# ANALISIS KINERJA ALGORITMA *PREFIXSPAN* DAN *APRIORIALL* PADA PENGGALIAN POLA SEKUENSIAL

**Rully Soelaiman, Dhany Saputra**
*Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS), Surabaya, 60111*
*E-mail: rully@its-sby.edu*

**ABSTRAKSI**

*PrefixSpan adalah salah satu metode penggalian pola sekuensial yang menggunakan pendekatan Pattern Growth serta melakukan proyeksi sufiks terhadap basis data sekuens, sedangkan AprioriAll adalah salah satu metode penggalian pola sekuensial yang menggunakan pendekatan Apriori serta melakukan pembangkitan dan pengujian sekuens kandidat. Dengan bekal basis data sekuens yang terdiri atas ID kastamer, ID transaksi, dan item yang dibeli, kedua algoritma tersebut dapat menghasilkan pola-pola sekuensial yang memenuhi dukungan minimum yang diinginkan. Secara teoretik kinerja PrefixSpan lebih unggul daripada AprioriAll, sehingga perlu dilakukan analisis terhadap kinerja dari kedua algoritma tersebut.*

*Dalam penelitian ini dilakukan analisis kinerja algoritma PrefixSpan dan AprioriAll pada penggalian pola sekuensial. PrefixSpan akan disusun menggunakan teknik proyeksi basis data Pseudoprojection dan AprioriAll akan disusun sesuai empat fase prosedurnya, yaitu fase pengurutan, fase litemset, fase transformasi, dan fase sekuens. Kemudian kedua aplikasi ini akan diuji ketangguhannya dengan menggali pola sekuensial dari basis data sekuens sintetik berukuran besar.*

*Berdasarkan uji coba dapat disimpulkan bahwa secara umum durasi eksekusi dan utilisasi memori PrefixSpan lebih kecil daripada AprioriAll dan secara grafis dapat dikatakan bahwa PrefixSpan lebih skalabel dan terpercaya daripada AprioriAll.*

***Kata Kunci:*** *PrefixSpan, Pseudoprojection, AprioriAll, pola sekuensial, penggalian data, dukungan minimum, basis data sekuens, proyeksi basis data.*

## 1. PENDAHULUAN

Penggalian pola sekuensial mula-mula didefinisikan oleh Agrawal dan Srikant di [2]: Diberikan sejumlah sekuens, setiap sekuens terdiri atas sederetan elemen, dan setiap elemen terdiri atas sejumlah item, serta diberikan nilai dukungan minimum, penggalian pola sekuensial adalah pencarian semua subsekuens berulang, yaitu subsekuens yang frekuensi kejadiannya tidak lebih kecil daripada dukungan minimum.

Beberapa pendekatan dalam menyelesaikan permasalahan penggalian pola sekuensial adalah pendekatan *Apriori* dan pendekatan *Pattern-Growth* seperti dijelaskan di [1]. Pendekatan *Apriori*, seperti halnya *AprioriAll* [2] dan *GSP* [5], menerapkan pendekatan pembangkitan dan pengujian kandidat. Sayangnya, pendekatan ini memiliki kelemahan bahwa pendekatan ini harus membangkitkan sejumlah besar kandidat, harus memindai basis data berulang kali, dan jumlah kandidat akan meningkat secara eksponensial apabila polanya panjang. Akan tetapi pendekatan *Pattern-Growth*, seperti halnya *FreeSpan* [6] dan *PrefixSpan* [1], menerapkan pendekatan *divide and conquer*. Dengan pendekatan ini, secara rekursif basis data akan diproyeksi menjadi sekumpulan basis data yang lebih kecil berdasarkan pola berulang saat itu, lalu proyeksi tersebut digali untuk memperoleh polanya. *PrefixSpan* akan memproyeksi prefiksnya saja sehingga ukuran proyeksi basis data akan semakin menyusut dan redundansi pemeriksaan pada setiap posisi yang mungkin dari sebuah kandidat potensial akan tereduksi.

Dalam pemelitian ini akan dilakukan analisis kinerja algoritma *AprioriAll* dan algoritma *PrefixSpan* yang menggunakan teknik proyeksi *Pseudoprojection* pada penggalian pola sekuensial. Definisi permasalahan, algoritma *PrefixSpan* dan *AprioriAll*, serta teknik proyeksi *Pseudoprojection* 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 *PrefixSpan* dan *AprioriAll* akan dijabarkan, serta teknik proyeksi *Pseudoprojection* akan dijelaskan.

