Algoritme *Generalized Sequential Pattern* (GSP)

(Ramos Somya)

Algoritme *Generalized Sequential Pattern* (GSP) adalah suatu algoritme yang dapat memproses dan menemukan semua pola sekuensial dan non sekuensial yang ada pada suatu data *sequence*. Algoritme GSP digunakan pada *mining sequence* dan cocok untuk memecahkan masalah *mining sequence* yang didasarkan pada prinsip algoritme Apriori, di mana algoritme GSP akan membangkitkan *frequent sequences*, sedangkan algoritme Apriori akan membangkitkan *frequent itemset*. Selain untuk menemukan aturan asosiasi, fungsi utama dari algoritme GSP yaitu menemukan pola sekuensial atau urutan (Zaki, 1997).

Setiap data sequential merupakan suatu daftar dari transaksi-transaksi, di mana setiap transaksi merupakan sekumpulan item. Umumnya setiap transaksi diasosiasikan dengan waktu transaksinya. Suatu *sequential-pattern* juga terdiri dari suatu daftar dari sekumpulan item. Masalahnya adalah bagaimana menemukan *sequential pattern* dengan *minimum support* yang ditentukan oleh *user*, di mana *support* dari sebuah *sequential pattern* merupakan persentase dari *data-sequences* yang mengandung suatu pola tertentu. Contoh data *sequence* misalnya: seorang pelanggan yang menyewa film "*Star Wars*", kemudian di transaksi berikutnya menyewa film "Titanic", dan di transaksi berikutnya menyewa film "*Iron Man*" dan "*Hereditary*". Data-data ini jika dibuat dalam tabel akan terlihat seperti Tabel 1.

Tabel 1 Contoh Data *Sequence* Transaksi

Customer ID	*Transaction Time*	*Items*
1	10	C, D
1	15	A, B, C, D
1	20	A, B. F
1	25	A, C, D, F
2	15	A, B, F
2	20	E
3	10	D, G, H
3	20	B, F
3	25	A, G, H

Algoritme GSP melakukan *multiple passes* melalui data yang ada untuk mencari pola sekuensial. Fase pertama menentukan *support* dari masing-masing item yang mana merupakan nomor dari *data-sequences* yang termasuk item-item tersebut pada akhir dari fase pertama. Algoritme ini akan mengetahui atau mendapatkan item mana yang akan menjadi *frequent*, yaitu yang memenuhi *minimum support*. Masing-masing item menghasilkan sebuah *frequent sequence* yang pertama yang terdiri dari item tersebut. Masing-masing *subsequence* pada setiap fase pada awalnya dimulai dengan sekumpulan calon candidat: suatu *frequent-sequence* yang ditemukan atau dihasilkan pada fase sebelumnya. Sekumpulan calon *candidate* tersebut digunakan untuk menghasilkan *frequent sequences* baru yang berpotensial, yang disebut *candidate sequences*. Masing-masing *candidate sequence* memiliki lebih dari satu item dari pada calon *sequence*; sehingga semua *candidate sequences* dalam suatu fase akan memiliki item dengan nomor yang sama. *Support* dari *candidate sequences* ini ditemukan selama proses melalui data yang ada. Pada akhir dari fase tersebut, algoritme GSP akan menghasilkan yang mana *candidate sequences* yang termasuk dalam *frequent*, di mana *frequent candidate* tersebut menjadi calon *candidate* untuk fase selanjutnya. Algoritme GSP berakhir ketika tidak ditemukan lagi *frequent sequences* pada akhir suatu fase, atau ketika tidak ada lagi *candidate sequences* yang dihasilkan. Terdapat 2 langkah utama dalam algoritme ini, yaitu *candidate generation* dan *support counting* (Srikant dan Agrawal, 1996).

a. *Candidate Generation*

Pada tahap candidate generation, terdapat 2 langkah, yaitu:

