ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

# DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN *TOP-K DOMINATING QUERIES* BERBASIS DATA *STREAMING* MENGGUNAKAN *INCOMPLETE* DATA


BAYU SEKTIAJI
NRP 05111440000122

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*

TUGAS AKHIR - KI141502

# DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN *TOP-K DOMINATING QUERIES* BERBASIS DATA *STREAMING* MENGGUNAKAN *INCOMPLETE* DATA

BAYU SEKTIAJI
NRP 05111440000122

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*


UNDERGRADUATE THESIS - KI141502

# DESIGN AND IMPLEMENTATION APPLICATION TOP-K DOMINATING QUERIES ON STREAMING DATA USING INCOMPLETE DATA

BAYU SEKTIAJI
NRP 05111440000122

Supervisor I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Supervisor II
Bagus Jati Santoso, S.Kom., Ph.D

Department of Informatics
Faculty of Information and Comunication Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*

# LEMBAR PENGESAHAN

## DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN *TOP-K DOMINATING QUERIES* BERBASIS DATA *STREAMING* MENGGUNAKAN *INCOMPLETE* DATA

**TUGAS AKHIR**

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
**BAYU SEKTIAJI**
**NRP: 05111440000122**

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D
NIP: 197708242006041001 ..... (Pembimbing 1)

Bagus Jati Santoso, S.Kom., Ph.D
NIP: 198611252018031001 ..... (Pembimbing 2)

**SURABAYA**
**Juni 2018**

*(Halaman ini sengaja dikosongkan)*

**DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN *TOP-K DOMINATING QUERIES* BERBASIS DATA *STREAMING* MENGGUNAKAN *INCOMPLETE* DATA**


| | | |
|---|---|---|
| **Nama** | : | **BAYU SEKTIAJI** |
| **NRP** | : | **05111440000122** |
| **Jurusan** | : | **Informatika FTIK-ITS** |
| **Pembimbing I** | : | **Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D** |
| **Pembimbing II** | : | **Bagus Jati Santoso, S.Kom., Ph.D** |


## Abstrak


*Decision support application merupakan aplikasi yang berperan penting dalam industri dan berbagia bidang lainnya. contoh dari decision support application seperti monitoring jaringan komputer, aplikasi analisa data ilmiah, aplikasi manajemen jaringan sensor, dan seterusnya. Dalam decision support application, top-k dominating query menjadi salah satu riset yang sedang dikembangkan akhir-akhir ini. Dengan metode ini memungkinkan untuk memperoleh k objek data yang memiliki skor dominasi tertinggi dari data yang ada.*

*Kebutuhan akan memonitor hasil query dengan sering adanya perubahan data (data stream) juga menjadi masalah tersendiri agar tidak melakukan perhitungan lagi dari awal. selain itu data pada kenyataannya tidak selalu lengkap (incomplete) sehingga tidak bisa melakukan query yang sama pada data lengkap.*

*Tugas akhir ini, mengusung permasalahan pengolahan top-k dominating query berbasis data streaming dengan menggunakan incomplete data. Diusulkan sebuah metode algoritme event based untuk menangani tantangan tersebut. Algoritme ini*

*diusulkan karena mempercepat waktu perhitungan pada setiap query-nya sehingga dapat memonitor top-k dominating query secara terus menurus dengan efisien. Dengan menggunakan data independen, anti corelated, dan forest covertype untuk uji coba metode ini terbukti memberikan hasil yang lebih baik dalam hal penggunaan waktu perhitungan dari pada algoritme naive dengan penurunan rata - rata waktu query sejumlah 77,87%.*

***Kata-Kunci**: Top-K Dominating Query, Incomplete Data, Streaming*

**DESIGN AND IMPLEMENTATION APPLICATION TOP-K DOMINATING QUERIES ON STREAMING DATA USING INCOMPLETE DATA**


| | | |
|---|---|---|
| **Name** | : | **BAYU SEKTIAJI** |
| **NRP** | : | **05111440000122** |
| **Major** | : | **Department of Informatics, Faculty of ICT - ITS** |
| **Supervisor I** | : | **Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D** |
| **Supervisor II** | : | **Bagus Jati Santoso, S.Kom., Ph.D** |


## Abstract


*Decision support aplication is play an important role in industrial field and many other field. the example of decision support application are computer network monitoring, scientific data analysis aplicatin management sensor network, and etc. In decision support application, top-k dominating query become one of research topic that still in development now days. With this method allow to gek k object which is has highest dominating score for all data.*

*Requirement to monitoring the results of query with frequent data changes (data stream) also be an problem itself so that not to recompute from scratch. Another problem, data in the reality is not always complete so it can't query like complete data.*

*In this thesis, raise a problem of streaming top-k dominating query using incomplete data. Event based algorithm is porposed to solve that problem with efficient. Using data independen, anti corelated, and forest covertype this method is proven gave better result in computation time than naive algorithm with reduce 77.87% the means query time.*



***Keywords**: Top-K Dominating Query, Incomplete Data,*

*Streaming*

# KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN TOP-K DOMINATING QUERIES BERBASIS DATA STREAMING MENGGUNAKAN INCOMPLETE DATA**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Ibu, Ayah , Mbak Puspa, dan keluarga besar penulis yang selalu memberikan dukungan baik berupa doa, motivasi, dan material yang tak dapat terhitung lagi kepada penulis, sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Bapak Bagus Jati Santoso, S.Kom., Ph.D dan Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D selaku dosen pembimbing penulis yang telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Bapak Dr.Eng. Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Jurusan Teknik Informatika ITS pada masa

pengerjaan Tugas Akhir, Bapak Radityo Anggoro, S.Kom., M.Sc., selaku koordinator TA, dan segenap dosen Teknik Informatika yang telah memberikan ilmu dan pengalamannya.

5. Teman - teman Laboratorium Pemrograman (Nafiar, Hari, Rina, Afiif, Dita, Nobby, Raca, Yuuta, Brian, Irsyad, Rizka, Bonbon) yang telah menemani, memberikan motivasi, saran - saran, dan hiburan saat penulis sedang jenuh mengerjakan Tugas Akhir ini.
6. Teman - teman satu perjuangan Ovan, Syukron, Wira, Toska, Dwika, dan Cahya yang telah bertukar pikiran dengan penulis dalam hal teknis maupun penulisan Tugas Akhir.
7. Mas Thiar dan Fatih yang sering memberikan saran dan pertimbangan dalam memilih jalan hidup pada penulis.
8. Warkop x Jojoran, Theredstc yang sering menemani futsal dan nonton bareng disaat penulis jenuh mengerjakan TA.
9. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2018

Bayu Sektiaji

# DAFTAR ISI

*(Halaman ini sengaja dikosongkan)*

# DAFTAR TABEL

# DAFTAR GAMBAR

# DAFTAR KODE SUMBER

*(Halaman ini sengaja dikosongkan)*

# BAB I

# PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, rumusan masalah, batasan masalahan, tujuan, manfaat, metodologi, dan sistematika penulisan laporan tugas akhir.

## 1.1 Latar Belakang

*Top - K Dominating (TKD) query* merupakan salah satu *query* yang saat ini memiliki peran penting pada banyak *decision support application*. *TKD query* merupakan gabungan antara *skyline* dan *top - k query*[2]. *Query* ini akan mengembalikan sebanyak $k$ object yang memiliki dominasi tertinggi pada suatu *dataset* yang diberikan [3].

Dewasa ini banyak *decision support application* yang mengadopsi basis *streaming* seperti aplikasi *monitoring* jaringan komputer, aplikasi analisa data ilmiah, aplikasi manajemen jaringan sensor, dan seterusnya. Aplikasi-aplikasi tersebut akan sangat terbantu dalam melakukan analisanya jika dapat menerapkan *TKD query*. Penerapan *TKD query* pada basis *streaming* memiliki permasalahan tersendiri. Dimana pada basis *streaming*, perubahan pada data sangat sering terjadi[2]. Hal ini menyebabkan permasalahan komputasi yang sangat tinggi jika hanya melakukan *requery* saat terjadi perubahan pada data.

Suatu data pada kenyataannya tidak selalu lengkap atau dapat disebut *incomplete* data. Hal ini biasanya disebabkan oleh beberapa hal seperti kesalahan pada perangkat, data privasi, dan lain sebagainya[1]. *Incomplete* data ini berpengaruh pada algoritme yang diterapkan pada *TKD query* karena *incomplete* data memiliki relasi dominasi yang unik. Pada *complete* data dapat dipastikan jika $O_1$ mendominasi $O_2$ dan $O_2$ mendominasi $O_3$ maka dapat disimpulkan bahwa $O_1$ mendominasi $O_3$. Hal ini berbeda pada *incomplete* data jika $O_1$ mendominasi $O_2$ dan $O_2$

mendominasi $O_3$ maka $O_1$ belum tentu mendominasi $O_3$ dikarenakan *incomplete* data dapat memiliki relasi dominasi *non-transitif* [4].

Dengan adanya beberapa permasalahan di atas maka dari itu dibutuhkan struktur data dan algoritme yang tepat untuk menangani permasalah *top – k dominating queries* berbasis *streaming* menggunakan *incomplete* data. Pada tugas akhir ini akan mengangkat permasalahan *top-k dominating queries* berbasis *streaming* menggunakan *incomplete* data sehingga hasil dari tugas akhir ini diharapkan dapat menentukan implementasi algoritme dan struktur data yang tepat untuk memecahkan permasalahan di atas secara optimal dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

## 1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana menentukan algoritme dan struktur data yang tepat untuk menyelesaikan permasalahan Pengolahan *Top – K Dominating Queries* Berbasis *Data Streaming* Menggunakan *Incomplete Data*?
2. Bagaimana mengimplementasikan algoritme dan struktur data yang dibangun pada permasalahan Pengolahan *Top – K Dominating Queries* Berbasis *Data Streaming* Menggunakan *Incomplete Data*?
3. Bagaimana hasil dari kinerja algorithme dan struktur data yang dibangun pada permasalahan Pengolahan *Top – K Dominating Queries* Berbasis *Data Streaming* Menggunakan *Incomplete Data*?

## 1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Nilai atribut yang akan diproses dalam algoritme ini bertipe numerik.
2. Implementasi dilakukan dengan bahasa pemrograman python.
3. Dataset yang digunakan data *real-life* dan sintetis.
4. Dataset yang digunakan adalah data yang tidak lengkap (*incomplete data*).
5. Konsep data streaming yang digunakan adalah menggukan konsep *sliding window* dengan tipe *count based*.

## 1.4 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Menentukan algoritme dan struktur data yang tepat untuk menyelesaikan permasalahan Pengolahan *Top – K Dominating Queries* Berbasis *Data Streaming* Menggunakan *Incomplete Data*.
2. Melakukan implementasikan algoritme dan struktur data yang dibangun pada permasalahan Pengolahan *Top – K Dominating Queries* Berbasis *Data Streaming* Menggunakan *Incomplete Data*.
3. Mengevaluasi hasil dan kinerja algoritme dan struktur data yang telah dirancang untuk permasalahan Pengolahan *Top – K Dominating Queries* Berbasis *Data Streaming* Menggunakan *Incomplete Data*.

## 1.5 Manfaat

Tugas akhir ini diharapkan dapat membantu memberikan kontibusi dalam perkembangan *decision support application*

yang mengunakan *top-k dominating query* sehingga manusia atau program lain untuk menentukan pilihan tanpa melihat data secara keseluruhan. Tugas akhir ini juga diharapkan dapat membantu dan memberikan rujukan untuk mengolah data *incomplete* data atau juga metode pengolahan *streaming* yang marak dewasa ini karena perkembangan data semakin pesat. Disamping itu tugas akhir ini juga diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

## 1.6 Metodologi

Tahap-tahap yang dilakukan dalam pengerjaan tugas akhir ini adalah:

1. Studi Literatur
   Pada tahap ini dilakukan pencarian informasi dan studi literatur yang diperlukan untuk penyelesaian persoalan yang akan dikerjakan. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritme yang digunakan dalam pengerjaan tugas akhir ini, materi-materi tersebut didapatkan dari paper, internet maupun buku acuan.
2. Design Perangkat Lunak
   Pada tahap ini, penulis akan mendesain beberapa algoritme pengolahan *Top – K Dominating queries* berbasis *streaming* menggunakan *incomplete* data.
3. Implementasi Perangkat Lunak
   Pada tahap ini dilakukan implementasi dari desain algoritme pengolahan *Top – K Dominating query* berbasis *streaming* menggunakan *incomplete* data. Implementasi ini dilakukan dengan menggunakan bahasa pemrograman python.
4. Uji Coba dan Evaluasi
   Pada tahap ini dilakukan uji coba dengan menggunakan beberapa *dataset* yang sudah disiapkan. Pengujian yang

akan dilakukan adalah pengujian *performance*. Untuk pengujian *performance* dilakukan pemantauan efek perkembangan jumlah *k*, *windows size*, dan dimensi terhadap perkembangan waktu rata-rata eksekusi satu *query* dan jumlah *memory* yang dibutuhkan.

## 1.7 Sistematika Penulisan

Buku Tugas Akhir ini memiliki tujuan untuk mendapat gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna untuk pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut:

**Bab I Pendahuluan**

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan tugas akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

**Bab II Tinjauan Pustaka**

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan tugas akhir ini.

**Bab III Perancangan**

Bab ini berisi tentang penjelasan umum permasalahan *Top-K Dominating query* berbasis data *stream* mengunakan *incomplete data* serta perancangan desain algoritme dalam menyelesaikan permasalahan tersebut.

**Bab IV Implementasi**

Bab ini membahas implementasi dari desain algoritme yang telah dibuat pada bab sebelumnya. Penjelasan berupa kode yang digunakan untuk proses implementasi.

**Bab V Pengujian dan Evaluasi**

Bab ini membahas tahap-tahap uji coba. Kemudian hasil uji

coba dievaluasi untuk kenerja dari aplikasi yang dibangun.

**Bab VI Penutup**

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba dilakukan dan saran untuk pengembangan aplikasi ke depannya.

# BAB II

# TINJAUAN PUSTAKA

Pada bab ini akan dipaparkan mengenai dasar - dasar teori yang akan digunakan pada tugas akhir ini. Untuk lebih mudah dalam memahami diberikan simbol beserta keterangan yang digunakan pada **Tabel 2.1**.

**Tabel 2.1:** Daftar simbol dan keterangan bab II

| Simbol | Keterangan |
|---|---|
| $S$ | Set objek yang aktif |
| $n$ | Jumlah point yang aktif ( $n = \|S\|$) |
| $d$ | Jumlah dimensi |
| $d_j$ | dimensi urutan ke-$j$ |
| $o_i$ | objek urutan ke-$i$ |
| $o_{i,j}$ | objek urutan ke-$i$ dimensi ke-$j$ |
| $o_i.arr$ | waktu objek $o$ masuk |
| $o_i.exp$ | waktu objek $o$ keluar |
| $score(o_i)$ | skor pada objek $o$ |
| $TOPK$ | set k point yang memiliki skor tertinggi dari pada objek yang lain |
| $k$ | jumlah objek pada hasil ( $k = \|TOPK\|$ ) |
| $now$ | waktu sekarang |

## 2.1 *Top-K Dominating Query*

*Top-K Dominating (TKD) query* merupakan gabungan antara dua *preference-based query processing* yang paling banyak digunakan yaitu *Top-K query* dan *Skyline query*. *TKD query* mengunakan fungsi ranking untuk merangking objek seperti *Top-K query* dan mengunakan *dominating relationship* seperti *Skyline query*[2].

