**Mining Frequent Patterns without Candidate Generation**

Jiawei Han, Jian Pei, and Yiwen Yin

By:

Haidar Arya Prasetya (5114100051)

Rahmatin Nadia (5114100130)

Irfan Hanif (5114100177)

**Deskripsi problem**

Apriori heuristic mencapai kinerja yang baik dengan mengurangi ukuran dari calon set. Algortima apriori masih dapat mengalami nontrivial cost seperti dapat merugikan apabila harus menangani jumlah kandidat set yang sangat besar, dan juga hal ini membosdankan apabila harus berulang kali memindai database dan mengecek set kandidat yang besar dengan menyocokkan pattern, terutama untuk pattern berukuran panjang. Lalu apakah ada cara lain untuk mengurangi cost ini?

Keefisiensian dari data mining dapat diperoleh melalui tiga teknik. Pertama, database dengan ukuran besar dikompresi memnjadi struktur data yang lebih kecil untuk menghindari cost dan pengulangan pemindaian database. Kedua, FP-tree based mengadopsi pattern dari metode fragment growth. Ketiga, metode divide and conquer berbasis partisi digunakan untuk mengurai tugas dari mining menjadi tugas yang lebih kecil untuk mengurangi ruang pencarian. Metode FP-growth efisien dan terukur untuk mengukur kedua jenis frequent pattern baik yang panjang maupun yang pendek. Berdasarkan besarnya FP-Growth lebih cepat daripada Apriori dan berdasarkan pertumbuhan marjinnya lebih luas ketika frequent pattern tumbuh memanjang, FP-Growth juga melebihi algoritma TreeProjection.

**Algoritma 1 (FP-tree construction)**

Input : database transaksi DB dan dukungan minimum dari threshold.

Output : Frequent pattern tree, FP-tree

Metode :

1. Memindai database transaksi DB. Mengumpulkan set dari ferquent item F dan masing-masing pendukungnya. Urutkan F dalam urutan descending L, sebagai list dari frequent item
2. Buat root dari FP-tree, T dan labeli sebagai “null”. Utnuk masing-masing Transaksi Trans di DB lakukan select dan sort frequent item menurut rutan L. Lalu panggil fungsi insert\_tree(p[P],T).

Analisis :

Dari proses kontruksi FP-tree, dapat disimpulkan bahwa dibutuhkan dua kali pemindaian transaksi database DB. Yang pertaama untuk mengumpulkan set dari frequent item dan yang kedua untuk membangun FP-tree.

*Lemma 2.1 Given a transaction database DB and a support threshold ξ, its corresponding FP-tree contains the complete information of DB in relevance to frequent pattern mining.*

*Lemma 2.2 Without considering the (null) root, the size of an FP-tree is bounded by the overall occurrences of the frequent items in the database, and the height of the tree is bounded by the maximal number of frequent items in any transaction in the database.*

Pembangunan dari FP-tree kompak memastikan bahwa sub-sequent dari mining dapat dilakukan dengan struktur data yang lebih kompak. Dibawah ini ada sifat-sifat dari struktur FP-tree yang akan memfasilitasi frequent pattern mining

*Property 3.1 (Node-link property) For any frequent item ai, all the possible frequent patterns that contain ai can be obtained by following ai's node-links, starting from ai's head in the FP-tree header.*

*Property 3.2 (Prefix path property) To calculate the frequent patterns for a node ai in a path P, only the prefix subpath of node ai in P need to be accumulated, and the frequency count of every node in the prefix path should carry the same count as node ai.*

*Lemma 3.1 (Fragment growth) Let α be an itemset in DB, B be α's conditional pattern base, and be an itemset in B. Then the support of α ᴜ β in DB is equivalent to the support of β in B.*

*Lemma 3.2 (Single FP-tree path pattern generation) Suppose an FP-tree T has a single path P. The complete set of the frequent patterns of T can be generated by the enumeration of all the combinations of the subpaths of P with the support being the minimum support of the items contained in the subpath.*

**Algoritma 2 (FP-growth mining frequent patterns with FP-tree by pattern fragment growth)**

Input : kontruksi FP-tree berdasarkan algoritma 1, menggunakan DB dan minimum support threshold

Output : set lengkap dari frequent pattern

Metode :

Call FP-Growth(FP-Tree,null). Prosedur FP-growth(Tree,α)

1. **If** Tree mengandung single path maka
2. **Then for each**  kombinasi (denoted sebagai β) dari node pada path P **do**
3. Generate pattern β ᴜ α dengan

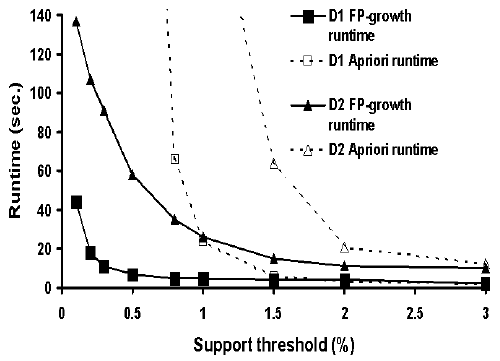
Support = minimum support dari nodes β

1. **Else for each**  ai pada Tree **do {**
2. Generate pattern β = ai ᴜ α dengan support = ai. Support;
3. Kontruksi kondisi pattern β dan kondisi FP-tree Tree β;
4. **If** Tree β ≠ ᴓ
5. **then** call FP-growth (Treeᵝ,β)}

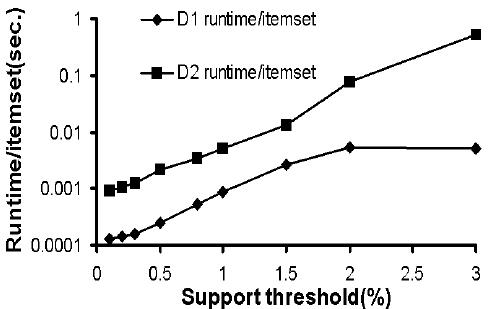
**}**

Analisis : algoritma diatas secara tepat menenukan set lengkap dari frequent itemset pada database transaksi DB

**Perbandingan antara FP-growth dan Apriori**



Skalabilitas dari FP-growth dan Apriori sebagai pendukung threshold menurun dari 3% menjadi 0.1%. ini menunjukkan FP-growth jauh lebih baik daripada Apriori



Gambar diatas menunjukkan bahwa run time itemset dari FP-growth. FP-growth memiliki skalabilitas yang baik dengan penurunan support minimum threshold, meskipun nilai dari frequent itemset bertumbuh secara eksponensial, run time dari FP-growth meningkat secara drastis.

**Conclusion**

Ada beberapa keuntungan dari FP-growth disbanding pendekatan yang lainnya, pertama FP-growth membangun FP-tree yang sangat tersusun rapat, yang pada dasarnya berukuran lebih kecil daripada database yang aslinya. Kedua FP-growth menerapkan metode pola pertumbuhan yang menolak cost dari calon generasi dan diuji secara sukses dengan menggabungan 1 itemset yang ditemukan di FP-tree, pada kasus ini minng tidak seperti algoritma Apriori (terbatas) generasi dan tes, tetapi hanya pertumbuhan frequent pattern. Ketiga, FP-growth juga menerapkan metode divide-and-conquer berbasis partisi yang dapat mengurangi ukuran dari subsequent kondisional dan FP-trees kondisional.ned patterns in conditional databases, which

dramatically reduces the search space.