### 2.1 AprioriAll

Berdasarkan [2], penggalian pola sekuens menggunakan *AprioriAll* memiliki empat fase, yaitu fase pengurutan, fase litemset, fase transformasi, dan fase sekuens. Di fase pengurutan, basis data sekuens diurutkan berdasarkan ID kastamer dan ID transaksi. Di fase litemset himpunan litemset harus diperoleh. Litemset (*large itemset*) adalah itemset yang memiliki dukungan tak kurang dari dukungan minimum. Setelah litemset diperoleh, litemset dipetakan ke bilangan bulat agar di fase berikutnya litemset diperlakukan sebagai entitas tunggal demi sehingga menghindari adanya pengecekan per item. Di fase transformasi, semua litemset yang muncul di setiap sekuens ditulis dalam bentuk angka petanya. Di fase ini sejumlah sekuens ditransformasi menjadi sejumlah himpunan litemset. Di fase sekuens, himpunan litemset hasil fase transformasi akan digunakan untuk memperoleh pola sekuensial. Fase sekuens dari *AprioriAll* menggunakan algoritma *AprioriAll* dalam memperoleh pola sekuensial yang lengkap.

Algoritma *AprioriAll* dijelaskan di gambar 1. Di setiap perulangan, sejumlah kandidat dibentuk dari *large sequence* ($L_{k-1}$) yang terbentuk dari perulangan sebelumnya sebagai *seed set*. *Seed set* dari awal perulangan diambil dari daftar item yang

ada dalam basis data sekuens. Kandidat berpanjang *k* dibangkitkan dengan cara mencari dua *large sequence* berpanjang *k-1* yang elemen sebelum *k-1* dari keduanya adalah sama. Setelah itu, setiap kandidat diuji dukungannya. Apabila dukungannya kurang dari dukungan minimum yang diberikan, kandidat tersebut akan gugur dan tidak termasuk dalam *large sequence* dan tidak berhak untuk ikut serta dalam pembangkitan kandidat di perulangan berikutnya.

**Algoritma 1 (*AprioriAll*)**
**Masukan:** Basis data sekuens S, dukungan minimum *min_support*
**Luaran**: Himpunan lengkap pola sekuensial
**Metode**:
1) k = 1
2) Pindai *S* satu kali, dapatkan distinct items $L_k$
3) Do
4) (
5) k = k + 1
6) $C_k$ = kandidat baru, dibangkitkan dari $L_{k-1}$
7) Hitung dukungan tiap elemen $L_{k-1}$ dalam $C_k$
8) $L_k = C_k$ yang memenuhi *min_support*
9) } while ($L_{k-1} \neq Ø$)

**Gambar 1.** Algoritma AprioriAll

## 2.2 PrefixSpan

Algoritma *PrefixSpan* dapat dilihat di gambar 1 dan dapat dipelajari di [1]. Sebagai contoh, berikut ini adalah langkah-langkah memperoleh himpunan pola sekuensial berdasarkan basis data sekuens pada tabel 1 dengan dukungan minimum 50%:

1) **Dapatkan semua pola sekuensial berpanjang 1**. Pindai basis data sekuens satu kali untuk mendapatkan semua item berulang. Dari basis data sekuens tersebut didapatkan item berulang berpanjang 1 adalah <*a*>:4,<*b*>:4, <*c*>:4, <*d*>:3, <*e*>:3, dan <*f*>:3, dengan notasi "<*pola*>:jumlah" merepresentasikan pola dan jumlah dukungannya.
2) **Bagilah ruang pencarian**. Himpunan lengkap dari pola sekuensial dapat dibagi menjadi enam bagian berdasarkan prefiksnya, yaitu: 1) pencarian pola yang berprefiks <*a*>, 2) pencarian pola yang berprefiks <*b*>, ..., dan 3) pencarian pola yang berprefiks <*f*>.
3) Dapatkan pola sekuensial berikutnya berdasarkan ruang pencarian. Pola sekuensial berikutnya bisa diperoleh dengan membangun proyeksi basis data dan menggalinya secara rekursif.

**Tabel 1.** Contoh Basis Data Sekuens

| *ID sekuens* | *Sekuens* |
|---|---|
| 1 | <a(abc)(ac)d(cf)> |
| 2 | <(ad)c(bc)(ae)> |
| 3 | <(ef)(ab)(df)cb> |
| 4 | <eg(af)cbc> |

**Algoritma 1 (*PrefixSpan*)**
**Masukan**: Basis data sekuens S, dukungan minimum *min_support*
**Luaran**: Himpunan lengkap pola sekuensial
**Metode**:
1) *PrefixSpan*(<> ,0,S)