- *Join Phase*, di mana *candidate sequence* dihasilkan dengan melakukan proses *join* atau penggabungan antara L_{k-1} dengan dirinya sendiri. Sekumpulan *candidate* yang dihasilkan dalam proses *join* ini nantinya akan dinotasikan dalam C_k. Di mana aturan dari fase *join* ini adalah setiap *candidate* yang dihasilkan tidak boleh mengandung *candidate* yang kembar antara satu dengan yang lainnya. Algoritme GSP melakukan *multiple passes* pada data, di mana pertama kali akan ditentukan *frequent* 1-*item patterns* (L_1). Berikutnya dilakukan pembangkitan *candidate sequence* (C_k) menggunakan GSP *gen join*. Untuk mendapatkan k-*sequence candidate* (C_k), *frequent sequence* yang didapatkan pada langkah sebelumnya (L_{k-1}), digabungkan dengan dirinya sendiri menggunakan prinsip Apriori. Pada proses penggabungan ini, setiap *sequence* - s pada L_{k-1} digabungkan dengan *sequence* - s yang lain di L_{k-1} jika elemen terakhir pada s (kecuali elemen pertama pada s) sama dengan elemen

pertama dari s (kecuali elemen terakhir pada s). Misalnya, *sequence* - s {(1)(2)(3)} dapat digabungkan dengan *sequence* - s {(2)(35)} karena elemen terakhir dari s yaitu (2)(3) adalah sama dengan elemen pertama dari (2)(3). Setelah *join phase*, pada *prune phase*, *candidate sequence* (C_k) yang tidak memenuhi *minimum support* akan dihapus untuk membangkitkan *frequent sequence* berikutnya (L_{k+1}). Proses ini dilakukan berulang sampai tidak ada lagi *candidate sequence* yang bisa dihasilkan.

- *Prune Phase*. Fase ini melakukan penghapusan *candidate sequence* yang tidak memenuhi *minimum support* yang telah ditentukan. Berdasarkan definisi, semua *candidate* yang memiliki jumlah *support* yang lebih besar sama dengan dari *minimum support* yang telah ditentukan disebut *frequent*, yang artinya juga memenuhi syarat untuk masuk menjadi L_k Di mana C_k dapat juga mengandung *candidate* dengan jumlah yang sangat besar yang berarti juga akan menyebabkan proses penghitungan C_k selanjutnya akan berjalan sangat lambat. Untuk mengurangi jumlah *candidate* C_k maka semua itemset yang tidak *frequent* tidak mungkin dapat menjadi *subset* dari *frequent* k-itemset. Oleh karena itu jika ada sebuah *subset* dari *candidate* k-itemset yang tidak termasuk dalam L_{k-1} maka *candidate* tidak mungkin *frequent* juga dan oleh karena itu dapat dihapus dari c_k.

b. *Counting Candidates*

Yaitu menemukan semua kandidat dalam suatu data *sequence* S. Secara konsep, semua k-*sub-sequence* dari S akan dibangkitkan. Misalnya, diberikan sekumpulan *candidate sequences* C dan data *sequence* D, di sini yang perlu ditemukan adalah semua *sequence* pada C yang termasuk di dalam D.

Contoh Kasus:

Berikut ini pada Tabel 2 diberikan *sequence dataset* transaksi yang akan dicari *sequence pattern*-nya menggunakan algoritme GSP dengan *minimum support* = 2.

Tabel 2 *Sequence* Dataset Transaksi

SID	*Time*	*Items*
1	10, 15, 20, 25	<{cd}{abc}{abf}{acdf}>
2	15, 20	<{abf}{e}>
3	10	<{abf}>
4	10, 20, 25	<{dgh}{bf}{agh}>

1. Menentukan *frequent sequence* 1-*itemset* berdasarkan Tabel 3.

Tabel 3 Penentuan *Frequent Sequence* 1-*itemset* (C_1)

Items	Frekuensi Kemunculan
{a}	4
{b}	4
{c}	1
{d}	2
{e}	1
{f}	4
{g}	1
{h}	1

Berdasarkan Tabel 3, item yang memenuhi *minimum support* adalah a, b, d, f, sehingga, item a, b, d, f merupakan *frequent* 1-*Sequence* (L_1) seperti ditunjukkan pada Tabel 4.