Tujuan *TKD query* adalah mempertahankan keuntungan dan mengeliminasi keterbatasan dari *Top-K* dan *Skyline query*. Oleh

karena itu *TKD query* memiliki beberapa sifat yang diinginkan seperti berikut:

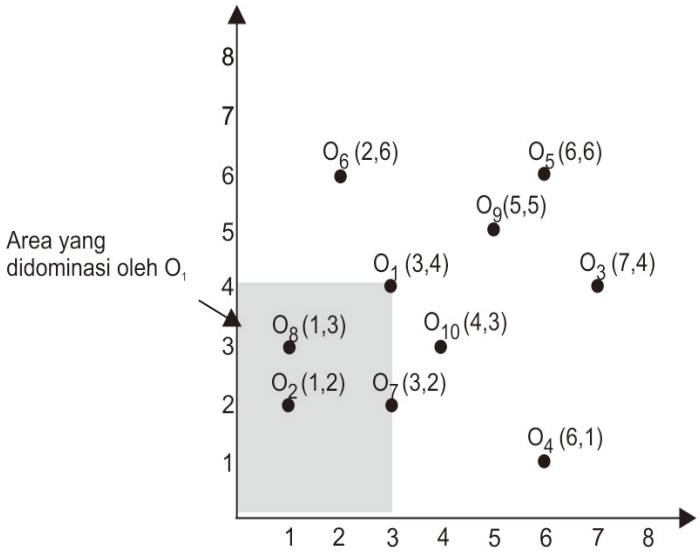
1. Jumlah dari hasilnya terkontrol.
2. Hasil tidak berubah.
3. Tidak perlunya definisi skoring dari user.
4. Setiap objek akan memiki skor yang akan menentukan rankingnya.

*TKD query* dapat didefinisikan sebagai sebuah *query* yang akan mengembalikan sebanyak $K$ objek yang memiliki skor dominasi tertinggi pada suatu *dataset* yang diberikan[3]. Penentuan skor pada objek dilakukan dengan cara menghitung jumlah objek lain yang didominasi oleh objek tersebut. Dalam hal ini yang dimaksud dengan dominasi adalah sebagai berikut. Diberikan dua buah objek yaitu $o$ dan $o$'. $o$ dapat dikatakan mendominasi $o$' yang disimbolkan dengan $o \prec o'$ jika dua kondisi ini terpenuhi. Kondisi yang pertama pada setiap dimensi $i$ pada objek $o$ tidak ada yang lebih besar dari dimensi $i$ pada objek $o$'. Kondisi yang kedua terdapat minimal satu dimensi pada objek $o$ yang lebih kecil dari pada objek $o$'. Dari penjelasan sebelumnya *Top-K Dominating (TKD) query* memiliki beberapa definisi formal sebagai berikut:

- **Definisi 1 (*Dominasi*).** Diberikan dua buah objek yaitu $o$ dan $o$'. $o$ dapat dikatan mendominasi $o$' yang disimbolkan dengan $o \prec o'$ jika dua kondisi berikut terpenuhi: 1) Pada setiap dimensi $i$ pada objek $o$ tidak ada yang lebih kecil dari dimensi $i$ pada objek $o$', 2) Terdapat minimal satu dimensi pada objek $o$ yang lebih besar dari pada objek $o$'.

- **Definisi 2 (*Skor*).** Skor dari sebuah objek $o$ (disimbolkan dengan $score(o)$) adalah jumlah dari objek $o' \in S - \{o\}$ yang didominasi oleh $o$ . Dapat di simbolkan $score(o) = |\{o' \in S - \{o\}\}| o \prec o'$.

- **Definisi 3 (*Top-k dominating query*).** Diberikan sebuah dataset $S$, *Top-k Dominating query* pada $S$ akan menghasilkan $S_G \subseteq S$ dari K objek dengan skor tertinggi.




**Gambar 2.1:** Dominasi pada complete data

**Tabel 2.2:** Skor dari data pada gambar 2.1

| $O$ | $o_1$ | $o_2$ | $o_3$ | $o_4$ | $o_5$ | $o_6$ | $o_7$ | $o_8$ | $o_9$ | $o_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| score(O) | 3 | 0 | 6 | 0 | 8 | 2 | 1 | 1 | 5 | 3 |

Untuk lebih jelas padat dilihat contoh pada Gambar 2.1. Terdapat sepuluh objek $o_1$, $o_2$, …, $o_9$, $o_{10}$ dengan dua dimensi yaitu $d_1$ dan $d_2$. Dari definisi 1 dapat dilihat bahwa $o_1(3, 4)$

mendominasi $o_8(1,3)$, $o_2(1,2)$ dan $o_7(3,2)$. Karena pada setiap dimensi pada $o_1(3,4)$ tidak ada yang lebih kecil dari $o_7(3,2)$ dan $o_1$ memiliki minimal satu dimensi yang lebih besar daripada $o_7$ yaitu pada $d_2$. Sama juga $o_1$ mendominasi $o_8(1,3)$ dikarenakan pada setiap dimensi pada $o_1(3,4)$ tidak ada yang lebih kecil dari $o_8(1,3)$ dan $o_1$ memiliki minimal satu dimensi yang lebih besar daripada $o_8$. Dalam hal ini, $o_1$ memiliki dua dimensi yang lebih besar dari pada $o_8$ yaitu $d_1$ dan $d_2$, dan seterusnya. Dengan mengacu pada definisi 2 skor pada $o_1$ adalah 3 karena $o_1$ mendominasi $o_8$, $o_2$ dan $o_7$. Untuk semua perhitungan skor dapat dilihat pada Tabel 2.2. Setelah mendapat Tabel 2.2 dengan melihat penjelasan pada Definisi 3 jika $k = 2$ maka keluran dari *TKD query* adalah $p_5$ dengan skor 8 dan $p_3$ dengan skor 6.

## 2.2 *Incomplete Data*

*Incomplete data* atau data yang tidak lengkap merupakan sebuah jenis data yang sering terjadi pada banyak kasus nyata. Dalam kasus ini sebuah data kehilangan nilai pada beberapa dimensinya. Ada beberapa sebab yang membuat data menjadi *incomplete* contoh: 1) Karena kerusakan pada *device*, 2) Sebuah data privasi yang hanya boleh dilihat oleh orang yang berkepentingan, 3) *Data loss*, dan seterusnya. [1].

Pada Tabel 2.3 merupakan sebuah contoh nyata data yang tidak lengkap. Data pada Tabel 2.3 merupakan *rating* empat film $m_1$, $m_2$, $m_3$, $m_4$ oleh lima orang penguna $a_1$, $a_2$, $a_3$, $a_4$, $a_5$. Data tidak lengkap karena setiap orang memberikan *rating* pada film yang dia kenal saja. Sedangkan beberapa orang tidak kenal semua film.

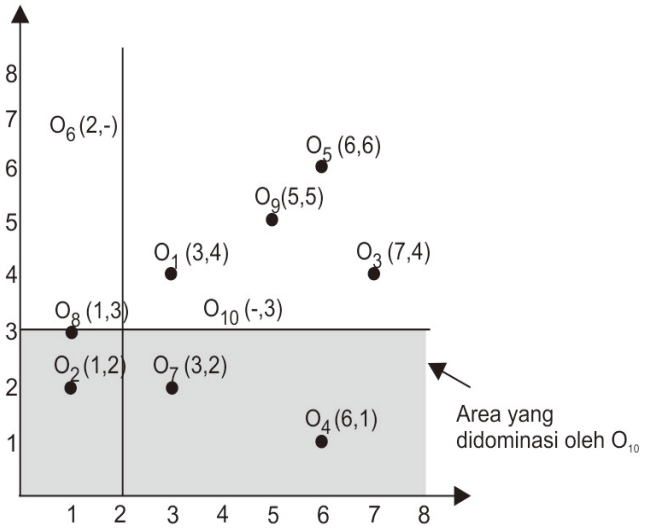
**Tabel 2.3:** Contoh *incomplete* data [1]

| **ID** | **Nama Film** | ***Rating* Film dari *Audience*** | | | | |
|---|---|---|---|---|---|---|
| | | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| $m_1$ | Schindler's List (1993) | - | - | 3 | 4 | 2 |
| $m_2$ | The Godfather (1972) | 5 | 3 | 4 | - | - |
| $m_3$ | The Silence of the Lamps (1991) | - | 2 | 1 | 5 | 3 |
| $m_4$ | Star Wars (1977) | 3 | 1 | 5 | 3 | 4 |

Dalam penerapat *TKD* pada *incomplete data* terdapat perbedaan dari pada *complete data*. Karena ada data yang *missing* sehingga definisi dari dominasi berubah menjadi seperti berikut:

- **Definisi 4 (Dominasi pada *incomplete data*).** Diberikan dua buah objek yaitu $o$ dan $o$'. $o$ dapat dikatan mendominasi $o$' yang disimbolkan dengan $o \prec o'$ jika dua kondisi berikut terpenuhi: 1) Pada setiap dimensi $i$ pada objek $o$ tidak ada yang lebih kecil dari dimensi $i$ pada objek $o$' atau minimal salah satu *missing*, 2) Terdapat minimal satu dimensi pada objek $o$ yang lebih besar dari pada objek $o$'.

Perbedaan yang paling mencolok akibat pergantian definisi ini adalah relasi dominasi pada *complete data* merupakan *transitif* sedangkan pada *incomplete data* relasi dominasi adalah *non-transitif*. Yang dimaksut dengan relasi dominasi *transitif* yaitu jika $a \prec b$ dan $b \prec c$ maka $a \prec c$. Sedangkan dalam *incomplete data* jika $a \prec b$ dan $b \prec c$ maka tidak selalu $a \prec c$.




**Gambar 2.2:** Dominasi pada *incomplete* data

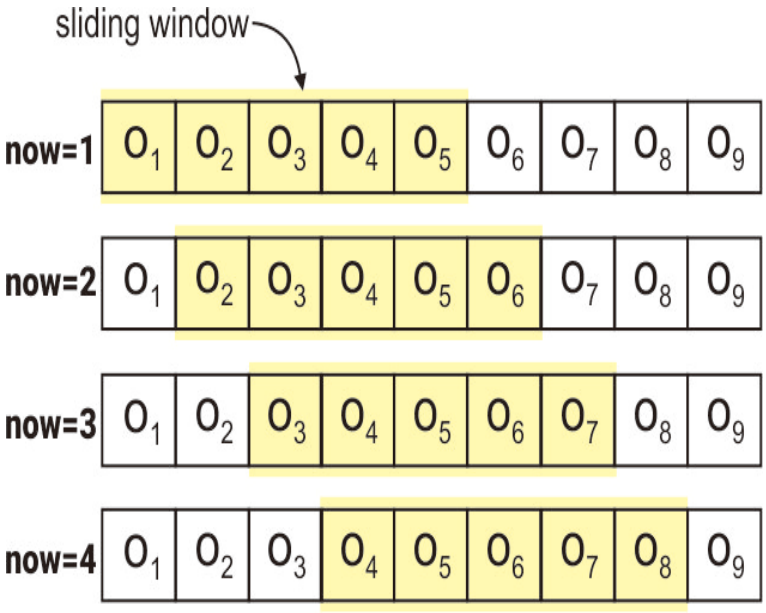
Pada Gambar 2.2 terdapat sepuluh objek $o_1$, $o_2$, …, $o_9$, $o_{10}$. dua diantara sepuluh objek tersebut meiliki *missing data* yang disimbolkan dengan simbol $-$. Menurut Definisi 4 $o_{10}(-,3)$ mendominasi $o_7(3,2)$ karena pada tiap dimensi $o_{10}$ tidak ada yang lebih besar dari $o_7$ atau minimal salah satu data *missing* dan terdapat minimal satu dimensi $o_{10}$ yang lebih besar dari $o_7$ yaitu pada dimensi $d_2$, $o_7(3,2)$ mendominasi $o_6(2,1)$ dengan alasan yang sama dengan $o_{10}$ mendominasi $o_7$. Maka dapat disimbolkan menjadi $o_7 \prec o_6$ dan $o_{10} \prec o_7$ tetapi $o_{10}(-,3)$ tidak mendominasi $o_6(2,-)$ karena di dimensi $d_1$ dan $d_2$, tidak ada yang saling mendominasi. Hal ini yang dimanakan dengan *non-transitif*.

## 2.3 Data *Stream*

Data *streaming* adalah suatu model data pada suatu aplikasi yang pada setiap periodenya melakukan *query* untuk me-*refresh* hasil dari *query* yang ditampilkan pada penggunanya karena data pada aplikasi ini sering berubah. Contoh aplikasi : pemantauan jaring komputer, analisis data ilmiah, manajemen data pada jaringan sensor, dan seterusnya[2]. Pada model ini sebuah objek memiliki dua *property* yang harus ada yaitu waktu objek datang yang dapat disimbolkan dengan $o.arr$ dan waktu objek keluar yang dapat disimbolkan dengan $o.exp$. Dalam penerapannnya pada pemograman data *streaming* dapat di modelkan menjadi sebuah *sliding window*.

## 2.4 *Sliding Window*

*Sliding Window* adalah sebuah teknik dalam pemrograman yang memodelkan data seperti sudut pandang kaca di dalam sebuah bis. Hal ini dapat dilihat pada **Gambar 2.3**. Terdapat dua tipe *sliding window* yang pertama adalah *count based sliding window* dimana $k$ selalu konstan. Jika ada sebanyak $r$ objek masuk menjadi objek aktif maka akan ada juga sebanyak $r$ objek aktif yang akan keluar. Tipe yang kedua adalah *time-based sliding window* dimana jumlah $n$ tidak konstan. Pada tipe ini setiap objek memiliki waktu aktif masing-masing yang tidak saling terkaitan. Sehingga jumlah objek yang aktif bisa berbeda pada waktu yang berbeda.




**Gambar 2.3:** contoh *count-based sliding window*

## 2.5 Python

Python adalah bahasa pemrograman interpretatif multiguna dengan prinsip agar sumber kode yang dihasilkan memiliki tingkat keterbacaan yang baik[5]. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. Python mendukung beragam paradigma pemrograman, seperti pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi. Salah satu fitur yang tersedia pada Python adalah sebagai bahasa pemrograman

dinamis yang dilengkapi dengan manajemen memori otomatis.

## 2.6 Flask

Flask adalah sebuah kerangka kerja web. Artinya, Flask menyediakan perangkat, pustaka, dan teknologi yang memungkinkan seorang pengembang untuk membangun aplikasi berbasis web [6]. Aplikasi web yang bisa dibangun bisa berupa sebuah halaman web, blog, wiki, bahkan untuk web komersial. Flask dibangun berbasiskan pada Werkzeug, Jinja 2, dan MarkupSafe yang mana menggunakan bahasa pemrograman Python sebagai basisnya. Flask sendiri pertama kali dikembangkan pada tahun 2010 dan didistribusikan dengan lisensi BSD.

Flask termasuk sebagai perangkat kerja mikro karena tidak membutuhkan banyak perangkat atau pustaka tertentu agar bisa bekerja. Flask tidak menyediakan fungsi untuk melakukan interaksi dengan basis data, tidak mempunya validasi *form* atau fungsi lain yang umumnya bisa digunakan dan disediakan pada sebuah kerangka kerja web. Meskipun memiliki kemampuan yang minim, tapi Flask mendukung dan memberikan kemudahan bagi pengembang untuk menambahkan pustaka sendiri untuk mendukung aplikasinya. Berbagai pustaka seperti validasi *form*, mengunggah *file*, berbagai macam teknologi autentifikasi bisa digunakan dan tersedia untuk Flask. Bahkan pustaka-pustaka pendukung tersebut lebih sering diperbarui dibandingkan dengan Flasknya sendiri.

