IMPLEMENTASI ALGORITMA *DATA MINING* FP-*GROWTH* DENGAN BAHASA PROGRAM FUNGSIONAL F#

I Gusti Agung Indrawan [1)] **Ni Kadek Ariasih**[2)]
Program Studi Teknik Informatika[1) 2)]
STMIK STIKOM Indonesia, Denpasar, Bali[1) 2)]
agung.indrawan@stiki-indonesia.ac.id [(1)] dekarik96@gmail.com [2)]

ABSTRACT

Retail businesses that use a point-of-sales (POS) system store large quantities of sales transaction data. Transaction data stored in this large amount has the potential to have unknown knowledge. An example is knowledge about items that are often purchased simultaneously in one transaction, which in data mining terms is called frequent itemset.

The frequent itemset can be found using the association rule discovery algorithm. The algorithms for finding association rules include the a priori algorithm and the frequent-pattern growth (FP-growth) algorithm. The FP-growth algorithm compresses sales transaction data that meet the criteria for frequent items into a data tree structure called the frequent-pattern tree (FP-tree), where the FP-tree data structure stores itemset association information (Han et al., 2011). The FP-growth algorithm performs frequent itemset discovery by performing tree traversal on the FP-tree data structure. Most of the execution time of the FP-growth algorithm is spent at this stage. Large shopping cart data has an impact on frequent itemset discovery times.

This study implements the FP-growth algorithm using the functional programming language F #. The implementation of the FP-growth algorithm using F # is expected to accelerate the discovery time of frequent itemset compared to the implementation of the FP-growth algorithm using the imperative programming language.

Keywords*: association rules, data mining, FP-growth, F #, frequent itemset, functional programming, shopping cart.*

ABSTRAK

Usaha retail yang menggunakan sistem *point-of-sales* (POS) menyimpan data transaksi penjualan barang dalam jumlah besar. Data transaksi yang tersimpan dalam jumlah besar ini berpotensi memiliki *knowledge* yang belum diketahui. Sebagai contoh adalah *knowledge* tentang barang yang sering dibeli secara bersamaan dalam satu transaksi, yang dalam istilah *data mining* disebut *frequent itemset*.

Frequent itemset dapat ditemukan menggunakan algoritma penemuan aturan asosiasi. Algoritma penemuan aturan asosiasi antara lain adalah algoritma apriori dan algoritma *frequent-pattern growth* (FP-*growth*). Algoritma FP-*growth* mengkompresi data transaksi penjualan yang memenuhi kriteria *frequent item* ke dalam struktur data *tree* yang disebut *frequent-pattern tree* (FP-*tree*), dimana struktur data FP-*tree* menyimpan informasi asosiasi *itemset* (Han dkk., 2011). Algoritma FP-*growth* melakukan penemuan *frequent itemset* dengan melakukan *tree traversal* pada struktur data FP-*tree*. Sebagian besar waktu eksekusi algoritma FP-*growth* dihabiskan pada tahap ini. Data keranjang belanja yang berukuran besar berdampak pada waktu penemuan *frequent itemset*.

Penelitian ini mengimplementasikan algoritma FP-*growth* menggunakan bahasa pemrograman fungsional F#. Implementasi algoritma FP-*growth* menggunakan F# diharapkan mempercepat waktu penemuan *frequent itemset* dibandingkan dengan implementasi algoritma FP-*growth* menggunakan bahasa program imperatif.

Kata kunci: aturan asosiasi, *data mining*, FP-*growth*, F#, *frequent itemset*, *functional programming*, keranjang belanja.

PENDAHULUAN

Usaha retail yang menggunakan sistem *point-of-sales* (POS) menyimpan data transaksi penjualan barang dalam jumlah besar. Paten dengan judul "*Point-Of-Sale System and Apparatus*" menjelaskan tentang *point-of-sales* terminal yang memiliki fitur penyimpanan dan pencarian transaksi penjualan [1]. Data transaksi yang tersimpan dalam jumlah besar ini berpotensi memiliki *knowledge* yang belum diketahui. Sebagai contoh adalah *knowledge* tentang barang yang sering dibeli secara bersamaan dalam satu transaksi, yang dalam istilah *data mining* disebut *frequent itemset*. *Knowledge* tersebut dapat digunakan oleh *retailer* untuk meningkatkan penjualan, misal dengan membuat promosi bundel barang-barang tertentu dan meletakkan barang yang sering dibeli secara berdekatan pada rak penjualan. *Frequent itemset* dapat ditemukan menggunakan algoritma penemuan aturan asosiasi. Algoritma penemuan aturan asosiasi antara lain adalah algoritma apriori dan algoritma *frequent-pattern growth* (FP-*growth*).