**procedure *PrefixSpan*** (a, L, $S|_\alpha$ )

1) Pindai $S|_\alpha$ satu kali, dapatkan semua item berulang b, sedemikian sehingga :
   a) b dapat diletakkan di dalam elemen terakhir dari α untuk membentuk pola sekuensial.
   b) <b> dapat diletakkan di belakang a untuk membentuk pola sekuensial.
2) Untuk setiap item berulang b, letakkan b di belakang α untuk membentuk pola sekuensial $\alpha'$ dan menghasilkan $\alpha'$.
3) Untuk setiap $\alpha'$, bangunlah proyeksi basis data $\alpha'$ $S|_{\alpha'}$.
4) *PrefixSpan* ( $\alpha'$, L+1, $S|_{\alpha'}$ )

**Gambar 2.** Algoritma PrefixSpan

Himpunan pola sekuensial adalah kumpulan pola yang ditemukan dalam proses penggalian rekursif di atas, seperti tercantum di Tabel 3.

## 2.3 Teknik Proyeksi *Pseudoprojection*

Salah satu teknik proyeksi untuk mereduksi jumlah dan ukuran proyeksi basis data adalah *Pseudoprojection*. *Pseudoprojection* mengelola indeks prefiks dari setiap kastamer melalui sebuah struktur data pasangan *pointer-offset*. *Pointer* menyatakan id sekuens dan *offset* menyatakan posisi awal dari proyeksi basis data dalam sekuens. Setiap *offset* mengindikasikan posisi awal proyeksi dari sebuah sekuens.

Sebagai contoh, himpunan pola sekuensial akan digali dari basis data sekuens pada tabel 2.1 dengan dukungan minimum 50% dan dengan menerapkan *Pseudoprojection* pada *PrefixSpan* (diasumsikan basis data sekuens dapat ditangani oleh memori). Beberapa simbol dalam *Pseudoprojection* adalah **$** yang mengindikasikan bahwa prefiks menyumbangkan frekuensi kejadian dalam sekuens tetapi proyeksi sufiksnya kosong dan **Ø** yang mengindikasikan bahwa tidak ada kejadian dari prefiks dalam sekuens yang bersesuaian.

## 3. HASIL 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 2.

**Tabel 2.** Basis Data Proyeksi

| *Cust_ID* | *Sekuens* | *<a>* | *<b>* | *<c>* | *<d>* | *<e>* | *<f>* | *<aa>* | *<ab>* | *<(ab)>* | *<ac>* | *<ad>* | *<ae>* | *<af>* | *<(ab)c>* | *…* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | <a(abc)(ac)d(cf)> | 1,3,7 | 4 | 5,8 | 10 | Ø | $ | 3,7 | 4 | 4 | 5,8 | 10 | Ø | $ | 8 | … |
| 2 | <(ad)c(bc)(ae)> | 1,9 | 6 | 4,7 | 2 | $ | Ø | 9 | 6 | Ø | 4,7 | Ø | $ | Ø | Ø | … |
| 3 | <(ef)(ab)(df)cb> | 4 | 5 | 10 | 7 | 1 | 2,8 | Ø | 5 | 5 | 10 | 7 | Ø | 8 | 10 | … |
| 4 | <eg(af)cbc> | 5 | 10 | 8 | Ø | 1 | 6 | Ø | 10 | Ø | 8 | Ø | Ø | Ø | Ø | … |

**Tabel 3.** Himpunan Pola Sekuensial

| *Prefiks* | *Proyeksi Basis Data* | *Pola Sekuensial* |
|---|---|---|
| <a> | <(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc> | <a>:4, <aa>:2, <ab>:4, <aba>:2, <abc>:2, <a(bc)>:2, <a(bc)a>:2, <ac>:4, <aca>:2, <acb>:3, <acc>:3, <ad>:2, <adc>:2, <af>:2, <(ab)>:2, <(ab)c>:2, <(ab)d>:2, <(ab)dc>:2, <(ab)f>:2 |
| <b> | <(_c)(ac)d(cf)>, <(_c)(ae)>, <(df)cb>, <c> | <b>, <ba>:2, <bc>:3, <bd>:2, <bdc>:2, <bf>:2, <(bc)>:2, <(bc)a>:2 |
| <c> | <(ac)d(cf)>, <(bc)(ae)>, <b>, <bc> | <c>, <ca>:2, <cb>:3, <cc>:3 |
| <d> | <(cf)>, (c(bc)(ae)>, <(_f)cb> | <d>, <db>:2, <dc>:3, <dcb>:2 |
| <e> | <(_f)(ab)(df)cb>, <(af)cbc> | <e>, <ea>:2, <eab>:2, <eac>:2, <eacb>:2, <eb>:2, <ebc>:2, <ec>:2, <ecb>:2, <ef>:2, <efb>:2, <efc>:2, <efcb>:2 |
| <f> | <(ab)(df)cb>, <cbc> | <f>, <fb>:2, <fbc>:2, <fc>:2, <fcb>:2 |

**Uji Coba Kebenaran**

Uji coba kebenaran dilakukan untuk menguji validitas sistem aplikasi. Basis data sekuens pada Tabel 1 digunakan untuk uji coba ini. Setelah dilakukan uji coba ternyata pola yang dihasilkan oleh kedua algoritma sama dengan pola sekuensial di Tabel 2 dan paper [1].