## 2.7 React

React adalah sebuah pustaka Javascript yang digunakan untuk membuat *user interface*. dengan React dapat mempermudah membuat situs web dengan data yang dinamis

tanpa melakukan *reload* setiap kali ada perubahan. karena React hanya melakukan *render* ulang pada komponen dengan *state* yang berubah saja menjadikan React efisien dalam melakukan memperbarui tampilan pada *client*.

## 2.8 Socketio

Socketio merupakan sebuah pustaka untuk membuat aplikasi situs web *real-time*. Pustaka ini membuat sebuah komunikasi *real-time* dua arah antara *client* dan *server*. Socketio menyediakan beberapa fitur yaitu *broadcasting* kepada *multiple sockets*, penyimpanan data yang terasosiasi dengan setiap *client*, dan *asynchronous I/O* sehingga Socketio memiliki kemampuan untuk mengimplementasikan sistem analisa *real-time*, *binary streaming*, dan pesan instan.

# BAB III

# PERANCANGAN

Bab perancangan berisi deskripsi umum permasalahan dan spesifikasi kebutuhan sistem yang akan menjadi dasar perancangan algoritme *query* dalam menyelesaikan *top-k dominating query* berbasis data *streaming* mengunakan *incomplete* data. Dalam bab ini juga akan dilakukan perancangan beberapa algoritme yang diajukan untuk menyelesaikan permasalahan tersebut. Untuk memudahkan dalam memahami *psudocode* dibuat daftar simbol dan keterangan pada **Tabel 3.1**.

**Tabel 3.1:** Daftar simbol dan keterangan bab III

| Simbol | Keterangan |
|---|---|
| $o_{in}$ | Objek yang masuk pada *sliding window* |
| $o_{out}$ | Objek yang keluar pada *sliding window* |
| $o.parent$ | Jumlah objek yang mendominasi objek o |
| $e$ | *Event* |
| $e.egt$ | *Event generating time* |
| $e.ept$ | *Event processing time* |
| $WINDOW$ | *Sliding window berupa qeueu* |
| $DELETEITEM$ | *Queue* dari objek yang telah *expired* |
| $EVENTQUEUE$ | *Priority queue* untuk menyipan event |

## 3.1 Deskripsi Umum

Aplikasi yang akan dibangun pada tugas akhir ini yaitu aplikasi yang dapat melakukan *top-k dominating query* berbasis data *stream* dengan menggunakan *incomplete* data. Pada subbab-subbab berikutnya akan dijelaskan tetang beberapa konsep yang dapat membantu dalam memahami permasakahan ini.

### 3.1.1 Dominasi

Diberikan dua buah objek yaitu $o$ dan $o$'. $o$ dapat dikatakan mendominasi $o$' yang disimbolkan dengan $o \prec o'$ jika dua kondisi berikut terpenuhi: 1) Pada setiap dimensi ke-$i$ pada objek $o$ tidak ada yang lebih kecil dari dimensi ke-$i$ pada objek $o$', 2) Terdapat minimal satu dimensi pada objek $o$ yang lebih besar dari pada objek $o$'.

### 3.1.2 Dominasi pada *Incomplete Data*

Diberikan dua buah objek yaitu $o$ dan $o$'. $o$ dapat dikatan mendominasi $o$' yang disimbolkan dengan $o \prec o'$ jika dua kondisi berikut terpenuhi: 1) Pada setiap dimensi ke-$i$ pada objek $o$ tidak ada yang lebih besar dari dimensi ke-$i$ pada objek $o$' atau minimal salah satu objek $o$ atau $o'$ kehilangan dimensi ke-$i$, 2) Terdapat minimal satu dimensi pada objek $o$ yang lebih besar dari pada objek $o$'.

### 3.1.3 Skor Dominasi

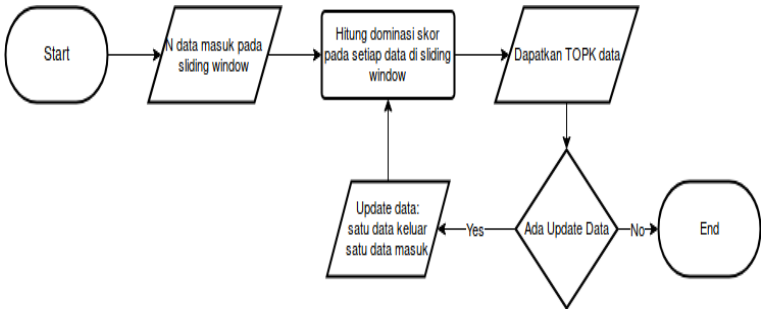
Skor dominasi dari sebuah objek $o$ disimbolkan dengan $score(o)$. Skor dominasi adalah jumlah dari objek dalam sliding window ($o' \in S - \{o\}$) yang didominasi oleh $o$ . Hal ini dapat dirumuskan sebagai berikut $score(o) = |\{o' \in S - \{o\}\}| o \prec o'$.

### 3.1.4 *Top-k dominating query*

Diberikan sebuah dataset $S$ yang masuk dalam sliding window, *Top-k Dominating query* pada $S$ akan menghasilkan $TOPK \subseteq S$. Dimana $TOPK$ merukan kumpulan objek yang

memiliki $k$ skor dominasi tertinggi.

### 3.1.5 Alur *Top-K Dominating Query* Berbasis Data *Streaming* Mengunakan *Incomplete* data




**Gambar 3.1:** Diagram alur deskripsi umum sistem

Pada permulaannya sistem akan menerima sejumlah $N$ data dengan $d$ dimensi yang tidak lengkap secara langsung. $N$ data tersebut akan menjadi data awal pada *sliding window*. Dari setiap data pada *sliding window* akan dilakukan pencarian skor dominasi. Berikutnya sistem akan mengeluarkan TOPK.

Setelah tahap permulaan selesai sistem akan mengalami perubahan data dimana setiap kali perubahan akan ada satu data yang keluar dari sliding window dan ada satu data yang masuk ke sliding window. Sistem akan melakukan perhitungan dominasi skor pada *sliding window* dan mengeluarkan sejumlah data yang memiliki $k$ dominasi skor tertinggi.

## 3.2 Spesifikasi Kebutuhan Sistem

Pada aplikasi *Top-K Dominating query* berbasis data *streaming* dengan menggunakan *incomplete* data dibutuhkan

beberapa fungsi untuk dapat memenuhi kebutuhan. berikut adalah fungsi - fungsi yang dibutuhkan:

1. Pengolahan data awal yang masuk pada *sliding window* (*initialize data*)
   Pada bagian ini data sejumlah panjang *sliding window* akan dimasukan secara langsung lalu dilakukan pembentukan stuktur data yang dapat meningkatkan *performance* pada saat pengolahan perubahan data.
2. Pengolahan pada perubahan data(*update data*)
   Pada bagian ini data yang masa aktifnya sudah selesai (*expired*) akan dikeluarkan dari *sliding window* sedangkan akan ada satu data yang masuk kedalam *sliding window* untuk mengantikan data yang *expired*.
3. Mendapat hasil *Top-K Dominating query*
   Pada bagian ini hasil *TOPK* akan dikeluarkan dari *sliding window*.

## 3.3 Perancangan Algoritme

Pada bagian perancangan algoritme akan dijelaskan tentang beberapa algoritme yang diajukan untuk melakukan pengolahan *Top-K Dominating query* berbasis data *stream* dengan mengunakan *incomplete* data. Terdapat dua algoritme yang diajukan yaitu *Naive* dan *Event based*

### 3.3.1 Algoritme *Naive*

. *Naive* adalah sebuah algoritme simple dan mudah untuk diimplementasikan. Algoritme *naive* bertujuan untuk menyelesaikan suatu persoalan dengan tepat dengan mengesampingkan *efficiency*. Pada algoritme *naive* tidak ada struktur data yang dibentuk.

#### 3.3.1.1 Pengolahan Data Awal Masuk pada Algoritme *Naive*

Pada kasus ini, saat *initialize* data algoritme *naive* melakukan perhitungan dengan cara membandingkan objek $o$ dengan objek lainnya $o'$. Jika objek $o$ itu mendominasi objek lainnya $o'$ maka skor dominasi $o$ bertambah 1 poin. Hal ini berlaku sebaliknya jika $o'$ mendominasi objek $o$ maka skor dominasi objek $o'$ bertambah 1 poin. **Gambar 3.2** adalah rancangan algoritme *intialize* data pada algoritme *naive*.

**Algorithm 1:** *Initalize Data pada Naive Algorithm*

```
   Input: K, WINDOW, Dimension
   Output: TOPK
1  Initialize all object score in WINDOW to zero;
2  dataLengt ← |WINDOW|;
3  for i ← 0 to dataLenght do
4      for j ← i to dataLenght do
5          o ← get which is more dominant WINDOW[i] or
             WINDOW[j];
6          score(o)+ = 1;
7      end
8  end
9  SORTEDWINDOW ← sort(WINDOW);
10 TOPK ← getTopK(SORTEDWINDOW);
```

**Gambar 3.2:** *Initalize Data pada Naive Algorithm*

#### 3.3.1.2 *Update* Data Keluar dan Data Masuk pada Algoritme *Naive*

Saat *update data*, satu objek keluar $o_{out}$ akan dibandingkan pada semua data yang ada pada *sliding window* dan mengurangi 1 poin skor dominasi jika objek pada *sliding window* mendominasi objek yang keluar. Sedangkan untuk objek masuk $o_{in}$ akan dibandingkan sama seluruh *sliding window* untuk mencari skor objek yang masuk dan mengupdate skor pada

seluruh objek yang ada pada *sliding window*. **Gambar 3.3** adalah algoritme *Naive* untuk kasus *Top-K Dominating query* berbasis *streaming* dengan mengunakan *incomplete* data.

**Algorithm 2:** Update Data *Naive Algorithm*

**Input:** $o_{in}$, $o_{ont}$
**Output:** TOPK
**for** $i \leftarrow 0$ **to** $|WINDOW|$ **do**
    $o \leftarrow$ get which is more dominant WINDOW[i] or $O_{out}$;
    $score(o) -= 1$;
**end**
Delete $o_{out}$ from window;
**for** $i \leftarrow 0$ **to** $|WINDOW|$ **do**
    $o \leftarrow$ get which is more dominant WINDOW[i] or $O_{in}$;
    $score(o) += 1$;
**end**
**if** *score*($o_{in}$) $>= score_k$ **then**
    insert $o_{in}$ to TOPK;
**end**
insert $o_{in}$ to $WINDOW$;

**Gambar 3.3:** *Update* Data *Naive Algorithm*

### 3.3.2 Algoritme *Event Based*

Dalam bab ini akan dijelaskan tetang algoritme *event based* yang dapat digunakan pada incomplete data. Algoritme *event based* memodelkan objek-objek pada *sliding window* yang tidak masuk kedalam $TOPK$ menjadi sebuat *event* sehingga objek tersebut akan dikomputasi disaat ia memungkinkan untuk menjadi $TOPK$. Ada beberapa teknik yang digunakan dalam algoritme *event based*. Pada subbab-subbab berikutnya akan dijelaskan tetang teknik-teknik yang digunakan dalam algoritme *event based*:

#### 3.3.2.1 *Event Scheduling*

*Event Scheduling* merupakan sebuah algoritme yang bertugas untuk melakukan *generate event* jika data tidak masuk pada $TOPK$. Dalam melakukan *generate event* pada objek sebuah akan dilakukan perhitungan *safe interval* dari objek tersebut.

Diberikan waktu terminimal dari seruluh objek yang berada pada $TOPK$ $exp1$ dan waktu sekarang $now$, suatu objek tidak akan bisa menjadi $TOPK$ dengan waktu yang dapat dihitung dengan persamaan 3.1[2].

$$SI(o_i) = min\{\lceil(score_k - score(o_i))/2\rceil, exp_1 - now\} \quad (3.1)$$

Persamaan 3.1 dapat dibuktikan dengan bukti: Terdapat dua kemungkinan suatu objek $o_i$ dapat masuk dalam $TOPK$ pada setiap kali *update*: 1) objek yang berada pada $TOPK$ *expire* atau 2) jika skor dominasi pada objek $o_i$ menjadi lebih besar dari pada skor terkecil yang berada pada $TOPK$. Pada kemungkinan yang pertama dapat terpenuhi dengan bagian kedua pada persamaan 3.1. Sedangkan kemungkinan yang ke dua, akan diasumsikan dengan kemungkinan terburuk dimana pada setiap update $o_i$ akan pertambah skor 1 point sedangkan objek dengan skor terkecil pada $TOPK$ ($score_k$) berkurang 1 point. Sehingga waktu pada kemungkinan kedua dapat rumuskan dengan $\lceil(score_k - score(o_i))/2\rceil$ seperti bagian pertama pada persamaan 3.1.
Sebuah event memiliki asosiasi dengan sebuah objek $o \notin TOPK$ dan berisi data sebagai berikut:

- *Event proccessing time* merupakan waktu dimana suatu event akan diproses kembali. *event processing time* disimbolkan dengan $e_i.ept$. Nilai*Event proccessing time* didapat dari nilai *safe interval* dari objek itu sendiri. $e_i.ept = SI(o_i)$.

- *Event generating time* merupakan waktu disaat sebuah *event* di-*generate*. *Event generating time* disimbolkan dengan $e_i.egt$.

- *Shadow score* merupkan perkiraan skor sementara suatu objek yang disimpan dalam sebuah *event*. Skor ini bukan merupakan skor yang sebenarkan dari objek tersebut.

Setelah meng-*generate* suatu event. Event tersebut akan dimasukkan ke dalam *priority queue*. Uturutan *priority* dalam *priority queue* berdasarkan *event proccessing time* $e_i.ept$. Psudocode dari *Event Scheduling* dapat dilihat pada **Gambar 3.4**.

**Algorithm 3:** Schedule Event

**Input:** $j, score, now$
**Output:** None

```
1 exp1 ← minimum expiration time of points in TOPK;
2 ept ← min(⌈(score_k − score)/2⌉ + now),exp1);
3 if ept ≥ now then
4 |   e_j.ept ← ept;
5 |   e_j.egt ← now;
6 |   e_j.shadowScore ← score;
7 |   EventQueue.Insert(e_j);
8 end
```

**Gambar 3.4:** *Schedule Event* pada Algoritme *Event Based*[2]

#### 3.3.2.2 *Recompute* Skor pada Objek yang Pernah Dikomputasi

Melakukan *recompute* pada objek $o_i$ akan sering sekali dilakukan pada algoritme *event based*. Jika pada setiap *recompute* dilakukan perhitungan kembali dengan mengiterasi pada semua objek pada *sliding window* akan membutuhkan

waktu sangat lama. Maka dari itu diajukan algoritme *compute the rest* yang dapat dilihat pada **Gambar 3.5**.

**Algorithm 4:** *Compute The Rest*

**Input:** $o_i$
**Output:** $o_i$ dengan score yang sudah diupdate
$start = |DELETEITEM| - (now - o_i.sg)$;
**if** $start < 0$ **then**
    $start = 0$;
**end**
**for** $j \leftarrow start$ **to** $|DELETEITEM|$ **do**
    $O \leftarrow$ get which is more dominant DELETEITEM[j] or $o_i$;
    $O' \leftarrow$ get which is less dominant DELETEITEM[j] or $o_i$;
    $score(o) -= 1$;
    $o'.parent -= 1$;
**end**
$o_i.sg = now$ **for** $j \leftarrow 0$**to** $|DELETEITEM|$ **do**
    **if** $DELETEITEM[j].parent > 0$ *OR* $score(DELETEITEM[j] > 0$ **then**
        break;
    **end**
    **else**
        delete $DELETEITEM[j]$;
    **end**
**end**
**if** $o_i.eg == 0$ **then**
    $start = 0$;
**end**
**else**
    $start = |WINDOW| - (now - o_i.eg)$;
    **for** $j \leftarrow start$ **to** $|WINDOW|$ **do**
        $o \leftarrow$ get which is more dominant WINDOW[j] or $o_i$;
        $o' \leftarrow$ get which is less dominant WINDOW[j] or $o_i$;
        $score(o) += 1$;
        $o'.parent += 1$;
    **end**
**end**

**Gambar 3.5:** Algoritme *Compute The Rest*

Ide kunci dari algoritme pada **Gambar 3.5** adalah tidak menghitung data yang pernah dihitung sebelumnya dengan cara mencatat waktu menghitung depan dan waktu menghitung belakang pada sliding window. Algoritme pada Gambar 3.5 dapat dibagi menjadi dua bagian yaitu perhitungan pada *sliding window* depan yaitu data yang keluar dan perhitungan pada *sliding window* belakang yaitu pada data yang masuk.