Algoritma FP-*growth* mengkompresi data transaksi penjualan yang memenuhi kriteria *frequent item* ke dalam struktur data *tree* yang disebut *frequent-pattern tree* (FP-*tree*), dimana struktur data FP-*tree* menyimpan informasi asosiasi *itemset* [2]. Algoritma FP-*growth* melakukan penemuan *frequent itemset* dengan melakukan *tree traversal* pada struktur data FP-*tree*, dan sebagian besar waktu eksekusi algoritma FP-*growth* dihabiskan pada tahap ini. Data keranjang belanja yang berukuran besar berdampak pada waktu penemuan *frequent itemset*.

Bahasa pemrograman yang umum digunakan (*mainstream*) adalah bahasa pemrograman imperatif seperti C dan PHP. Implementasi suatu algoritma secara paralel menggunakan bahasa pemrograman imperatif, memerlukan penggunaan *synchronization primitive* seperti *spinlock*, *barrier* dan *semaphore*. *Synchronization primitive* ini memastikan tiap *thread* pada program agar tersinkron dan tidak berebut ketika mengakses suatu *shared resources*. Selain bahasa pemrograman imperatif, terdapat paradigma lain bahasa pemrograman, yang disebut bahasa pemrograman fungsional. Bahasa pemrograman fungsional memiliki konsep *pure functions*, yaitu hasil eksekusi *functions* selalu sama ketika diberikan *arguments* yang sama, dan eksekusi *functions* tidak memiliki *side effects*.

Contoh bahasa pemrograman fungsional yang populer adalah Haskell dan F#. Haskell memisahkan kode program dengan *side-effects* dengan kode program lainnya [3]. Pemrograman paralel menggunakan bahasa program fungsional memungkinkan algoritma diimplementasikan secara *determinate*, yaitu ketika program yang dijalankan secara serial menghasilkan *output* yang benar, maka ketika program dijalankan secara paralel akan menghasilkan *output* yang benar juga [4]. Penelitian ini mengimplementasikan algoritma FP-*growth* menggunakan bahasa pemrograman fungsional F#. Implementasi algoritma FP-*growth* menggunakan F# diharapkan mempercepat waktu penemuan *frequent itemset* dibandingkan dengan implementasi algoritma FP-*growth* menggunakan bahasa program imperatif.

TINJAUAN PUSTAKA

Penelitian sebelumnya tentang implementasi algoritma penemuan aturan asosiasi menggunakan bahasa pemrograman fungsional telah dilakukan di tahun 2007 oleh Nittaya Kerdprasop dan Kittisak Kerdprasop diartikel yang berjudul "*Mining Frequent Patterns with Functional Programming*". Artikel tersebut meneliti tentang implementasi algoritma apriori menggunakan bahasa pemrograman fungsional Haskell. *Dataset* yang digunakan untuk testing implementasi algoritma diunduh dari *UC Irvine Machine Learning Database Repository* berupa empat *dataset* dengan nama *Vote*, *Chess*, DNA dan *Mushroom*. Testing dilakukan pada *notebook* dengan CPU AMD Athlon 796 MHz, RAM 512 MB dan HDD 40 GB. Kesimpulan yang didapat dari testing implementasi algoritma apriori yang diimplementasikan menggunakan Haskell dibandingkan dengan algoritma apriori yang diimplementasikan menggunakan Java adalah algoritma apriori bisa

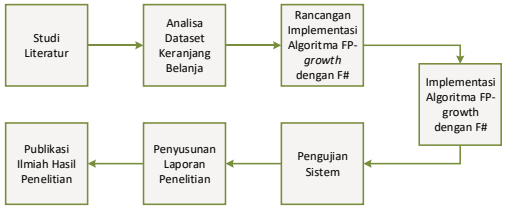
diimplementasikan hanya dengan 20 baris kode bahasa program Haskell. Implementasi algoritma apriori dengan Haskell juga lebih unggul dalam hal kecepatan eksekusi dan penggunaan *memory* dibandingkan implementasi algoritma apriori menggunakan Java [5].

