriori akan ditemukan frequent itemset seperti pada gambar 22.

Berdasarkan dengan hasil pengujian dengan menggunakan dataset sintetik mapun dengan menggunakan dataset asli (didapatkan melalui *FIMI Repository* dan *ARMiner Project*), ternyata dihasilkan sejumlah itemset frequent yang bervariasi. Jumlah dan panjang dari itemset frequent yang dihasilkan bergantung pada ukuran datasets dan juga bergantung pada nilai minimum support yang ditentukan oleh user.



Gambar 22. Frequent Itemset yang Ditemukan di Dataset CSC (min_support 1.5%, 2.5% dan 5%)

4. SIMPULAN

Dari hasil uji coba, beberapa simpulan yang bisa diambil adalah:

1. *Fold-growth* berhasil diimplementasikan.
2. Durasi eksekusi, skalabilitas, reliabilitas, dan utilisasi memori *Fold-growth* lebih baik daripada *FP-growth*.
3. Jumlah frequent itemsets mulanya meningkat secara divergen seiring meningkatnya panjang pola yang telah diperoleh, lalu di titik kulminasi jumlah pola akan menurun karena mengalami konvergensi pembentukan pola maksimal.
4. Dukungan minimum berbanding terbalik dengan panjang maksimum pola, dengan jumlah pola untuk panjang pola yang sama, dan dengan total jumlah pola yang diperoleh secara eksponensial.
5. Semakin besar rata-rata jumlah item dalam suatu transaksi, maka semakin besar durasi eksekusi untuk dukungan minimum yang sama dan semakin dini konvergensi untuk pembentukan pola maksimal.

DAFTAR PUSTAKA

[1] Goethals, B., Survey on Frequent Pattern Mining, *HIIT Basic Research Unit, Department of Computer Science, University of Helsinki, Finland,* 2003.

[2] Han, J., Micheline, K., Data Mining: Concepts and Techniques, *Simon Fraser University, Morgan Kaufmann Publisher,* 2000.

[3] Han, J., Pie, J., Yin, Y., Mining Frequent Patterns without Candidate Generation, *School of Computing Science, Simon Fraser University,* 2000.

[4] Woon, Y.K., Ng, W.K., Lim, E.P, A Support-Ordered Trie for Fast Frequent Itemset Discovery, *IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 7, pg 875-879*, 2004.

[5] Woon, Y.K., Ng, W.K., Das, A., Fast Online Dynamic Association Rule Mining, *Proc. Second Int'l Conf. Web Information System Eng.,* pp.278-287, 2001.

[6] Zhao, Q., Bhowmick, S.S., Association Rule Mining: A Survey, *Nanyang Technological University, Singapore,* 2003.