Perhitungan pada bagian depan *sliding window* dapat dilihat pada **Gambar 3.5** dengan baris 1 sampai 18. Ide dari bagian ini adalah menerapkan *soft delete* dimana objek yang *expire* tidak akan langsung dihapus tapi disimpan dulu pada *array* $DELETEITEM$ dengan model *queue*. $DELETEITEM$ ini akan menjadi referensi objek mana saja yang telah *expire*. Dapat dilihat pada gambar 3.5 baris 5 sampai 10. Objek $o$ akan dicek dengan seluruh objek pada $DELETEITEM$. Jika $o$ mendominasi objek pada $DELETEITEM$ $o'$ maka skor $o$ berkurang 1 poin dan *parent* dari $o'$ akan berkurang 1 dan sebaliknya. Untuk menghidari perhitungan berulang pada $DELETEITEM$ maka *index start* ditentukan dengan $start = |DELETEITEM| - (now - o_i.sg)$ jika $start$ kurang dari 0 maka $start = 0$ sehingga tidak akan ada lagi perhitungan yang berulang. Pada akhir perhitunga DELETEITEM akan di bersihkan dari objek yang sudah tidak punya parent dan skor lagi.

Perhitungan pada bagian belakan lebih sederhana daripada perhitungan depan. Perhitundan pada bagian belakang dapat dilihat pada baris 19 sampai 30. Tujuan utama dari perhitungan ini adalah skor dari suatu objek $o$ tampa perlu melakukan perhitungan dengan objek yang sudah pernah di hitung sebelumnya. Hal ini dilakukan dengan cara menentukan ini *index start*. *Index start* ditentukan dengan $start = |WINDOW| - (now - o_i.eg)$ jika $start$ kurang dari 0 maka $start = 0$ sehingga tidak akan ada lagi perhitungan yang berulang.

#### 3.3.2.3 Skor *Upper Bound* untuk Objek yang Baru Masuk

Dalam melakukan proses pada *event based* suatu objek memerlukan skor atau paling tidak *upper bound* skor. Dalam kasus *streaming* objek baru akan terus menerus datang tanpa adanya informasi tentang skor objek yang baru datang. Untuk menghindari perhitungan skor dengan cara membandingkan satu persatu dengan semua objek pada *sliding window* dilakukan pencarian *uppper bound* skor untuk membuat *event* pada objek yang baru masuk $o_{in}$. Pencarian *upper bound* skor dilakukan dengan cara mencari *reference object*. Skor *reference object* akan menjadi $shadowScore$ di *event* dari objek yang baru masuk $o_{in}$. Terdapat beberapa kriteria dalam pencarian *reference object*. Kriteria tersebut adalah sebagai berikut:

1. Objek reference tidak termasuk dalam $TOPK$.
   Dengan mencari objek yang yang tidak masuk dalam $TOPK$ maka objek tersebut dipastikan memiliki event yang akan diambil skornya untuk membentu event pada objek yang baru masuk.
2. Dimensi yang hilang pada objek reference $o_r$ dan objek masuk $o_{in}$ sama.
   Pada kasus ini data yang digunakan adalah data *incomplete*. Pada *complete* data dapat dipastikan jika $o_1$ mendominasi $o_2$ dan $o_2$ mendominasi $o_3$ maka dapat disimpulkan bahwa $o_1$ mendominasi $o_3$. Hal ini berbeda pada *incomplete* data jika $o_1$ mendominasi $o_2$ dan $o_2$ mendominasi $o_3$ maka $o_1$ belum tentu mendominasi $o_3$ dikarenakan *incomplete* data dapat memiliki relasi dominasi *non-transitif*. Dengan menjadikan mencari data yang sama missing dimensinya maka hubungan transitif antar objek dapat diperoleh kembali.
3. Objek reference $o_r$ mendominasi $o_{in}$. Dengan mencari $o_r$ refence dengan sama dimensi missing dengan $o_{in}$ dan mendominasi $o_{in}$ dapat dipastikan $o_r$ memiliki score yang

lebih dari $o_{in}$.

**Algorithm 5:** *Find Reference Object*

```
   Input: o_i
   Output: o_ref
1  dataLengt ← |WINDOW|;
2  for j ← 0 to dataLenght do
3      o ← get which is more dominant WINDOW[j] or o_i;
4      o' ← get which is less dominant WINDOW[j] or o_i;
5      if o_i ≠ o then
6          score(o)+ = 1;
7          o'.parent+ = 1;
8      end
9      else
10         if WINDOW[J] ∉ TOPK then
11             if WINDOW[J] and o_i has same missing dimension
                 then
12                 return WINDOW[j]
13             end
14         end
15     end
16 end
```

**Gambar 3.6:** *Find Reference Object*

Pada **Gambar 3.6** merupakan desain algoritme untuk menghitung skor awal pada saat objek baru $o_{in}$ masuk dengan cara membandingkan satu persatu dengan objek yang berada dalam *sliding window*. Pada saat yang bersamaan juga algoritme ini mencarikan *reference object* $o_r$ untuk $o_{in}$. Jika *reference object* sudah sudah didapat maka algoritme akan berhenti melakukan perhitungan dan mengembalikan *reference object*. Sebaliknya jika algoritme tidak menemukan *reference object* algoritme ini akan menghitung terus sampai skor asli dari $o_{in}$ ditemukan.

#### 3.3.2.4 Pengolahan Data Awal Masuk pada Algoritme *Event Based*

Pada bagian pengolahan data awal masuk pada algoritme *event based* sama seperti algoritme *naive*. Dapat dilihat pada **Gambar 3.7** perbedaan terletak pada baris 8 dan 13. Dimana dibaris ke 8 objek yang terdominasi mencatat jumlah yang mendominasinya. Serta pada baris 13 adalah *pseudocode* untuk melakukan *Schedule* pada data yang yang tidak masuk pada $TOPK$.

**Algorithm 6:** *Initial* Algoritma *Event Based Algorithm*

```
   Input: K, WINDOW, Dimension
   Output: TOPK
1  Initialize all object score in WINDOW to zero;
2  dataLengt ← |WINDOW|;
3  for i ← 0 to dataLenght do
4      for j ← i i to dataLenght do
5          o ← get which is more dominant WINDOW[i] or
             WINDOW[j];
6          o' ← get which is less dominant WINDOW[i] or
             WINDOW[j];
7          score(o)+ = 1;
8          o'.parent+ = 1;
9      end
10 end
11 SORTEDWINDOW ← sort(WINDOW);
12 TOPK ← getTopK(SORTEDWINDOW);
13 ScheduleEvent objek in window that not in TOPK
```

**Gambar 3.7:** *Initial* Algoritme *Event Based Algorithm*

#### 3.3.2.5 *Update* data keluar dan data masuk pada algoritma *Event based*

Pada Bagian update di algoritme *event based* pada dibagi menjadi tiga bagian. Yang Pertama yaitu update semua skor objek pada $TOPK$ dengan data yang keluar dan yang masuk.

Kedua menghitung skor pada objek yang baru masuk $o_{in}$ dengan cara objek baru yang masuk $o_{in}$ akan dicari reference pointnya $o_r$ terlebih dahulu. Jika dapat *reference object* maka *shadow score* akan di-*generate* dari objek *reference*. setelah itu, objek dengan *shadow score* tersebut akan dicoba dimasukkan ke $TOPK$. Jika objek itu tidak memungkinkan masuk pada $TOPK$ maka sebuah *event* akan di-*generate* dan dimasukkan ke *priority queue*. Sedangkan Jika objek tersebut memungkinkan untuk masuk pada $TOPK$ maka skor asli dari objek tersebut akan dihitung dengan mengunakan computeTheRest seperti pada bab 3.3.2.2. Pada bagian ketiga *maintenance event* dimana event-event dengan *event processing time* sama dengan $now$ akan dikeluarkan dari priority queue. lalu akan dicari nilainya update dengan algoritme compute the rest. jika objek tersebut dapat masuk $TOPK$ maka masukkan objek pada $TOPK$. Sedangkan jika objek tersebut tidak bisa masuk pada $TOPK$ maka $rescheduleevent$ dan masukkan pada $EVENTQUEUE$ kembali. Algoritme update secara keseluruhan dapat dilihat pada **Gambar 3.8**.

**Algorithm 7:** *Update* Algoritme *Event Based*

**Input:** $o_{in}, o_{ont}$
**Output:** TOPK

update Window; update score of points in $TOPK$;
$o_r \leftarrow$ FindReferencePoint($o_{in}$);
**if** $o_r$ **then**
    $e_r \leftarrow$ event of $o_r$;
    $score(o_{in}) \leftarrow e_r.score + now - e_r.egt - 1$;
    **if** $score(o_{in}) \geq score_k$ **then**
        compute the rest $score(o_{in})$;
        **if** $score(o_{in}) \geq score_k$ **then**
            insert $o_{in}$ in $TOPK$;
        **end**
    **end**
**end**
**else**
    **if** $score(o_{in}) \geq score_k$ **then**
        insert $o_{in}$ in $TOPK$;
    **end**
**end**
**if** $o_{in} \notin TOPK$ **then**
    ScheduleEvent($o_{in}, score(o_{in}), now$);
**end**
$e_i \leftarrow EventQueue$.RemoveTop();
**while** $e_i.ept = now$ **do**
    **if** $e_i$ *is not rescheduled* **then**
        compute the rest score($p_i$);
        **if** $score(p_i) \geq score_k$ **then**
            insert $p_i$ in $TOPK$;
        **else**
            ScheduleEvent($i, score(p_i), now$);
        **end**
    **end**
    $e_i \leftarrow EventQueue$.RemoveTop();
**end**

**Gambar 3.8:** *Update* Algoritme *Event Based*

## 3.4 Perancangan *Server* dan Antarmuka *Client*

*Server* yang akan dibangun dalam tugas akhir akan dirancang sehingga dapat menjalankan *Top-K Dominating query* serta pada mengirimkan hasilnya ke *client*. *Server* ini akan mengunakan Flash. Saat pertama kali *server* dihidupkan *server* akan melakukan pemanggilan fungsi initialisasi algoritme yang digunakan dengan data yang sudah ada pada awal struktur data. *server* kemudian *server* akan menerima data dari form yang ada pada *client*.

Atarmuka yang akan dirancang adalah antarmuka pada *client* yang akan digunakan oleh *client* untuk melihat pergerakan data pada hasil *Top-K Dominating query*. Atarmuka ini akan mengunakan Reactjs dalam pembangunannya.

# BAB IV

# IMPLEMENTASI

Bab ini membahas implementasi Aplikasi *Top-K Dominating query* berbasis data *streaming* dengan mengunakan *incomplete* data. Pembahasan dengan memberikan gambaran sistem yang besar terlebih dahulu dan dilanjutkan dengan pembahasan secara rinci untuk setiap komponen yang akan dibagun.

## 4.1 Lingkungan Implementasi

Lingkungan implementasi dalam tugas akhir ini meliputi perangkat lunak *server* dan *client* yang akan digunakandan dalam pembangunan aplikasi *top-k dominating query* berbasis data *streaming* dengan mengunakan *incomplete* data adalah sebagai berikut:

1. Perangkat Lunak *Server*:
    - Droplet memory 64GB, 16 VCPU
    - Sistem Operasi Linux Ubuntu Server 16.04 LTS
    - Python 3.6.5
    - Flask
    - Socketio
2. Perangkat Lunak *Client*:
    - Reactjs
    - Socketio
    - Chrome Web Browser

## 4.2 Implementasi Algoritme

Pada subbab impletmentasi algoritme ini akan dijelaskan tentang implementasi algoritme berdasarkan desain yang telah dijelaskan pada Bab III. Implementasi yang dilakukan akan menggunakan bahasa pemrograman python.

### 4.2.1 Fungsi Umum

Pada subbab ini akan diberikan tentang fungsi - fungsi yang digunakan pada semua metode.

#### 4.2.1.1 Fungsi *Dominate*

Fungsi ini adalah implementasi dari fungsi perbandingkan antara dua objek. Pada fungsi ini terdapat masukan dua objek yaitu objek1 dan objek2. Jika objek1 mendominasi objek2 maka fungsi akan mengeluarkan nilai 1. Sebaliknya jika objek2 mendominasi objek1 maka fungsi akan mengeluarkan nilai -1. Sedangkan jika kedua objek tidak saling mendominasi maka akan mengeluarkan nilai 0.

```
1  def dominate(obj1,obj2,dimensi):
2      dominate = 0
3      for i in range(0,dimensi):
4          if obj1[i] == '-' or obj2[i] == '-':
5              continue
6          if obj1[i] > obj2[i]:
7              if dominate == -1:
8                  dominate = 0
9                  break
10             dominate = 1
11         elif obj1[i] < obj2[i]:
12             if dominate == 1:
13                 dominate = 0
14                 break
15             dominate = -1
16     return dominate
```

**Kode Sumber IV.1:** fungsi *Dominate*

#### 4.2.1.2 Fungsi *Compute from Scratch*

Fungsi ini merupakan yang diimplementasikan untuk memakukan mengolahan perhitungan skor dari suatu objek

dengan cara membandingkan pada semua objek pada *sliding window*. Pada fungsi ini akan mengiterasi semua objek yang ada pada *sliding window*. Lalu membandingkan objek yang dikomputasi dengan objek pada sliding window mengunakan fungsi *dominate*. Jika objek yang dihitung mendominasi objek yang lainnya maka skor dari objek tersebut akan bertambah 1 poin. Sedangkan jika objek tersebut terdominasi maka *parent* dari objek tersebut akan bertambah 1 poin.

```
1  def computeFromScratch(self, item):
2      for j in range(self.data_lenght):
3          dominate_status = dominate(
4              item.data, self.windows[j].data, self.
                   ↪ dimension)
5          if dominate_status == 1:
6              item.score += 1
7          if dominate_status == -1:
8              item.parent += 1
9      return item
```

**Kode Sumber IV.2:** fungsi *Compute From Scratch*

### 4.2.2 Algoritme *Naive*

Pada bab ini akan dijelaskan tentang bagimana algoritme *naive* diimplementasikan.