Penelitian lain tentang implementasi algoritma penemuan aturan asosiasi menggunakan bahasa pemrograman fungsional dilakukan di tahun 2010 oleh Termier dkk diartikel berjudul "*HLCM: a first experiment on parallel data mining with Haskell*". Artikel tersebut meneliti implementasi algoritma *Linear Time Closed-itemset Miner* (LCM) menggunakan bahasa pemrograman fungsional Haskell. *Dataset* yang digunakan diunduh dari Universiteit Antwerpen *Frequent Itemset Mining Dataset Repository*. Testing dilakukan pada komputer dengan CPU 2x Intel Xeon 5520 2,26 GHz dengan RAM 24 GB. Kesimpulan yang didapat dari penelitian adalah implementasi algoritma LCM menggunakan bahasa pemrograman Haskell memerlukan 500 baris kode dibandingkan 3500 baris kode untuk versi paralel C++. Waktu eksekusi versi Haskell juga lebih lambat antara 6x-30x dibandingkan dengan versi paralel C++ pada *dataset* yang digunakan untuk testing [6].

METODE PENELITIAN

Alur Penelitian

Alur penelitian dapat dilihat pada Gambar 1.



Gambar 1. Alur Penelitian

Penelitian dilaksanakan dalam tujuh tahap, yaitu:

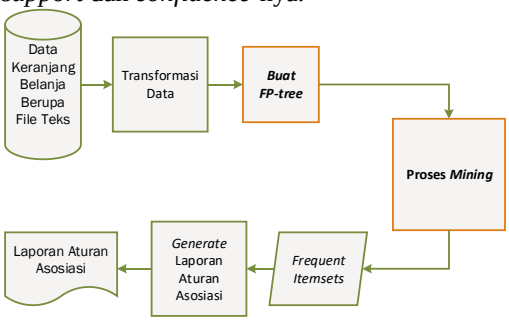
1. Studi literatur.
2. Analisa *dataset* keranjang belanja.
3. Rancangan implementasi algoritma fp-*growth* dengan F#.
4. Implementasi algoritma fp-*growth* dengan F#.
5. Pengujian sistem.
6. Penyusunan laporan penelitian.
7. Publikasi ilmiah hasil penelitian.

Dataset

Dataset keranjang belanja yang digunakan, adalah *dataset* contoh yang di-*input*-kan langsung melalui *interface command line* saat program dijalankan. Hal ini dikarenakan keterbatasan program dalam melakukan *loading dataset* eksternal.

Rancangan Sistem

Penelitian ini meneliti implementasi algoritma *FP-growth* menggunakan bahasa program fungsional F#. Gambar 2. menunjukkan gambaran umum sistem. Proses penemuan *frequent itemset* pada keranjang belanja dimulai dengan mentransformasi data keranjang belanja menjadi FP-*tree*. Setelah *FP-tree* terbentuk, tahap berikutnya adalah proses *mining frequent itemset. Frequent itemset* yang ditemukan kemudian dihitung *support* dan *confidence*-nya.



Gambar 2. Rancangan Sistem

HASIL DAN PEMBAHASAN

Algoritma FP-*growth* dalam F#

Inti algoritma *data mining* FP-*growth* yang diimplementasikan menggunakan bahasa program F#, yaitu tahapan *mining*, dapat dilihat pada Gambar 3.

```
let mineTree support (tree:Tree<_>) =
  let paths = tree.Root.Heads |> Seq.collect (fun kv->mineNode [] kv.Value)
paths |> Seq.collect (fun {Items=is;Support=c} ->
   let combos = allCombinations is |> List.map set
   combos |> List.map (fun combo -> combo,c))
|> Seq.filter (fun (combo,_) -> Set.isEmpty combo |> not)
|> Seq.groupBy fst
|> Seq.map (fun (k,vs)->k,vs|>Seq.sumBy snd)
|> Seq.filter (fun (_,s)-> s >= support)
```