Tabel 4 *Frequent* 1-*Sequence* (L_1)

Items	Frekuensi Kemunculan
{a}	4
{b}	4
{d}	2
{f}	4

2. Langkah berikutnya adalah melakukan *Join Phase* dengan menggabungkan L_1 dan L_1 untuk membangkitkan *frequent sequence* 2-*itemset* (C_2).

   Sebagai catatan, {(a, b)} adalah sama dengan {(b, a)}, sedangkan {ab} tidak sama dengan {ba}. Artinya, {(a, b)} merupakan item-item yang dibeli sekaligus dalam satu kali transaksi, sedangkan {ab} merupakan item-item pada transaksi yang berbeda, atau dengan kata lain a→b (jika membeli a pada suatu transaksi, maka akan membeli b pada transaksi berikutnya).

   Hasil C_2 ditunjukkan pada Tabel 5.

Tabel 5 *Frequent Sequence* 2-*itemset* (C_2)

Items	Frekuensi Kemunculan
{ab}	1
{ad}	1
{af}	1
{(a,b)}	3
{(a,d)}	1
{(a,f)}	3
{ba}	2
{bd}	1
{bf}	1
{(b,d)}	0
{(b,f)}	4
{da}	2
{db}	2
{df}	2
{(d,f)}	1
{fa}	2
{fb}	0
{fd}	1

Atau jika dijabarkan secara lengkap akan terlihat seperti Tabel 6.

Tabel 6 Penjabaran *Frequent Sequence* 2-*itemset*

2-len Seq (a)		2-len Seq (b)		2-len Seq (d)		2-len Seq (f)	
{ab}	1	{ba}	2	{da}	2	{fa}	2
{ad}	1	{bd}	1	{db}	2	{fb}	0
{af}	1	{bf}	1	{df}	2	{fd}	1
{(a,b)}	3	{(b,d)}	0	{(d,f)}	1		
{(a,d)}	1	{(b,f)}	4				
{(a,f)}	3						

Kandidat *sequence* pada C_2 yang memenuhi *minimum support* ditunjukkan pada Tabel 7.

Tabel 7 *Frequent* 2-*Sequence* (L_2)

Items	Frekuensi Kemunculan
{(a,b)}	3
{(a,f)}	3
{ba}	2
{(b,f)}	4
{da}	2
{db}	2
{df}	2
{fa}	2

3. Langkah berikutnya adalah melakukan *Join Phase* dengan menggabungkan L_2 dan L_2 untuk membangkitkan *frequent sequence* 2-*itemset* (C_3). Cara penggabungan ini mengacu pada ketentuan pada bagian *Join Phase* di penjelasan sebelumnya.

   Contoh penggabungan L_2 dan L_2 misalnya:

   {~~a~~b} dan {b~~e~~} digabungkan menjadi {abe}

   {(~~a~~,b)} dan {b~~e~~} digabungkan menjadi {(a,b)e}

   {~~a~~b} dan {a~~e~~} tidak dapat digabungkan.

   Rincian penggabungan L_2 dan L_2 adalah sebagai berikut:

   {(a,b)} dan {(a,f)}

   {(a,b)} dan {ba} → {(a,b)a}

   {(a,b)} dan {(b,f)} → {a,b,f}

   {(a,b)} dan {da}

   {(a,b)} dan {db}

   {(a,b)} dan {df}

   {(a,b)} dan {fa}

{(a,f)} dan {(a,b)}

{(a,f)} dan {ba}

{(a,f)} dan {(b,f)}

{(a,f)} dan {da}

{(a,f)} dan {db}

{(a,f)} dan {df}

{(a,f)} dan {fa} → {(a,f)a}

{ba} dan {(a,b)} → {b(a,b)}

{ba} dan {(a,f)} → {b(a,f)}

{ba} dan {(b,f)}

{ba} dan {da}

{ba} dan {db}

{ba} dan {df}

{ba} dan {fa}

{(b,f)} dan {(a,b)}

{(b,f)} dan {(a,f)}

{(b,f)} dan {ba}

{(b,f)} dan {da}

{(b,f)} dan {db}

{(b,f)} dan {df}

{(b,f)} dan {fa} → {(b,f)a}

{da} dan {(a,b)} → {d(a,b)}

{da} dan {(a,f)} → {d(a,f)}

{da} dan {ba}

{da} dan {(b,f)}

{da} dan {db}

{da} dan {df}

{da} dan {fa}

{db} dan {(a,b)}

{db} dan {(a,f)}

{db} dan {ba} → {dba}

{db} dan {(b,f)} → {d(b,f)}

{db} dan {da}

{db} dan {df}

{db} dan {fa}

{df} dan {(a,b)}

{df} dan {(a,f)}

{df} dan {ba}

{df} dan {(b,f)}

{df} dan {da}

{df} dan {db}

{df} dan {fa} → {dfa}

{fa} dan {(a,b)} → {f(a,b)}

{fa} dan {(a,f)} → {f(a,f)}

{fa} dan {ba}

{fa} dan {(b,f)}

{fa} dan {da}

{fa} dan {db}

{fa} dan {df}

Hasil penggabungan ditunjukkan pada Tabel 8.

Tabel 8 *Frequent Sequence* 3-*itemset* (C_3)

Items	Frekuensi Kemunculan
{(a,b)a}	1
{a,b,f}	3
{(a,f)a}	1
{b(a,b)}	1
{b(a,f)}	1
{(b,f)a}	2
{d(a,b)}	1
{d(a,f)}	1
{dba}	2
{d(b,f)}	2
{dfa}	2
{f(a,b)}	0
{f(a,f)}	1

Berdasarkan Tabel 8, kandidat *sequence* pada C_3 yang memenuhi *minimum support* ditunjukkan pada Tabel 9.

Tabel 9 *Frequent* 3-*Sequence* (L_3)

Items	Frekuensi Kemunculan
{a,b,f}	3
{(b,f)a}	2
{dba}	2
{d(b,f)}	2
{dfa}	2

4. Langkah berikutnya adalah membangkitkan *frequent sequence* 4-*itemset* (C_4) dengan menggabungkan L_3 dan L_3.

   Rincian penggabungan L_3 dan L_3 adalah sebagai berikut:

{abf} dan {(b,f)a}

{abf} dan {dba}

{abf} dan {d(b,f)}

{abf} dan {dfa}

{(b,f)a} dan {a,b,f}

{(b,f)a} dan {dba}

{(b,f)a} dan {d(b,f)}

{(b,f)a} dan {dfa}

{dba} dan {a,b,f}

{dba} dan {(b,f)a}

{dba} dan {d(b,f)}

{dba} dan {dfa}

{d(b,f)} dan {a,b,f}

{d(b,f)} dan {(b,f)a} → {d(b,f)a}

{d(b,f)} dan {dba}

{d(b,f)} dan {dfa}

{dfa} dan {a,b,f}

{dfa} dan {(b,f)a}

{dfa} dan {dba}

{dfa} dan {d(b,f)}

Hasil penggabungan ini ditunjukkan pada Tabel 10.

Tabel 10 *Frequent Sequence* 4-*itemset* (C_4)

Items	Frekuensi Kemunculan
{d(b,f)a}	2

Kandidat yang memenuhi *minimum support* bergabung dalam L_4.

Tabel 11 *Frequent* 4-*Sequence* (L_4)

Items	Frekuensi Kemunculan
{d(b,f)a}	2

Jadi, hasil akhir dari pembangkitan *sequence* adalah L_4, di mana hanya ada 1 *sequence* saja yaitu {d(b,f)a} dengan frekuensi kemunculan sebanyak 2 kali.

Berdasarkan semua proses *Join Phase* dan *Prune Phase*, didapatkan hasil seperti Tabel 12.