#### 4.2.2.1 Pengolahan Data Awal Masuk Pada *Naive*

Implementasi *initial* pada algoritme naive dari *pseudocode* **Algoritme 1** pada bab 3.3.1.1. Proses inisiasi dilakukan dengan cara menghitung satu persatu skor objek dengan menggunakan fungsi *Compute from Scratch*. Hal ini dapat dilihat pada baris 10-13 dimana pada pemenapannya menggunakan *multi processing* dengan 12 core untuk mempercepat kinerja. setelah itu semua objek akan *sorting* berdasarkan skor yang didapat yang

diimplementasikan pada baris 16-17. Setelah itu akan dilakukan pencarian $TOPK$ pada baris 20-32.

```
1  def __init__(self, K, windows, dimension):
2      self.K = K
3      self.TopK = []
4      self.windows = windows
5      self.dimension = dimension
6      self.data_lenght = len(windows)
7      self.now = len(windows)
8
9      # Compute from scrach
10     p = Pool(12)
11     hasil = p.map(self.computeFromScratch, list(self.
           ↪ windows))
12     for i in range(self.data_lenght):
13         self.windows[i] = hasil[i]
14
15     # sort window by score
16     self.sorted_window = sorted(
17         self.windows, key=lambda object: object.score,
               ↪  reverse=True)
18
19     # get Top-K from sorted window
20     self.TopK = []
21     localK = 0
22     tao = self.data_lenght
23     for row in self.sorted_window:
24         if localK < self.K:
25             self.TopK.append(row)
26             localK += 1
27             if tao > row.score:
28                 tao = row.score
29         elif tao == row.score and localK == self.K:
30             self.TopK.append(row)
31         else:
32             break
```

**Kode Sumber IV.3:** fungsi *initial Naive*

#### 4.2.2.2 Update Data Keluar dan Data Masuk pada Algoritme Naive

Berikut adalah implementasi *update* objek keluar dan masuk dari *pseudocode* **Algoritme 2** pada bab 3.3.1.2. Proses update dilakukan menjadi dua bagian yaitu pengolahan objek masuk dan objek keluar. Untuk objek keluar diproses pada baris 5-13 dengan cara menghitung melakukan perbandingan menggunakan fungsi *dominate* data yang keluar dengan data pada *window*. Bila objek yang akan keluar didominasi oleh objek pada *window* skor akan berkurang dari objek pada *window* akan berkurang dan *parent* dari objek yang keluar akan berkurang. Perhitungan akan berhenti bila *parent* dari objek yang keluar sudah habis. Sedangkan untuk data masuk perhitungan skor dilakukan pada baris 16-27. perhitungan dilakukan seperti pada fungsi *compute from scratch*. Setelah data keluar dan data masuk diproses, *sorting* akan dilakukan pada *window* pada baris 29-30 dan $TOPK$ akan didapatkan kembali pada baris 36-45.

```
1  def update(self, item):
2    # Update time
3    self.now += 1
4    # take item
5    itemleft = self.windows.popleft()
6    for row in self.windows:
7        if itemleft.parent <= 0:
8            break
9        dominate_status_out = dominate(
10           itemleft.data, row.data, self.dimension)
11       if dominate_status_out == -1:
12           row.score -= 1
13           itemleft.parent -=1
14
15   # compute who dominate item
16   for row in self.windows:
17       dominate_status = dominate(
18           item.data, row.data, self.dimension)
19       if dominate_status == 1:
```

```
20              item.score += 1
21              row.parent += 1
22          elif dominate_status == -1:
23              row.score += 1
24              item.parent += 1
25
26      # insert item into windows
27      self.windows.append(item)
28
29      self.sorted_window = sorted(
30          self.windows, key=lambda object: object.score,
                ↪ reverse=True)
31
32      # get Top-K from sorted window
33      self.TopK = []
34      localK = 0
35      tao = self.data_lenght
36      for row in self.sorted_window:
37          if localK < self.K:
38              self.TopK.append(row)
39              localK += 1
40              if tao > row.score:
41                  tao = row.score
42          elif tao == row.score and localK == self.K:
43              self.TopK.append(row)
44          else:
45              break
```

**Kode Sumber IV.4:** fungsi *update* pada *naive*

### 4.2.3 Algoritme *Event Based*

Pada bab ini akan dijelaskan tentang bagimana algoritme *event based* diimplementasikan.

#### 4.2.3.1 *Schedule Event*

Berikut adalah implementasi fungsi *Schedule Event* dari *pseudocode* **Algoritme 3** pada bab 3.3.2.1. Pada baris 2-6

merupakan implementasi dari pembuatan *event processing time*. Sedangkan pada baris 7-11 merupakan implementasi dari pembuatan event dan menyimpannya pada *priority queue*.

```
1  def scheduleEvent(self, item, score, now):
2      Exp1 = None
3      for row in self.TopK:
4          if Exp1 == None or Exp1 > row.exp:
5              Exp1 = row.exp
6      ept = min((math.ceil((self.TopK[-1].score - score)
           ↪ /2)+now), Exp1)
7      if ept >= now:
8          event = Event(item, score, ept, now)
9          item.event = event
10         heapq.heappush(self.EventQueue, event)
11         return True
12     else:
13         return False
```

**Kode Sumber IV.5:** Fungsi *Schedule Event*

### 4.2.4 *Compute The Rest*

Berikut adalah implementasi fungsi *Compute The Rest* dari *pseudocode* **Algoritme 4** pada bab 3.3.2.2. Fungsi *Compute The Rest* pada dibagi menjadi dua bagian utama yaitu perhitungan pada objek yang telah di-*delete* tetapi belum dikomputasi pada objek yang dikomputasi. Hal ini dilakukan pada baris 2 - 14. Sedangkan perhitungan untuk objek yang baru datang tapi belum dikomputasi oleh objek yang sedang melakukan komputasi skor dilakukan pada baris 23-32. Untuk baris 16-21 merupakan baris pendukung dimana objek yang sudah tidak memiliki kepentingan lagi pada objek yang aktif akan di-*delete*.

```
1   def computeTheRest(self, item):
2       start = len(self.delete_item) - (self.now - item.
            ↪ start_generate)
3       if start < 0:
```

```
4           start = 0
5
6       for i in range(start, len(self.delete_item)):
7           dominate_status = dominate(
8               item.data, self.delete_item[i].data, self.
                    ↪ dimension)
9           if dominate_status == 1:
10              item.score -= 1
11              self.delete_item[i].parent -= 1
12          elif dominate_status == -1:
13              self.delete_item[i].score -= 1
14              item.parent -= 1
15
16      item.start_generate = self.now
17      for row in list(self.delete_item):
18          if row.parent > 0 or row.score > 0:
19              break
20          else:
21              self.delete_item.popleft()
22
23      if item.end_generate == 0:
24          start = 0
25      else:
26          start = self.data_lenght - (self.now - item.
                ↪ end_generate)
27      for j in range(start, self.data_lenght):
28          dominate_status = dominate(
29              item.data, self.windows[j].data, self.
                    ↪ dimension)
30          if dominate_status == 1:
31              item.score += 1
32              self.windows[j].parent += 1
33      item.end_generate = self.now
34
35      return item
```

**Kode Sumber IV.6:** fungsi *Compute The Rest*

### 4.2.5 *Find Reference Object*

Berikut adalah implementasi fungsi *Find Reference Object* dari *pseudocode* **Algoritme 5** pada bab 3.3.2.3. Dalam fungsi ini akan dilakukan pencarian skor dominasi dari objek yang masuk. Sambil melakukan perhitungan skor pada fungsi ini juka akan melakukan pencarian objek referesi yang sebagai *upper bound*. Dalam pencarian skor implementasikan pada baris 4-10 sedangkan dalam pencarian objek referensi diimplementasikan pada baris 12-15.

```
1  def findReferencePoint(self, item):
2      isGetReference = 0
3      countDominate = 0
4      for row in self.windows:
5          countDominate += 1
6          dominate_status = dominate(
7              item.data, row.data, self.dimension)
8          if dominate_status == 1:
9              item.score += 1
10             row.parent += 1
11         elif dominate_status == -1:
12             if row.event:
13                 if isSameDimension(item.data, row.data,
                       ↪  self.dimension):
14                     isGetReference = 1
15                     break
16     item.end_generate = self.now - self.data_lenght +
           ↪ countDominate
17     item.start_generate = self.now
18     return isGetReference, row
```

**Kode Sumber IV.7:** fungsi *Find Reference Objek*

#### 4.2.5.1 Pengolahan Data Awal Masuk pada Event Based

Berikut adalah implementasi *initial event based algorithm* dari *pseudocode* **Algoritme 6** pada bab 3.3.2.4. Dalam

melakukan implemetasinya *initial event based algorithm* mirip seperti *naive based* dengan perbedaan pada baris 17-19 yaitu menginisiasi variabel $start\_generate$ dan $end\_generate$ pada setiap objek. Selain itu perbedaan terletak pada baris 36 dimana pemangilan *generate event* akan dilakukan untuk objek yang tidak masuk pada $TOPK$.

```
1  def __init__(self, K, windows, dimension):
2    self.K = K
3    self.TopK = []
4    self.windows = windows
5    self.dimension = dimension
6    self.data_lenght = len(windows)
7    self.delete_item = deque()
8    self.now = len(windows)
9    self.EventQueue = []
10
11   ## COMPUTE ALL FROM SCRATCH
12   p = Pool(12)
13   hasil = p.map(self.computeFromScratch, list(self.
         ↪ windows))
14   for i in range(self.data_lenght):
15       self.windows[i] = hasil[i]
16
17   for i in range(self.data_lenght):
18       self.windows[i].start_generate = 0
19       self.windows[i].end_generate = self.now
20
21   ## SORT ALL AND TAKE TOP-K
22   sorted_window = sorted(
23       self.windows, key=lambda object: object.score,
             ↪ reverse=True)
24   localK = 0
25   tao = self.data_lenght
26   for row in sorted_window:
27       if localK < self.K:
28           self.TopK.append(row)
29           localK += 1
30           if tao > row.score:
31               tao = row.score
```

```
32          elif tao == row.score and localK == self.K:
33              self.TopK.append(row)
34          else:
35              ## JIKA TIDAK MASUK TOP-K CREATE SCHEDULE
36              self.scheduleEvent(row, row.score, self.now)
```

**Kode Sumber IV.8:** fungsi *initial* pada *event based*

#### 4.2.5.2 Update Data Keluar dan Data Masuk pada Event Based

Berikut adalah implementasi *update event based algorithm* dari *pseudocode* **Algoritme 7** pada bab 3.3.2.5. Pada baris 5-8 merupakan implementasi dari *soft delete*. Sedangkan pada baris 10-27 merupakan implementasi untuk update $TOPK$. Selanjutnya pada baris 30 - 43 merupakan implementasi dari insert objek. Dan terakhir baris 46-55 merupakan implementasi dari *maintenance event*.

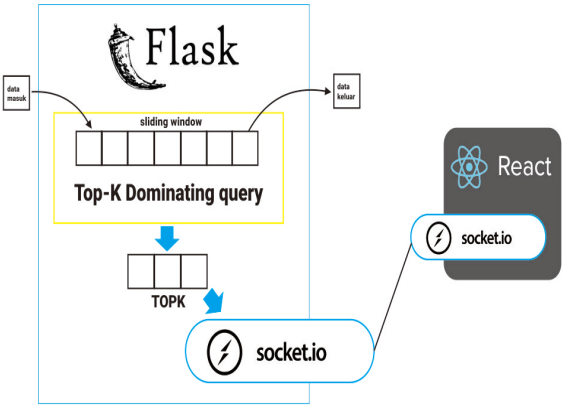
```
1  def update(self, item):
2      self.now += 1
3      self.windows.append(item)
4
5      # DELETE ITEM
6      itemleft = self.windows.popleft()
7      if itemleft.parent > 0:
8          self.delete_item.append(itemleft)
9
10     # UPDATE TOP-K DOMINATING SCORE
11     for i, row in enumerate(self.TopK):
12         if row.exp > self.now:
13             self.TopK[i] = self.computeTheRest(row)
14         else:
15             self.TopK.pop(i)
16
17     self.TopK = sorted(
18         self.TopK, key=lambda object: object.score,
               ↪ reverse=True)
19     # CHECK AFTER UPDATE IT STILL HAS LENGHT K
```

```
20      if len(self.TopK) > self.K:
21          topKextention = self.TopK[self.K:]
22          topKextention.reverse()
23          for row in topKextention:
24              if row.score < self.TopK[self.K-1].score:
25                  self.scheduleEvent(
26                      self.TopK[-1], self.TopK[-1].score,
                            ↪  self.now)
27                  self.TopK.pop()
28
29      ## COMPUTE FROM SCRATCH
30      isGetReference, point = self.findReferencePoint(
            ↪ item)
31      ## INSERT TOPK JIKA TIDAK SCHEDULE ULANG
32      if isGetReference:
33          event = point.event
34          score = event.shadowScore + self.now - event.
                ↪ egt - 1
35          if score >= self.TopK[-1].score:
36              self.computeTheRest(item)
37              if not self.InsertTopKD(item):
38                  self.scheduleEvent(item, item.score,
                        ↪ self.now)
39          else:
40              self.scheduleEvent(item, score, self.now)
41      else:
42          if not self.InsertTopKD(item):
43              self.scheduleEvent(item, item.score, self.
                    ↪ now)
44
45      ## SEE IS THERE ANY EVENT NOW
46      while self.EventQueue:
47          if self.EventQueue[0].ept == self.now:
48              event = heapq.heappop(self.EventQueue)
49              if event.item.exp > self.now:
50                  event.item = self.computeTheRest(event.
                        ↪ item)
51                  if not self.InsertTopKD(event.item):
52                      self.scheduleEvent(
53                          event.item, event.item.score,
                                ↪ self.now)
```

```
54          else:
55              break
```

**Kode Sumber IV.9:** fungsi *update* pada *event based*

## 4.3 Implementasi *Server* dan Antarmuka

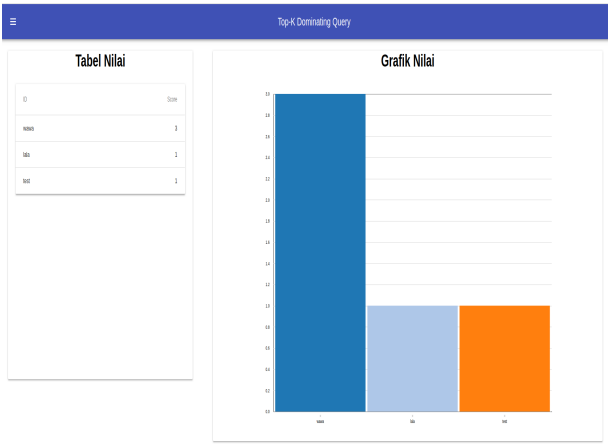



**Gambar 4.1:** Keseluruhan Sistem yang Dibangun

Pada bab ini akan dijelaskan implementasi algoritme yang diterapkan pada *server* sampai tampil ke *client*. Pada **Gambar 4.1** dapat dilihat bahwa query akan dijalankan oleh flask *server*. Sering dengan terjadinya *update* dimana ada data yang keluar dan ada data yang masuk. Sistem akan melakukan query dan mendapatkan $TOPK$. Hasil dari $TOPK$ akan dikirim kepada *client* dengan mengunakan socket.io sehingga *client* tidak perlu mekalukan refresh untuk mengupdate hasil dari $TOPK$.

Pada **Gambar 4.2** merupakan hasil implementasi antarmuka pada *client* yang akan dibuat. Tampilan terdiri dari dua bagian.

Pertama bagian tabel rangking dimana disana akan diperlihatkan hasil rangking dari objek yang berada pada *sliding window* dan yang kedua adalah grafik batang untuk melihat hasil rangking dengan lebih baik.




**Gambar 4.2:** Tampilan pada *Client*

# BAB V

# UJI COBA DAN EVALUASI

Dalam bab ini dibahas mengenai hasil uji coba aplikasi yang telah dirancang dan diimplementasikan. Uji coba dilakukan untuk mengetahui kinerja aplikasi dengan lingkungan uji coba yang telah ditentukan.

## 5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan dalam menguji implementasi dua metode yaitu *dictionary based* dan *event based* dalam mengolah *top-k dominating query* berbasis data *streaming* dengan menggunakan *incomplete* data. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat Keras:
    - Prosesor: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
    - Memori: 64 GB
    - Arsitektur Prosesor: 64-bit
2. Perangkat Lunak:
    - Sistem Operasi: Ubuntu 16.04.4 LTS
    - Perangkat Pengembang: Python 3.6.5

## 5.2 Data Uji Coba

Terdapat tiga jenis data yang akan digunakan dalam pengujian Tugas Akhir ini. Tiga jenis data tersebut terdiri dari data *independent* (IND), data *anti corelated* (ANT), dan data *forest covertype* (FC).