Gambar 3. Algoritma FP-*growth* dalam F#

Pembahasan Coding

Baris 2: menemukan *support* untuk setiap *path* dari *root* menuju *leaf*. Baris 4: memasukkan semua kemungkinan kombinasi item di path, ke sebuah *list*. Baris 5: mengasosiasikan kombinasi item tersebut dengan *support* dari *path*. Baris 6: menghapus kombinasi yang kosong. Baris 7: melakukan *group-by* untuk setiap kombinasi item yang telah di-*generate* sebelumnya. Baris 8: menghitung jumlah *support* individual untuk setiap group menjadi total jumlah *support*. Baris 9: memfilter kombinasi item yang tidak memenuhi batasan *support* minimal yang ditentukan pengguna.

Pengujian

Algoritma FP-*growth* yang telah diimplementasikan, diuji menggunakan data contoh yang dapat dilihat pada Gambar 4.

```
#load "FPGrowth.fs"
open FPGrowth
let transactions =
  [|"abcefo"; "acg"; "ei"; "acdeg"; "acegl"; "ej"; "abcefp"; "acd"; "acegm"; "acegn"|]
  |> Array.mapi (fun i s-> i,s |> Seq.toList |> set)

let support = 3
let tree = makeTree transactions
let frequentItems = mineTree support tree |> Seq.toList
```

Gambar 4. *Loading Dataset*

Output yang dihasilkan dapat dilihat pada Gambar 5, dimana program menemukan kombinasi *frequent itemsets* berupa 1-*itemset*, 2-*itemsets*, dan 3-*itemsets* beserta *support*-nya berdasarkan *dataset* yang diberikan pada Gambar 4.

```
[(set ['a'], 8); (set ['a'; 'c'], 8); (set ['a'; 'c'; 'e'], 6);
 (set ['a'; 'e'], 6); (set ['c'], 8); (set ['c'; 'e'], 6); (set ['e'], 8);
 (set ['a'; 'c'; 'e'; 'g'], 4); (set ['a'; 'c'; 'g'], 4);
 (set ['a'; 'e'; 'g'], 4); (set ['a'; 'g'], 4); (set ['c'; 'e'; 'g'], 4);
 (set ['c'; 'g'], 4); (set ['e'; 'g'], 4); (set ['g'], 4)]
```

Gambar 5. *Output*

SIMPULAN

Algoritma *data mining* FP-*growth* dapat diimplementasikan menggunakan bahasa pemrograman fungsional F#. Implementasi dilakukan menggunakan software IDE Visual Studio 2019. Bahasa program F# yang digunakan adalah versi 4.1 (FSharp.Core, 4.4.3.0) dan dijalankan pada *runtime* .NET Framework 4.7.2.

Saran yang dapat diberikan berdasarkan hasil penelitian yang telah dilakukan adalah:

1. Algoritma FP-*growth* yang telah diimplementasikan dalam bahasa program F#, agar dikembangkan lebih lanjut, supaya proses *mining* bisa dilakukan secara paralel untuk mempercepat proses penemuan *frequent itemset*.
2. Dataset yang digunakan untuk pengujian adalah dataset yang berbasis transaksi *real-world* dan memiliki lebih dari 100.000 transaksi.

DAFTAR PUSTAKA

[1] W. M. Brobeck, J. S. Givins Jr., P. F. Meads Jr., and R. E. Thomas, "Point-of-sale system and apparatus," no. May 1966, 1976.

[2] J. Han, M. Kamber, and J. Pei, *Data mining: Concepts and Techniques*. 2011.

[3] A. S. Mena, *Beginning Haskell*. New York: Apress, 2014.

[4] K. Hammond and G. Michaelson, *Research Directions in Parallel Functional Programming*. London: Springer-Verlag, 1999.

[5] N. Kerdprasop and K. Kerdprasop, "Mining Frequent Patterns with Functional Programming," *Intern. J. Comput. Inf. Sci. Eng.*, pp. 282–287, 2007.

[6] A. Termier, N. Benjamin, S. Marlow, and S. Singh, "HLCM : a first experiment on parallel data mining with Haskell," *Symp. A Q. J. Mod. Foreign Lit.*, pp. 1–6, 2010.