**Uji Coba Kinerja**

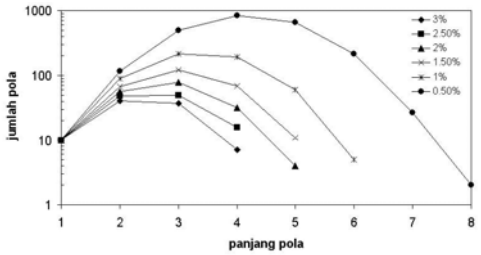
Uji coba kinerja dilakukan untuk menguji kehandalan kinerja algoritma *PrefixSpan* dibandingkan *AprioriAll* pada sejumlah basis data sekuens sintetik dari berbagai ukuran dan distribusi data.

Pengujian pertama dilakukan terhadap himpunan data C1T2S2I2.5 Versi I, yang mengandung 1000 kastamer dengan 10 item yang berbeda. Rata-rata jumlah item dalam satu transaksi dan rata-rata jumlah transaksi dalam satu sekuens adalah 2. Satu pola terdiri atas rata-rata 4 transaksi dan setiap transaksi terdiri atas rata-rata 1.25 item.

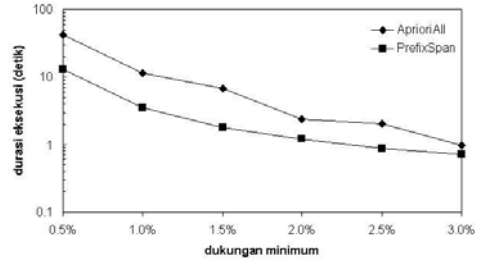
Gambar 5 menunjukkan distribusi jumlah pola dari himpunan data C1T2S2I1.25 untuk dukungan minimum 3% hingga 0.5%. Berdasarkan grafik tersebut, penurunan dukungan minimum menyebabkan peningkatan eksponensial terhadap total jumlah pola yang diperoleh. Semakin kecil dukungan minimum, semakin besar panjang maksimum pola dan semakin banyak jumlah pola untuk panjang pola yang sama. Jumlah pola yang diperoleh mulanya akan mengalami peningkatan yang divergen seiring dengan meningkatnya panjang pola yang telah diperoleh, lalu di suatu titik kulminasi jumlah pola akan mengalami penurunan karena mengalami konvergensi dalam pembentukan pola maksimal.

Gambar 6 menunjukkan durasi eksekusi dari kedua algoritma untuk dukungan minimum 3% hingga 0.5%. Berdasarkan grafik tersebut, *PrefixSpan* selalu berjalan lebih cepat daripada *AprioriAll*. *PrefixSpan* juga dikatakan lebih skalabel daripada *AprioriAll* dikarenakan konsistensi peningkatan durasi eksekusi *PrefixSpan* saat penurunan dukungan minimum lebih baik daripada *AprioriAll*.

Gambar 7 menunjukkan distribusi jumlah pola dari himpunan data C1T2S2I1.25 untuk dukungan minimum tak lebih dari 1%. Berdasarkan grafik tersebut, penurunan linear dari dukungan minimum diikuti oleh peningkatan eksponensial dari jumlah pola.