### 5.2.1 Data Independen

Data independen (IND) adalah himpunan data sintetis yang dibuat pada penelitian ini. Data independen memiliki persebaran

nilai atribut yang acak tidak saling terpengaruh antara atribut satu dengan lainnya ataupun antar data lainnya. Penggunaan data ini bertujuan untuk menguji performa algoritme jika berhadapan dengan data yang nilai atributnya tidak memiliki keterkaitan dengan apapun. Rentang nilai yang digunakan untuk setiap atribut adalah 1 – 100.

### 5.2.2 Data *Anti Corelated*

Data *anti correlated* (ANT) adalah himpunan data sintetis yang dibuat pada penelitian ini. Data *anti correlated* memiliki persebaran nilai atribut yang saling bertolak belakang antara satu antribut dengan atribut lainnya, yang artinya sebuah data memiliki nilai yang sangat baik pada salah satu atributnya namun sangat buruk pada atribut lainnya. Penggunaan data ini bertujuan untuk menguji performa algoritme jika berhadapan dengan data yang nilai atributnya saling bertolak belakang dan memiliki relasi dominasi antar data paling rendah dibandingkan dengan jenis data lainnya. Hal ini dikarenakan seluruh data akan menjadi titik *skyline* pada himpunan data ini. Rentang nilai yang digunakan untuk setiap atribut adalah 1 – 100.

### 5.2.3 Data *Forest Covertype*

Data *forest covertype* adalah sebuah data yang didapat dari situs UCI machine learning. Penggunaan data ini bertujuan untuk menguji performa algoritme pada data dengan persebaran dan rentang nilai atribut sebenarnya yang bersifat saling berkaitan. Sebelum data *forest covertype* digunakan, akan dilakukan *preprocessing* yaitu normalisasi dengan menggunakan algoritme *min-max* dalam rentang 1 - 100 lalu untuk mengambil dimensi dan baris yang dibutuhkan dengan cara memotong data

secara langsung.

## 5.3 Skenario Uji Coba

Uji coba dilakukan untuk mengetahui *performance* dari algoritme - algoritme yang telah didesain dan diimplementasikan untuk mengetahui algoritme mana yang lebih cocok dalam permasalahan ini. Sebelum uji coba data independent, *anti corelated*, dan *forest covertype* akan dihilangkan 20 persen datanya secara acak untuk membuat data menjadi *incomplete*.

Dalam skenario uji coba, *performance* antara algoritme - algoritme yang diusulkan akan dibandingkan. Pada tugas akhir ini *performance* merupakan rata-rata waktu eksekusi satu query dan jumlah penggunaan memory. Skenario uji coba diberlakukan ke semua jenis data (independen, dan *anti correlated*,*forest covertype*). Berikut adalah beberapa skenario pengujian:

### 5.3.1 Skenario Uji Coba *Performance* Terhadap Perubahan Dimensi Data

Skenario ini bertujuan untuk mengetahui hubungan antara dimensi ($d$) dengan rata-rata waktu eksekusi satu *query* dan jumlah penggunaan memori. Detail skenario dapat dilihat pada **Table 5.1**.

**Tabel 5.1:** Skenario Perubahan Dimensi

| Nomor | Dimensi ($d$) | Panjang *Sliding Window* ($n$) | $k$ |
|---|---|---|---|
| A01 | 2 | 30.000 | 50 |
| A02 | 3 | 30.000 | 50 |
| A03 | 5 | 30.000 | 50 |
| A04 | 7 | 30.000 | 50 |
| A05 | 10 | 30.000 | 50 |

### 5.3.2 Skenario Uji Coba *Performance* Terhadap Perubahan Panjang *Sliding Window*

Skenario ini bertujuan untuk mengetahui hubungan antara panjang *sliding window* ($n$) dengan rata-rata waktu eksekusi satu *query* dan jumlah penggunaan memori. Detail skenario dapat dilihat pada **Table 5.2**.

**Tabel 5.2:** Skenario Perubahan Panjang *Sliding Window*

| Nomor | Dimensi ($d$) | Panjang *Sliding Window* ($n$) | $k$ |
|---|---|---|---|
| B01 | 3 | 10.000 | 50 |
| B02 | 3 | 30.000 | 50 |
| B03 | 3 | 50.000 | 50 |
| B04 | 3 | 100.000 | 50 |
| B05 | 3 | 200.000 | 50 |

### 5.3.3 Skenario Uji Coba *Performance* Terhadap Perubahan Nilai $K$

Skenario ini bertujuan untuk mengetahui hubungan antara Nilai $K$ dengan rata-rata waktu eksekusi satu *query* dan jumlah penggunaan memori. Detail skenario dapat dilihat pada **Table 5.3**.

**Tabel 5.3:** Skenario Perubahan Nilai $K$

| Nomor | Dimensi ($d$) | Panjang *Sliding Window* ($n$) | $k$ |
|---|---|---|---|
| C01 | 3 | 30.000 | 25 |
| C02 | 3 | 30.000 | 50 |
| C03 | 3 | 30.000 | 100 |
| C04 | 3 | 30.000 | 200 |
| C05 | 3 | 30.000 | 500 |

## 5.4 Hasil Uji Coba dan Evaluasi

Pada bab ini akan dipaparkan hasil dari uji coba *performance* dari dua algoritme yaitu algoritme *naive* dan algoritme *event based*.

### 5.4.1 Skenario Uji Coba *Performance* Terhadap Perubahan Dimensi Data

Pada subbab ini akan dipaparkan hasil dari pengujian skenario uji coba penggunaan *memory* dan waktu *query* terhadap perubahan dimensi.

#### 5.4.1.1 Independen

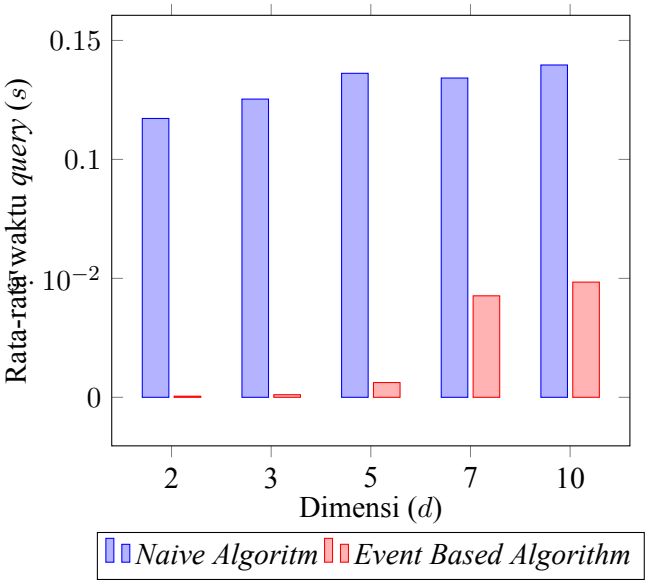
Dalam percobaan dengan data independen didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk menjalankan satu *query* dari algoritme *naive* seperti pada **Tabel 5.4** dan algoritme *event based* seperti pada **Tabel 5.5**.

**Tabel 5.4:** Hasil Percobaan Perubahan Dimensi Algoritme *Naive* Menggunakan Data Independen

| Dimension | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 2 | 52,04 | 185,66 | $1,17 \times 10^{-1}$ |
| 3 | 62,28 | 199,10 | $1,25 \times 10^{-1}$ |
| 5 | 67,43 | 218,99 | $1,36 \times 10^{-1}$ |
| 7 | 70,22 | 227,00 | $1,34 \times 10^{-1}$ |
| 10 | 88,48 | 239,42 | $1,39 \times 10^{-1}$ |

**Tabel 5.5:** Hasil Percobaan Perubahan Dimensi Algoritme *Event Based* Menggunakan Data Independen

| Dimension | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 2 | 54,78 | 184,73 | $4,77 \times 10^{-4}$ |
| 3 | 62,60 | 197,40 | $1,09 \times 10^{-3}$ |
| 5 | 67,62 | 226,42 | $6,20 \times 10^{-3}$ |
| 7 | 70,04 | 228,72 | $4,26 \times 10^{-2}$ |
| 10 | 82,58 | 234,67 | $4,84 \times 10^{-2}$ |




**Gambar 5.1:** Perubahan Dimensi Terhadap Perubahan Rata-Rata Waktu Satu *Query* Menggunakan Data Independen

Dapat dilihat pada **Tabel 5.4** dan **Tabel 5.5** bahwa *memory* dan *preparing time* dari kedua algoritme *naive* dan algoritme *event based* cenderung tidak jauh berbeda. Dari **Tabel 5.4** dan

**Tabel 5.5** dapat ditarik informasi juga semakin tinggi dimensi pada data berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive* dan algoritme *event based*. Dari **Tabel 5.4** dan **Tabel 5.5** dengan mengambil data *query time* pada kolom ke empat dibentuklah grafik seperti pada **Gambar 5.1** untuk memudahkan dalam melihat perubahan yang terjadi.

Dari **Gambar 5.1** dapat diketahui bahwa perbedaan rata-rata waktu menjalankan satu *query* antara algoritme *naive* dan algoritme *event based* cukup jauh. Pada setiap *case* uji coba algoritme *event based* lebih unggul dari pada algoritme *naive*. Dapat diamati pula pada data independen perubahan rata-rata waktu *query* pada algoritme *naive* fluktuatif tetapi cenderung naik pada setiap kenaikan dimensi. Sedangkan pada rata-rata waktu *query* pada algoritme *event based* cenderung naik dengan kenaikan yang paling signifikan terjadi dari $5$ dimensi ke $7$ dimensi yaitu $6,20 \times 10^{-3}$ *second* ke $4,26 \times 10^{-2}$ *second*.

#### 5.4.1.2 *Anti Corelated*

Dalam percobaan dengan data *anti corelated* didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk menjalankan satu *query* dari algoritme *naive* seperti pada **Table 5.6** dan algoritme *event based* seperti pada **Table 5.7**.

**Tabel 5.6:** Hasil Percobaan Perubahan Dimensi Algoritme *Naive* Menggunakan Data *Anti Corelated*

| Dimension | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 2 | 52,17 | 183,18 | $1,12 \times 10^{-1}$ |
| 3 | 62,39 | 203,53 | $1,25 \times 10^{-1}$ |
| 5 | 67,71 | 222,78 | $1,35 \times 10^{-1}$ |
| 7 | 70,32 | 234,14 | $1,42 \times 10^{-1}$ |

**Tabel 5.6:** Hasil Percobaan Perubahan Dimensi Algoritme *Naive* Menggunakan Data *Anti Corelated*

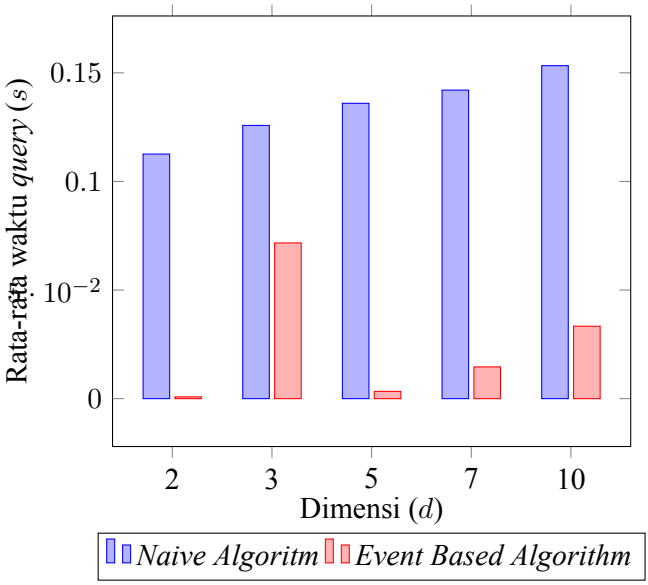
| Dimension | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 10 | 82,25 | 266,02 | $1,53 \times 10^{-1}$ |

**Tabel 5.7:** Hasil Percobaan Perubahan Dimensi Algoritme *Event Based* Menggunakan Data *Anti Corelated*

| Dimension | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 2 | 54,85 | 181,07 | $8,31 \times 10^{-4}$ |
| 3 | 62,65 | 193,89 | $7,16 \times 10^{-2}$ |
| 5 | 67,85 | 222,38 | $3,34 \times 10^{-3}$ |
| 7 | 70,23 | 244,53 | $1,46 \times 10^{-2}$ |
| 10 | 82,74 | 262,29 | $3,33 \times 10^{-2}$ |

Dapat dilihat pada **Tabel 5.6** dan **Tabel 5.7** bahwa *preparing time* dari kedua algoritme *naive* dan algoritme *event based* tidak jauh berbeda. Sedangkan pada penggunaan *memory* algoritme *event based* cenderung lebih tinggi sedikit dari pada algoritme *naive*. Dari **Tabel 5.6** dan **Tabel 5.7** dapat ditarik informasi juga semakin tinggi dimensi pada data berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive* dan algoritme *event based*.

Pada **Tabel 5.6** dan **Tabel 5.7** dengan mengambil data *query time* pada kolom ke empat dibentuklah grafik seperti pada **Gambar 5.2** untuk memudahkan dalam melihat perubahan yang terjadi.




**Gambar 5.2:** Perubahan Dimensi Terhadap Perubahan Rata-Rata Waktu Satu *Query* Menggunakan Data *Anti Corelated*

Dari **Gambar 5.2** dapat diketahui bahwa perbedaan rata-rata waktu *query* antara algoritme *naive* dan algoritme *event based* cukup jauh. Pada setiap *case* uji coba algoritme *event based* lebih unggul dari pada algoritme *naive*. Dapat diamati pula pada data *anti corelated* perubahan rata-rata waktu query pada algoritme *naive* cenderung naik pada setiap kenaikan dimensi. Sedangkan pada rata-rata waktu *query* pada algoritme *event based* cenderung fluktiatif dengan rata-rata waktu *query* tertinggi terdapat saat data dengan 3 dimensi yaitu $7,16 \times 10^{-2}$ *second*. Hal ini terjadi disebabkan pengolahan *event based* yang melakukan *filter* untuk melakukan *event rescheduling* tidak berjalan pada kebanyakan objek sehingga pada kasus uji ini banyak objek yang diolah secara keseluruhan.

#### 5.4.1.3 *Forest Covertype*

Dalam percobaan dengan data *forest covertype* didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk menjalankan satu *query* dari algoritme naive seperti pada **Table 5.8** dan algoritme *event based* seperti pada **Table 5.9**.