Tabel 12 Hasil Pembangkitan *Frequent Sequence* dari L_1 Hingga L_4

L_1		L_2		L_3		L_4	
{a}	4	{(a,b)}	3	{a,b,f}	3	{d(b,f)a}	2
{b}	4	{(a,f)}	3	{(b,f)a}	2		
{d}	2	{ba}	2	{dba}	2		
{f}	4	{(b,f)}	4	{d(b,f)}	2		
		{da}	2	{dfa}	2		
		{db}	2				
		{df}	2				
		{fa}	2				

5. Hasil pembangkitan *frequent sequence* pada Tabel 12 selanjutnya bisa digunakan untuk pengambilan keputusan. *Sequence* pada L_4 yaitu {d(b,f)a} dapat juga dituliskan dengan notasi {d} → {(b, f)} → {a}, di mana konsumen akan membeli produk a pada transaksi pertama, kemudian membeli produk b dan f pada transaksi kedua, dan pada transaksi ketiga akan membeli produk a.

   Setiap *sequence* yang diperoleh dapat dihitung nilai *support* dan *confidence*-nya. Nilai *support* atau nilai penunjang menunjukkan persentase keberadaan sequence tersebut dalam *dataset*, sedangkan nilai *confidence* atau nilai kepastian akan menunjukkan seberapa kuat kombinasi item dalam *sequence* tersebut.

   Sequence pada L_4 yaitu {d(b,f)a} memiliki *support count* sejumlah 2 dari jumlah semua konsumen (4 orang konsumen), sehingga nilai *support*-nya adalah 2/4 * 100% = 50%. Sedangkan nilai *confidence* untuk sequence tersebut adalah sebesar nilai *support count* dibagi jumlah konsumen yang membeli *item* pada bagian *antecedent* (d), sehingga nilai *confidence*-

nya adalah 2/2 * 100% = 100%. Hasil perhitungan nilai *support* dan *confidence* secara lengkap dapat dilihat pada Tabel 13.

Tabel 13 Hasil Perhitungan Nilai *Support* dan *Confidence* dari L_1 Hingga L_4

Rule	***Support Count***	***Support Value***	***Confidence Value***
L_1			
{a}	4	4 / 4 * 100% = 100%	4 / 4 * 100% = 100%
{b}	4	4 / 4 * 100% = 100%	4 / 4 * 100% = 100%
{d}	2	2 / 4 * 100% = 50%	2 / 2 * 100% = 100%
{f}	4	4 / 4 * 100% = 100%	4 / 4 * 100% = 100%
L_2			
{(a,b)}	3	3 / 4 * 100% = 75%	3 / 4 * 100% = 75%
{(a,f)}	3	3 / 4 * 100% = 75%	3 / 4 * 100% = 75%
{ba}	2	2 / 4 * 100% = 50%	2 / 4 * 100% = 50%
{(b,f)}	4	4 / 4 * 100% = 100%	4 / 4 * 100% = 100%
{da}	2	2 / 4 * 100% = 50%	2 / 2 * 100% = 100%
{db}	2	2 / 4 * 100% = 50%	2 / 2 * 100% = 100%
{df}	2	2 / 4 * 100% = 50%	2 / 2 * 100% = 100%
{fa}	2	2 / 4 * 100% = 50%	2 / 4 * 100% = 50%
L_3			
{a,b,f}	3	3 / 4 * 100% = 75%	3 / 4 * 100% = 75%
{(b,f)a}	2	2 / 4 * 100% = 50%	2 / 4 * 100% = 50%
{dba}	2	2 / 4 * 100% = 50%	2 / 2 * 100% = 100%
{d(b,f)}	2	2 / 4 * 100% = 50%	2 / 2 * 100% = 100%
{dfa}	2	2 / 4 * 100% = 50%	2 / 2 * 100% = 100%
L_4			
{d(b,f)a}	2	2 / 4 * 100% = 50%	2 / 2 * 100% = 100%

Referensi

Srikant, R. dan Agrawal, R. (1996). Mining Sequential Patterns: Generalization and Performance Improvements. 5th International Conference Extending Database Technology (EDBT), Springer-Verlag, Avignon France, pp: 3–17.

Zaki, Mohammed J. (1997). Fast Mining of Sequential Patterns in Very Large Databases. The University of Rochester Computer Science Department Rochester, New York 14627. Technical report 668.