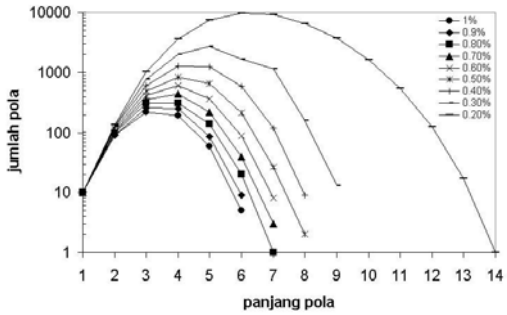

**Gambar 5.** Distribusi pola dataset C1T2S2I1.25 Versi I (min_support 3-0.5%)




**Gambar 6.** PrefixSpan vs AprioriAll pada dataset C1T2S2I1.25 Versi 1 (min_support 3% - 0.5%)

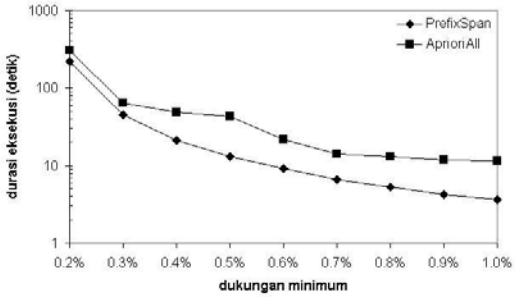
Gambar 8 menunjukkan durasi eksekusi dari kedua algoritma untuk dukungan minimum tak lebih dari 1%. Berdasarkan grafik tersebut, *PrefixSpan* selalu berjalan lebih cepat daripada *AprioriAll*. Namun, semakin kecil dukungan minimum, *PrefixSpan* (durasi eksekusi = 220.391 detik) hanya sedikit lebih cepat daripada *AprioriAll* (durasi eksekusi = 304.854 detik). Meskipun demikian, skalabilitas grafik *PrefixSpan* tetap terjaga.

Pengujian kedua dilakukan terhadap himpunan data C1T2S2I2.5 Versi II, yang mengandung 1000 kastamer dengan 150 item yang berbeda. Rata-rata jumlah item dalam satu transaksi dan rata-rata jumlah transaksi dalam satu sekuens adalah 2. Satu pola terdiri atas rata-rata 4 transaksi dan setiap transaksi terdiri atas rata-rata 1.25 item.




**Gambar 7.** Distribusi pola dataset C1T2S2I1.25 Versi I (min_support ≤ 1%)

Gambar 9 menunjukkan distribusi jumlah pola dari himpunan data C1T2S2I1.25 untuk dukungan minimum tak lebih dari 1%. Berdasarkan grafik tersebut, jumlah dari pola yang lebih panjang selalu lebih kecil daripada jumlah dari pola yang lebih pendek, kecuali untuk pola berpanjang satu. Penurunan nilai dukungan minimum diikuti oleh peningkatan panjang maksimum dari himpunan pola. Ternyata semakin besar jumlah *distinct items*, panjang maksimal dari himpunan pola yang didapat akan semakin kecil, lokasi titik kulminasi akan semakin ke arah panjang sekuens yang lebih kecil, dan konvergensi pembentukan pola maksimal akan semakin dini.

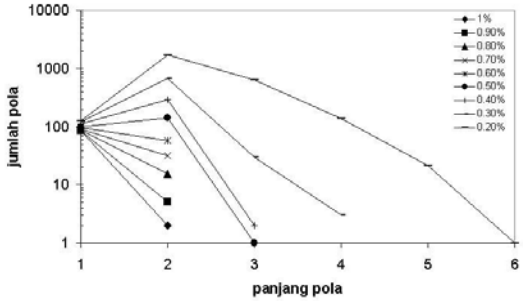



**Gambar 8.** PrefixSpan vs AprioriAll pada dataset C1T2S2I1.25 Versi 1 (min_support ≤ 1%)

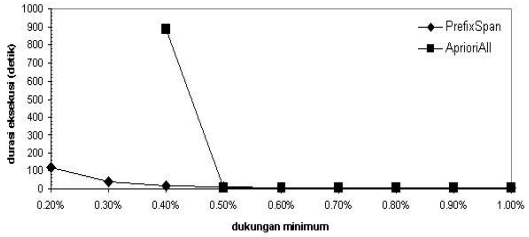
Gambar 10 menunjukkan durasi eksekusi dari kedua algoritma untuk dukungan minimum tak lebih dari 1%. Berdasarkan grafik tersebut, *PrefixSpan* dapat dikatakan lebih skalabel daripada *AprioriAll* karena saat dukungan minimum sangat kecil *AprioriAll* akan melambat secara dramatis sedangkan *PrefixSpan* akan masih tetap meningkat secara perlahan. Terlebih lagi, saat dukungan minimum mencapai tak lebih dari 0.3% *AprioriAll* sudah tidak mampu menyelesaikan tugasnya sehingga juga dapat dikatakan bahwa *PrefixSpan* lebih terpercaya daripada *AprioriAll*.

Pengujian ketiga dilakukan terhadap himpunan data C1T4S2I1.25, yang mengandung 1000 kastamer dengan 10 item yang berbeda. Rata-rata jumlah item dalam satu transaksi adalah 4 dan rata-rata jumlah transaksi dalam satu sekuens adalah 2. Satu pola terdiri atas rata-rata 4 transaksi dan setiap transaksi terdiri atas rata-rata 1.25 item.