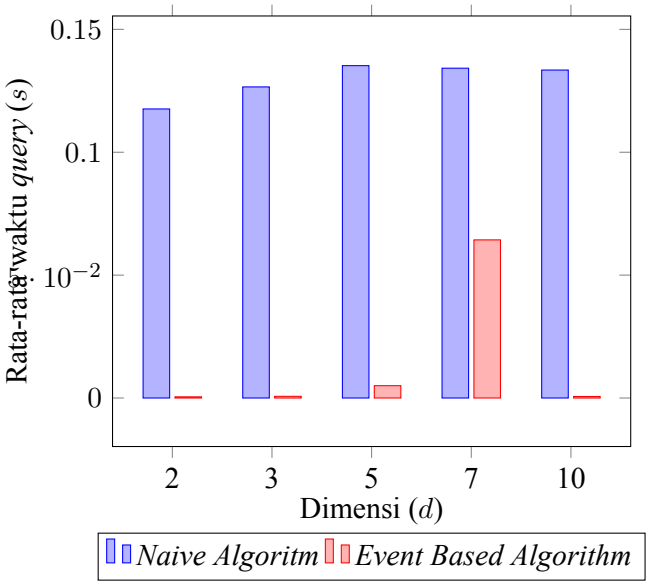
**Tabel 5.8:** Hasil Percobaan Perubahan Dimensi Algoritme *Naive* Menggunakan Data *Forest Covertype*

| Dimension | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 2 | 55,15 | 184,97 | $1,17 \times 10^{-1}$ |
| 3 | 62,37 | 197,10 | $1,26 \times 10^{-1}$ |
| 5 | 67,15 | 230,67 | $1,35 \times 10^{-1}$ |
| 7 | 70 | 230,43 | $1,34 \times 10^{-1}$ |
| 10 | 90,10 | 242,47 | $1,33 \times 10^{-1}$ |

Dapat dilihat pada **Tabel 5.8** semakin tinggi dimensi pada data berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive*.

**Tabel 5.9:** Hasil Percobaan Perubahan Dimensi Algoritme *Event Based* Menggunakan Data *Forest Covertype*

| Dimension | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 2 | 55,60 | 189,73 | $4,72 \times 10^{-4}$ |
| 3 | 62,44 | 202,32 | $6,84 \times 10^{-4}$ |
| 5 | 67,48 | 218,48 | $5,01 \times 10^{-3}$ |
| 7 | 75,91 | 235,60 | $6,43 \times 10^{-2}$ |
| 10 | 90,08 | 238,77 | $6,13 \times 10^{-4}$ |




**Gambar 5.3:** Perubahan Dimensi Terhadap Perubahan Rata-Rata Waktu Satu *Query* Menggunakan Data *Forest Covertype*

Dari **Tabel 5.9** dapat ditarik informasi juga semakin tinggi dimensi pada data berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *event based*.

Dapat dilihat pada **Tabel 5.8** dan **Tabel 5.9** bahwa *preparing time* dari kedua algoritme *naive* dan algoritme *event based* tidak jauh berbeda. Sedangkan pada penggunaan *memory* algoritme *event based* cenderung lebih tinggi sedikit dari pada algoritme *naive*.

Pada **Tabel 5.8** dan **Tabel 5.9** dengan mengambil *query time* pada kolom ke empat dibentuklah grafik pada **Gambar 5.3** untuk memudahkan dalam melihat perubahan yang terjadi.

Dari **Gambar 5.3** dapat diketahui bahwa perbedaan rata-rata waktu *query* antara algoritme *naive* dan algoritme *event based* cukup jauh. Pada setiap *case* uji coba algoritme *event based*

lebih unggul dari pada algoritme *naive*. Dapat diamati pula pada data *forest covertype* perubahan rata-rata waktu *query* pada algoritme *naive* dan algoritme *event based* sama-sama cenderung fluktuatif pada setiap kenaikan dimensi. Pada algoritme *naive* rata-rata waktu *query* tertinggi terdapat saat pengujian dengan data dengan 5 dimensi yaitu $1,35 \times 10^{-1}$ *second*. Sedangkan pada algoritme *event based* rata-rata waktu *query* tertinggi terdapat saat pengujian dengan data 7 dimensi yaitu $6,43 \times 10^{-2}$ *second*. Hal ini terjadi disebabkan pengolahan *event based* yang melakukan *filter* untuk melakukan event rescheduling tidak berjalan pada kebanyakan objek sehingga pada kasus uji ini banyak objek yang diolah secara keseluruhan.

### 5.4.2 Skenario Uji Coba *Performance* Terhadap Perubahan Panjang *Sliding Window*

Pada subbab ini akan dipaparkan hasil dari pengujian skenario uji coba penggunaan memory dan waktu query terhadap perubahan panjang *sliding window*.

#### 5.4.2.1 Independen

Dalam percobaan dengan data independen didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk menjalankan satu *query* dari algoritme *naive* seperti pada **Tabel 5.10** dan algoritme *event based* seperti pada **Tabel 5.11**.

**Tabel 5.10:** Hasil Percobaan Perubahan Panjang *Sliding Window* Algoritme *Naive* Menggunakan Data Independen

| Panjang *Sliding Window* | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 10.000 | 29,34 | 21,26 | $4,08 \times 10^{-2}$ |
| 30.000 | 62,28 | 199,10 | $1,25 \times 10^{-1}$ |
| 50.000 | 86,56 | 635,23 | $2,12 \times 10^{-1}$ |

**Tabel 5.10:** Hasil Percobaan Perubahan Panjang *Sliding Window* Algoritme *Naive* Menggunakan Data Independen

| Panjang *Sliding Window* | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 100.000 | 155,64 | 3640,93 | $4,43 \times 10^{-1}$ |
| 200.000 | 284,18 | 23694,78 | $9,17 \times 10^{-1}$ |

Dapat dilihat pada **Tabel 5.10** semakin tinggi dimensi pada data berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive*.

**Tabel 5.11:** Hasil Percobaan Perubahan Panjang *Sliding Window* Algoritme *Event Based* Menggunakan Data Independen

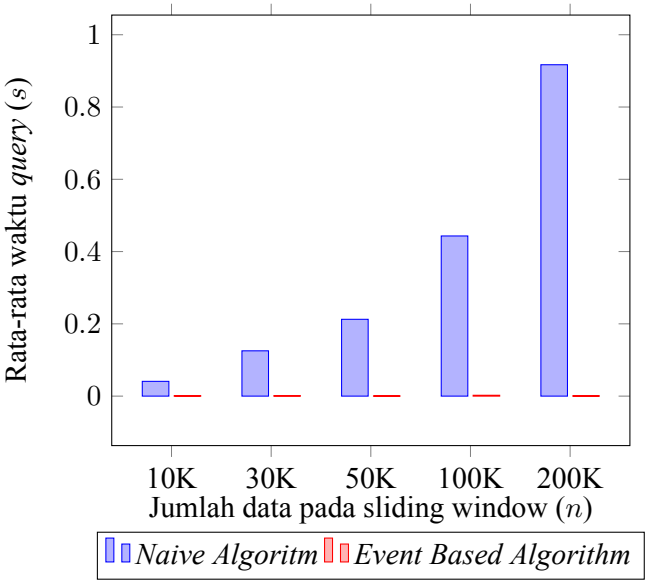
| Panjang *Sliding Window* | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 10.000 | 28,51 | 22,04 | $9,15 \times 10^{-4}$ |
| 30.000 | 62,60 | 197,40 | $1,09 \times 10^{-3}$ |
| 50.000 | 87,03 | 635,23 | $5,62 \times 10^{-4}$ |
| 100.000 | 159,65 | 3641,52 | $2,37 \times 10^{-3}$ |
| 200.000 | 276,07 | 22838,15 | $7,56 \times 10^{-4}$ |

Dapat dilihat pada **Tabel 5.11** semakin tinggi panjang *sliding windows* pada data berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *event based*.

Dari **Tabel 5.10** dan **Tabel 5.11** dapat ditarik informasi juga bahwa penggunaan *memory* dan *preparing time* dari kedua algoritme *naive* dan algoritme *event based* tidak jauh berbeda.

Pada **Tabel 5.10** dan **Tabel 5.11** dengan mengambil *query time* pada kolom ke empat dibentuklah grafik pada **Gambar 5.4** untuk memudahkan dalam melihat perubahan yang terjadi.

Dari **Gambar 5.4** dapat diketahui bahwa perbedaan rata-rata waktu *query* antara algoritme *naive* dan algoritme *event based* cukup jauh. Pada setiap *case* uji coba algoritme *event based* lebih unggul dari pada algoritme *naive*. Dapat diamati pula pada data independen perubahan rata-rata waktu *query* pada algoritme *naive* cenderung naik pada setiap kenaikan jumlah data pada *sliding window*. Sedangkan algoritme *event based* rata-rata waktu *query* cenderung konstan.




**Gambar 5.4:** Perubahan Panjang *Sliding Window* Terhadap Perubahan Rata-Rata Waktu Satu *Query* Menggunakan Data Independen

### 5.4.2.2 *Anti Corelated*

Dalam percobaan dengan data *anti corelated* didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk menjalankan satu *query* dari algoritme *naive* seperti pada **Table**

**5.12** dan algoritme *event based* seperti pada **Table 5.13**.

**Tabel 5.12:** Hasil Percobaan Perubahan Panjang *Sliding Window* Algoritme *Naive* Menggunakan Data *Anti Corelated*

| Panjang *Sliding Window* | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 10.000 | 29,61 | 20,60 | $3,93 \times 10^{-2}$ |
| 30.000 | 62,39 | 203,53 | $1,25 \times 10^{-1}$ |
| 50.000 | 86,68 | 627,19 | $2,09 \times 10^{-1}$ |
| 100.000 | 155,65 | 3620,32 | $4,24 \times 10^{-1}$ |
| 200.000 | 284,74 | 23398,80 | $8,93 \times 10^{-1}$ |

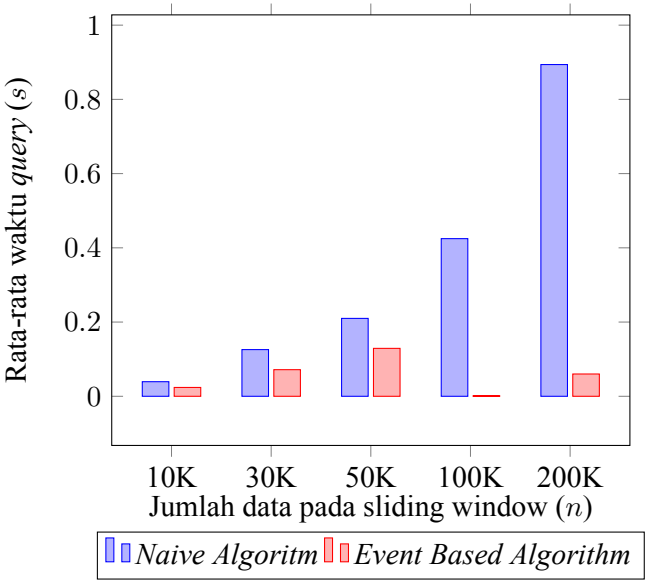
**Tabel 5.13:** Hasil Percobaan Perubahan Panjang *Sliding Window* Algoritme *Event Based* Menggunakan Data *Anti Corelated*

| Panjang *Sliding Window* | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 10.000 | 28,89 | 21,44 | $2,38 \times 10^{-2}$ |
| 30.000 | 62,65 | 193,89 | $7,16 \times 10^{-2}$ |
| 50.000 | 87,14 | 646,91 | $1,29 \times 10^{-1}$ |
| 100.000 | 156,71 | 3589,59 | $1,41 \times 10^{-3}$ |
| 200.000 | 283,16 | 23098,57 | $6,00 \times 10^{-2}$ |

Dapat dilihat pada **Tabel 5.12** dan **Tabel 5.13** bahwa *preparing time* dan penggunaan *memory* dari kedua algoritme *naive* dan algoritme *event based* tidak jauh berbeda. Dari **Tabel 5.12** dan **Tabel 5.13** dapat ditarik informasi juga semakin panjang *sliding window* berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive* dan algoritme *event based*.

Pada **Tabel 5.12** dan **Tabel 5.13** dengan mengambil data *query time* pada kolom ke empat dibentuklah grafik seperti pada

**Gambar 5.5** untuk memudahkan dalam melihat perubahan yang terjadi.




**Gambar 5.5:** Perubahan Panjang *Sliding Window* Terhadap Perubahan Rata-Rata Waktu Satu *Query* Menggunakan Data *Anti Corelated*

Dari **Gambar 5.5** dapat diketahui bahwa perbedaan rata-rata waktu *query* antara algoritme *naive* dan algoritme *event based* cukup jauh. Pada setiap *case* uji coba algoritme *event based* lebih unggul dari pada algoritme *naive*. Dapat diamati pula pada data *anti corelated* perubahan rata-rata waktu query pada algoritme *naive* cenderung naik pada setiap kenaikan dimensi. Sedangkan pada rata-rata waktu *query* pada algoritme *event based* cenderung fluktiatif dengan rata-rata waktu *query* tertinggi terdapat saat panjang *sliding window* 50.000 yaitu $1,29 \times 10^{-1}$ *second*.

#### 5.4.2.3 *Forest Covertype*

Dalam percobaan dengan data *forest covertype* didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk menjalankan satu *query* dari algoritme naive seperti pada **Table 5.14** dan algoritme *event based* seperti pada **Table 5.15**.

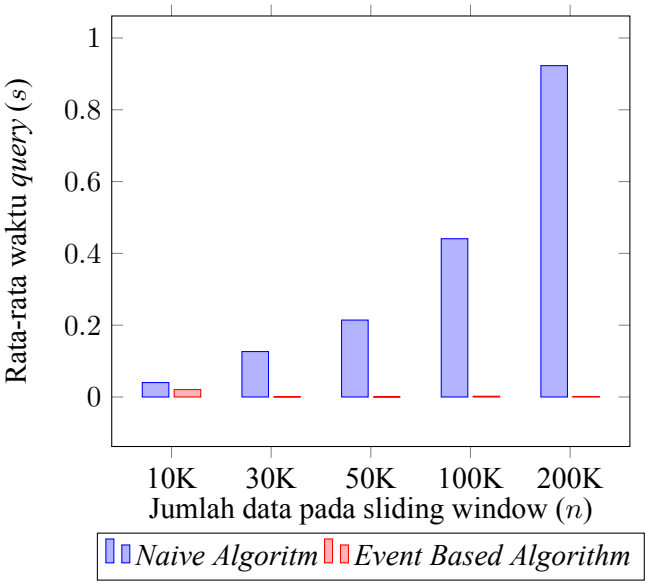
**Tabel 5.14:** Hasil Percobaan Perubahan Panjang *Sliding Window* Algoritme *Naive* Menggunakan Data *Forest Covertype*

| Panjang *Sliding Window* | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 10.000 | 28,97 | 21,87 | $4,01 \times 10^{-2}$ |
| 30.000 | 62,37 | 197,10 | $1,26 \times 10^{-1}$ |
| 50.000 | 86,46 | 630,08 | $2,14 \times 10^{-1}$ |
| 100.000 | 166,28 | 3638,27 | $4,40 \times 10^{-1}$ |
| 200.000 | 281,96 | 24087,30 | $9,22 \times 10^{-1}$ |

Dapat dilihat pada **Tabel 5.14** semakin panjang *sliding window* berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive*.

**Tabel 5.15:** Hasil Percobaan Perubahan Panjang *Sliding Window* Algoritme *Event Based* Menggunakan Data *Forest Covertype*

| Panjang *Sliding Window* | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 10.000 | 28,65 | 21,52 | $2,07 \times 10^{-2}$ |
| 30.000 | 62,44 | 202,32 | $6,84 \times 10^{-4}$ |
| 50.000 | 95,17 | 640,66 | $4,15 \times 10^{-4}$ |
| 100.000 | 158,42 | 3705,73 | $2,07 \times 10^{-3}$ |
| 200.000 | 280,55 | 24194,09 | $1,35 \times 10^{-3}$ |




**Gambar 5.6:** Perubahan Rata-Rata Waktu Satu *Query* Terhadap Perubahan Panjang *Sliding Window* Menggunakan Data *Forest Covertype*

Dari **Tabel 5.15** dapat ditarik informasi juga semakin panjang *sliding window* berpengaruh pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *event based*.