Gambar 11 menunjukkan distribusi jumlah pola dari himpunan data ini untuk dukungan minimum tak lebih dari 1%. Berdasarkan grafik tersebut, semua pola memiliki panjang maksimum dua dan letak pola nampak lebih tersebar. Ternyata semakin besar rata-rata jumlah item dalam suatu transaksi maka semakin besar durasi eksekusi untuk dukungan minimum yang sama dan semakin dini konvergensi pembentukan pola maksimal.

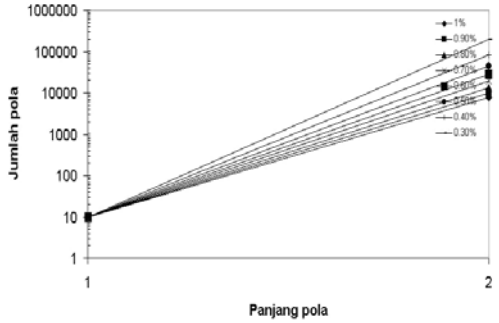



**Gambar 9.** Distribusi pola dataset C1T2S2I1.25 Versi II (min_support ≤ 1%)

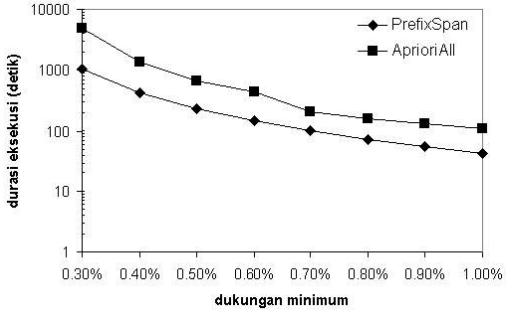



**Gambar 10.** PrefixSpan vs AprioriAll pada dataset C1T2S2I1.25 Versi II (min_support ≤ 1%)

Gambar 12 menunjukkan durasi eksekusi dari kedua algoritma untuk dukungan minimum tak lebih dari 1%. Berdasarkan grafik tersebut, *PrefixSpan* selalu lebih cepat daripada *AprioriAll*. Di samping itu stabilitas peningkatan *PrefixSpan* tetap jauh lebih terjamin dibandingkan *AprioriAll*.



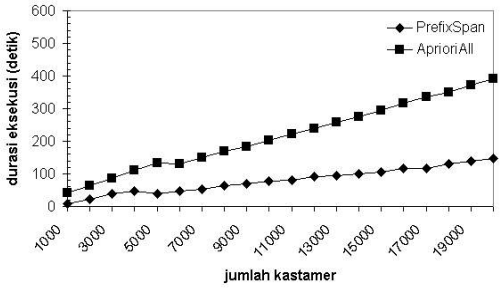

**Gambar 11.** Distribusi pola dataset C1T4S2I1.25 (min_support ≤ 1%)




**Gambar 12.** PrefixSpan vs AprioriAll pada dataset C1T4S2I1.25 (min_support ≤ 1%)

Pengujian keempat dilakukan terhadap himpunan data T3S2I1.25, yang mengandung 10 item yang berbeda. Rata-rata jumlah item dalam satu transaksi adalah 2 dan rata-rata jumlah transaksi dalam satu sekuens adalah 3. Satu pola terdiri atas rata-rata 4 transaksi dan setiap transaksi terdiri atas 1.25 item. Dalam skenario ini jumlah kastamer bisa diubah-ubah menjadi 1000 hingga 20000.

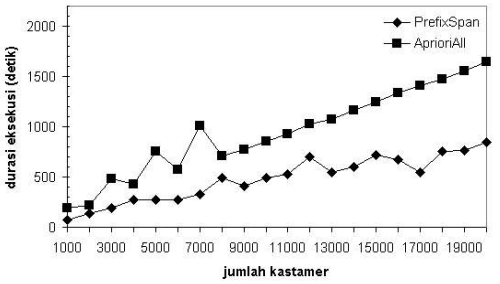
Gambar 13 menunjukkan durasi eksekusi dari kedua algoritma untuk dukungan minimum 2.5%. Berdasarkan grafik tersebut, *PrefixSpan* rata-rata lebih cepat 2.8 kali daripada *AprioriAll*.




**Gambar 13.** Uji skalabilitas PrefixSpan vs AprioriAll pada dataset T3S2I1.25 dengan dukungan minimum 2.5%

Gambar 14 menunjukkan durasi eksekusi dari kedua algoritma untuk dukungan minimum 1%. Berdasarkan grafik tersebut, *PrefixSpan* lebih cepat satu setengah hingga tiga kali lipat daripada *AprioriAll*.

Gambar 15 menunjukkan durasi eksekusi dari kedua algoritma untuk dukungan minimum 0.75%. Berdasarkan grafik tersebut, durasi eksekusi *AprioriAll* meningkat secara lebih dramatis daripada *PrefixSpan* sehingga dapat dikatakan *PrefixSpan* lebih skalabel daripada *AprioriAll*.



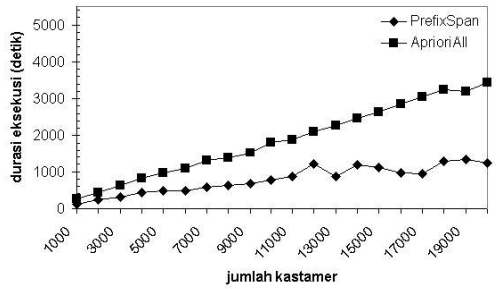

**Gambar 14.** Uji skalabilitas *PrefixSpan* vs *AprioriAll* pada dataset T3S2I1.25 dengan dukungan minimum 1%

Dari ketiga grafik tersebut diperoleh bahwa semakin banyak jumlah kastamer yang dicari polanya, semakin kecil selisih durasi eksekusi antara *PrefixSpan* dan *AprioriAll*, meskipun secara umum durasi eksekusi *PrefixSpan* tetap lebih baik daripada *AprioriAll*.

Pengujian kelima dilakukan terhadap dataset C1T2S2I1.25 Versi 1. Gambar 7 menunjukkan distribusi jumlah pola dari dataset C1T4S2I1.25 untuk dukungan minimum tak lebih dari 1%.

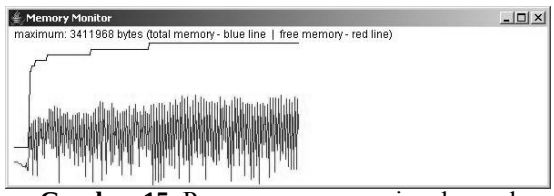
Gambar 15 hingga 18 menunjukkan eksekusi dari aplikasi Memory Monitor yang digunakan untuk mengawasi penggunaan memori selama aplikasi berjalan. Garis di atas menunjukkan *total memory*, sedangkan garis di bawahnya menunjukkan *free memory*.

Berdasarkan gambar 15 hingga 18, memori di aplikasi *PrefixSpan* selalu dibebaskan setelah selesai suatu subproses dan lalu kembali ke ukuran memori *used memory* saat subproses lain dilakukan. Akan tetapi, *AprioriAll* cenderung menggunakan *used memory* sebesar nilai maksimum dari *used memory* saat itu secara konstan. Hal ini disebabkan oleh adanya pembangkitan dan penyimpanan kandidat ke dalam memori pada *AprioriAll*. *PrefixSpan* tidak menyimpan kandidat, melainkan cukup menyimpan indeks sehingga objek yang didestruksi menjadi lebih banyak. Berdasarkan perubahan nilai *maximum used memory*, kebutuhan akan tambahan *used memory* relatif selalu meningkat pada *AprioriAll* dan relatif jarang meningkat pada *PrefixSpan*.

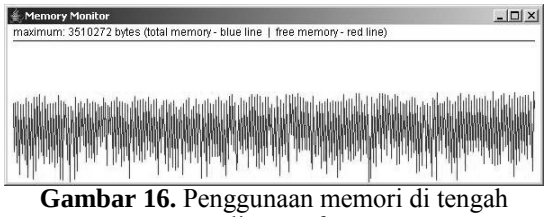



**Gambar 14.** Uji skalabilitas *PrefixSpan* vs *AprioriAll* pada dataset T3S2I1.25 dengan dukungan minimum 0.75%

Gambar 19 menunjukkan daya guna memori dari kedua algoritma untuk dukungan minimum tak lebih dari 1%. Berdasarkan grafik tersebut, daya guna memori *PrefixSpan* selalu lebih efisien dan lebih stabil daripada *AprioriAll* terutama saat dukungan minimum semakin kecil.




**Gambar 15.** Penggunaan memori pada awal penggalian *PrefixSpan*

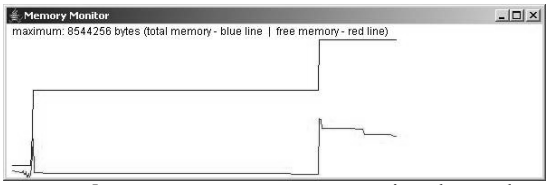



**Gambar 16.** Penggunaan memori di tengah penggalian *PrefixSpan*

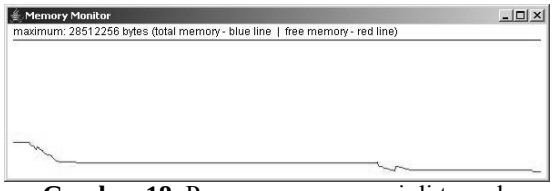
Pengujian terakhir dilakukan terhadap dataset C1T2S2I1.25 Versi 2. Gambar 9 menunjukkan distribusi jumlah pola dari dataset C1T4S2I1.25 untuk dukungan minimum tak lebih dari 1%.

Gambar 20 menunjukkan daya guna memori dari kedua algoritma untuk dukungan minimum tak lebih dari 1%. Berdasarkan grafik tersebut, daya

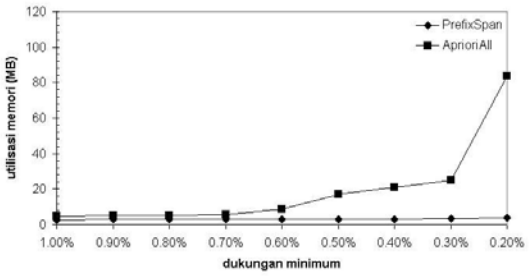
guna memori *PrefixSpan* relatif lebih besar daripada *AprioriAll*. Ini karena jumlah *distinct items* meningkat sehingga ukuran penyimpanan menjadi lebih besar. Meskipun demikian, *PrefixSpan* tetap menunjukkan skalabilitas daya guna memori yang lebih baik daripada *AprioriAll*.



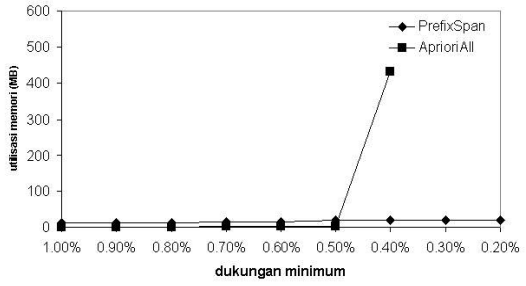

**Gambar 17.** Penggunaan memori pada awal penggalian *AprioriAll*




**Gambar 18.** Penggunaan memori di tengah penggalian *AprioriAll*




**Gambar 19.** *U*ji Memori *PrefixSpan* vs *AprioriAll* pada dataset C1T2S2I1.25 Versi I




**Gambar 20.** *U*ji Memori *PrefixSpan* vs *AprioriAll* pada dataset C1T2S2I1.25 Versi II

## 4. SIMPULAN

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

1. Durasi eksekusi, skalabilitas, reliabilitas, dan utilisasi memori *PrefixSpan* lebih baik daripada *AprioriAll*.
2. Jumlah pola 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.
3. 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.
4. Semakin besar jumlah *distinct items*, panjang maksimal pola yang didapat akan semakin kecil, lokasi titik kulminasi akan semakin ke arah panjang sekuens yang lebih kecil, dan konvergensi pembentukan pola maksimal akan semakin dini.
5. Jumlah kastamer yang dicari polanya berbanding terbalik dengan selisih durasi eksekusi antara *PrefixSpan* dan *AprioriAll*, meskipun durasi eksekusi *PrefixSpan* masih lebih baik daripada *AprioriAll*.
6. 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] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, et al. "*Mining Sequential Patterns by Pattern Growth: The PrefixSpan Approach"*. IEEE Transaction on Knowledge and Data Engineering, vol. 16, nos. 11, pp. 1041-4347, November 2004.

[2] Rakesh Agrawal and Ramakrishnan Srikant. "*Mining Sequential Patterns*". Proc. 1995 Int'l Conf. Data Eng. (ICDE '95), pp. 3-14, March 1995.

[3] Jiawei Han and Micheline Kamber. *"Data Mining: Concept and Techniques",* Morgan Kaufmann Publisher, 2000.

[4] Jack Shirazi, "*Java™ Performance Tuning, 2nd Edition*", O'Reilly Publisher, 2003.

[5] Rakesh Agrawal and Ramakrishnan Srikant, "*Mining Sequential Patterns: Generalization and Performance Improvements*," Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96), pp. 3-17, March 1996.

[6] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu. "*Freespan: Frequent pattern-projected sequential pattern mining*" Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00), pp. 355–359, August 2000.

[7] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, et al. "*PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth*," Proc. 2001 Int'l Conf. Data Eng. (ICDE '01), pp. 215-224, April 2001.

[8] Sonny David P.B., Arif Djunaidy, Rully Soelaiman. *"Penerapan Algoritma DynamicSome pada Data Mining untuk Pola Sekuensial".* Skripsi Jurusan Teknik Informatika Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember, 2000.

[9] Terry Quatrani. "*Visual Modeling With Rational Rose 2002 and UML*". Addison-Wesley Publisher, 2003.