Dapat dilihat pada **Tabel 5.14** dan **Tabel 5.15** bahwa penggunaan *memory* dari kedua algoritme *naive* dan algoritme *event based* tidak jauh berbeda. Sedangkan pada *preparing time* algoritme *event based* cenderung lebih tinggi dari pada algoritme *naive*. Pada **Tabel 5.14** dan **Tabel 5.15** dengan mengambil *query time* pada kolom ke empat dibentuklah grafik pada **Gambar 5.6** untuk memudahkan dalam melihat perubahan yang terjadi.

Dari **Gambar 5.6** dapat diketahui bahwa perbedaan rata-rata waktu *query* antara algoritme *naive* dan algoritme *event based* cukup jauh. Pada setiap *case* uji coba algoritme *event based*

lebih unggul dari pada algoritme *naive*. Dapat diamati pula pada data *forest covertype* perubahan rata-rata waktu *query* pada algoritme *naive* mengalami kenaikan setiap kali panjang *sliding window* dinaikkan sedangkan algoritme *event based* cenderung fluktuatif pada setiap kenaikan dimensi dengan rata-rata waktu *query* tertinggi terdapat saat pengujian dengan panjang *sliding window* 10.000 yaitu $2,07 \times 10^{-2}$ *second*.

### 5.4.3 Skenario Uji Coba *Performance* Terhadap Perubahan Nilai $K$

Pada subbab ini akan dipaparkan hasil dari pengujian skenario uji coba penggunaan memory dan waktu query terhadap perubahan nilai $k$.

#### 5.4.3.1 Independen

Dalam percobaan dengan data independen didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk menjalankan satu *query* dari algoritme *naive* seperti pada **Tabel 5.16** dan algoritme *event based* seperti pada **Tabel 5.17**.

Dapat dilihat pada **Tabel 5.16** semakin tinggi nilai $K$ pada data tidak memiliki pengaruh yang signifikan pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive*.

**Tabel 5.16:** Hasil Percobaan Perubahan Nilai $K$ Algoritme *Naive* Menggunakan Data Independen

| Nilai $K$ | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 25 | 62.58 | 199.18 | $1,25 \times 10^{-1}$ |
| 50 | 62.28 | 199.10 | $1,25 \times 10^{-1}$ |
| 100 | 62.64 | 200.42 | $1,25 \times 10^{-1}$ |

**Tabel 5.16:** Hasil Percobaan Perubahan Nilai $K$ Algoritme *Naive* Menggunakan Data Independen

| Nilai $K$ | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 200 | 62.60 | 202.60 | $1,27 \times 10^{-1}$ |
| 500 | 62.72 | 205.65 | $1,31 \times 10^{-1}$ |

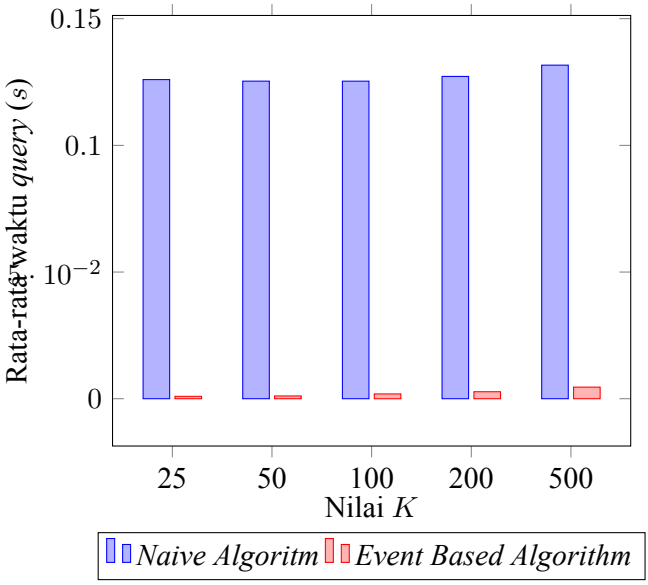
**Tabel 5.17:** Hasil Percobaan Perubahan Nilai $K$ Algoritme *Event Based* Menggunakan Data Independen

| Nilai $K$ | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 25 | 62.59 | 202.22 | $9,40 \times 10^{-4}$ |
| 50 | 62.60 | 197.40 | $1,09 \times 10^{-3}$ |
| 100 | 62.46 | 203.92 | $1,89 \times 10^{-3}$ |
| 200 | 62.63 | 202.01 | $2,76 \times 10^{-3}$ |
| 500 | 62.40 | 203.97 | $4,58 \times 10^{-3}$ |

Dapat dilihat pada **Tabel 5.17** semakin tinggi nilai $K$ pada data tidak memiliki pengaruh yang signifikan pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *event based*.

Dari **Tabel 5.16** dan **Tabel 5.17** dapat ditarik informasi juga bahwa penggunaan *memory* dan *preparing time* dari kedua algoritme *naive* dan algoritme *event based* tidak jauh berbeda.

Pada **Tabel 5.16** dan **Tabel 5.17** dengan mengambil data *query time* pada kolom ke empat dibentuklah grafik pada **Gambar 5.7** untuk memudahkan dalam melihat perubahan yang terjadi.




**Gambar 5.7:** Perubahan Nilai $K$ Terhadap Perubahan Rata-Rata Waktu Satu *Query* Menggunakan Data Independen

Dari **Gambar 5.7** dapat diketahui bahwa perbedaan rata-rata waktu *query* antara algoritme *naive* dan algoritme *event based* cukup jauh. Pada setiap *case* uji coba algoritme *event based* lebih unggul dari pada algoritme *naive*. Dapat diamati pula pada data independen perubahan rata-rata waktu *query* pada algoritme *naive* cenderung konstan pada setiap kenaikan nilai $K$. Sedangkan algoritme *event based* rata-rata waktu *query* cenderung meningkat dengan nilai yang kecil pada setiap kenaikan nilai $K$.

#### 5.4.3.2 Anti Corelated

Dalam percobaan dengan data *anti corelated* didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk

menjalankan satu *query* dari algoritme *naive* seperti pada **Table 5.18** dan algoritme *event based* seperti pada **Table 5.19**.

Dapat dilihat pada **Tabel 5.18** dan **Tabel 5.19** bahwa *preparing time* dan penggunaan *memory* dari kedua algoritme *naive* dan algoritme *event based* tidak jauh berbeda.

**Tabel 5.18:** Hasil Percobaan Perubahan Nilai $K$ Algoritme *Naive* Menggunakan Data *Anti Corelated*

| Nilai $K$ | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 25 | 62.75 | 199.82 | $1,24 \times 10^{-1}$ |
| 50 | 62.39 | 203.53 | $1,25 \times 10^{-1}$ |
| 100 | 62.62 | 201.46 | $1,24 \times 10^{-1}$ |
| 200 | 62.60 | 198.66 | $1,23 \times 10^{-1}$ |
| 500 | 62.44 | 197.34 | $1,23 \times 10^{-1}$ |

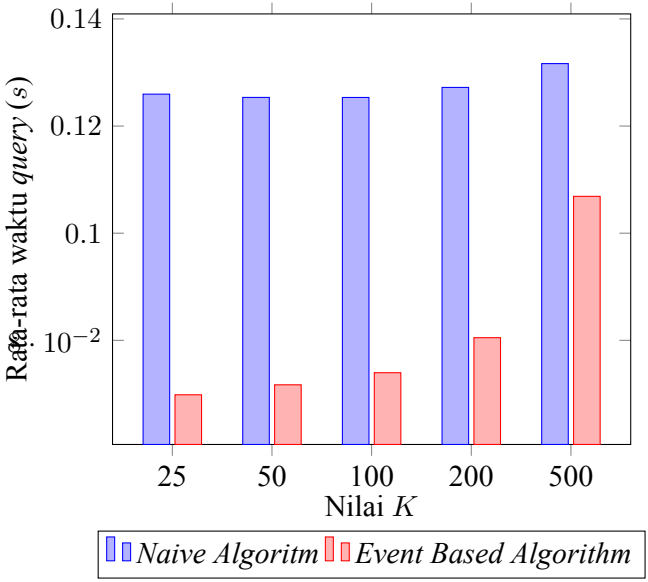
Dapat dilihat pada **Tabel 5.18** semakin tinggi nilai $K$ pada data tidak memiliki pengaruh yang signifikan pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive*.

**Tabel 5.19:** Hasil Percobaan Perubahan Nilai $K$ Algoritme *Event Based* Menggunakan Data *Anti Corelated*

| Nilai $K$ | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 25 | 62.73 | 200.21 | $6,98 \times 10^{-2}$ |
| 50 | 62.65 | 193.89 | $7,16 \times 10^{-2}$ |
| 100 | 62.64 | 198.65 | $7,39 \times 10^{-2}$ |
| 200 | 62.71 | 199.87 | $8,04 \times 10^{-2}$ |
| 500 | 62.85 | 206.45 | $1,06 \times 10^{-1}$ |

Dapat dilihat pada **Tabel 5.19** semakin tinggi nilai $K$ pada data tidak memiliki pengaruh yang signifikan pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *event based*.

Pada **Tabel 5.18** dan **Tabel 5.19** dengan mengambil data *query time* pada kolom ke empat dibentuklah grafik seperti pada **Gambar 5.8** untuk memudahkan dalam melihat perubahan yang terjadi.




**Gambar 5.8:** Perubahan Nilai $K$ Terhadap Perubahan Rata-Rata Waktu Satu *Query* Menggunakan Data *Anti Corelated*

Dari **Gambar 5.8** dapat diketahui bahwa perbedaan rata-rata waktu *query* antara algoritme *naive* dan algoritme *event based* cukup jauh. Pada setiap *case* uji coba algoritme *event based* lebih unggul dari pada algoritme *naive*. Dapat diamati pula pada data *anti corelated* perubahan rata-rata waktu *query* pada

algoritme *naive* cenderung konstan pada setiap kenaikan nilai $K$. sedangkan algoritme *event based* rata-rata waktu *query* cenderung meningkat dengan nilai yang kecil pada setiap kenaikan nilai $K$.

#### 5.4.3.3 Forest Covertype

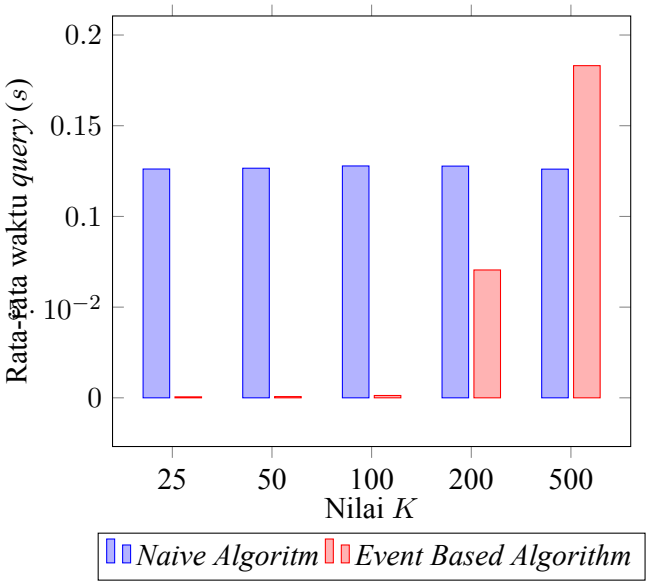
Dalam percobaan dengan data *forest covertype* didapat hasil penggunaan *memory*, *preparing time*, dan rata-rata waktu untuk menjalankan satu *query* dari algoritme *naive* seperti pada **Table 5.20** dan algoritme *event based* seperti pada **Table 5.21**.

**Tabel 5.20:** Hasil Percobaan Perubahan Nilai $K$ Algoritme *Naive* Menggunakan Data *Forest Covertype*

| Nilai $K$ | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 25 | 62.21 | 203.64 | $1,26 \times 10^{-1}$ |
| 50 | 62.37 | 197.10 | $1,26 \times 10^{-1}$ |
| 100 | 62.38 | 200.05 | $1,27 \times 10^{-1}$ |
| 200 | 62.30 | 201.29 | $1,27 \times 10^{-1}$ |
| 500 | 62.66 | 203.30 | $1,26 \times 10^{-1}$ |

**Tabel 5.21:** Hasil Percobaan Perubahan Nilai $K$ Algoritme *Event Based* Menggunakan Data *Forest Covertype*

| Nilai $K$ | Memory (byte) | Preparing Time($s$) | Query Time($s$) |
|---|---|---|---|
| 25 | 62.76 | 203.33 | $5,40 \times 10^{-4}$ |
| 50 | 62.44 | 202.32 | $6,84 \times 10^{-4}$ |
| 100 | 62.33 | 203.36 | $1,29 \times 10^{-3}$ |
| 200 | 62.27 | 207.70 | $7,04 \times 10^{-2}$ |
| 500 | 62.45 | 206.79 | $1,83 \times 10^{-1}$ |




**Gambar 5.9:** Perubahan Nilai *K* Terhadap Perubahan Rata-Rata Waktu Satu *Query* Menggunakan Data *Forest Covertype*

Dapat dilihat pada **Tabel 5.21** dan **Tabel 5.20** semakin tinggi nilai *K* pada data tidak memiliki pengaruh yang signifikan pada semakin tingginya pengunaan *memory* dan *preparing time* pada algoritme *naive* dan algoritme *event based*.

Dapat dilihat pada **Tabel 5.20** dan **Tabel 5.21** bahwa penggunaan *memory* dari kedua algoritme *naive* dan algoritme *event based* tidak jauh berbeda. Sedangkan pada *preparing time* algoritme *event based* cenderung lebih tinggi dari pada algoritme *naive*. Dari **Tabel 5.20** dan **Tabel 5.21** dengan mengambil data *query time* pada kolom ke empat dibentuklah grafik pada **Gambar 5.9** untuk memudahkan dalam melihat perubahan yang terjadi.

Dari **Gambar 5.9** dapat diketahui bahwa rata-rata waktu *query* pada algoritme *naive* cenderung konstan pada setiap

kenaikan nilai $K$. Sedangkan terjadi peningkatan yang cukup derastis pada rata-rata satu *query* pada algoritme *event based* yaitu pada nilai $K = 200$ dan $K = 500$. Hal ini terjadi karena algoritme *event based* selalu mengupdate $TOPK$. Dengan jumlah $TOPK$ yang semakin banyak pada batas bawah skor $TOPK$ akan semakin rendah. Sehingga banyak objek yang masuk tidak mendapatkan *reference* objek dan melakukan perhitungan pada semua objek yang membuat perhitungan waktu seperti algoritme *naive*. Ditambah *event based* harus melakukan *maintenance* pada *event* sehingga waktu yang dibutuhkan lebih banyak dari pada *naive based* pada nilai $K$ yang tinggi.

# BAB VI

# KESIMPULAN DAN SARAN

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

## 6.1 Kesimpulan

Dari proses perencangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa hasil sebagai berikut berikut:

1. Terdapat dua algoritme yang diusulkan dalam menyelesaikan pengolahan *top-k dominating queries* berbasis data *streaming* dengan menggunakan *incomplete* data yaitu algoritme *naive* dan algoritme *event based*.
2. Dalam implementasi pengolahan *top-k dominating queries* berbasis *streaming* dengan mengunakan *incomplete* data, algoritme *naive* melakukan perhitungan skor dominasi secara keseluruhan pada saat inisialisasi lalu meng-*update* data tersebut dengan cara membandingkan semua data pada *sliding window* jika ada data yang keluar atau data yang masuk. Sedangkan dalam implementasi algoritme *event based* melakukan inisialisasi yang sama seperti *naive* dengan menambahkan pembuatan *event* pada objek yang tidak masuk $TOPK$ sehingga saat melakukan update hanya melakukan perhitungan yang diperlukan.
3. Dalam melakukan pengujian pemakaian memory dan waktu eksekusi query terhadap dua algoritme yaitu *naive* dan *event based* terhadap perubahan dimensi(d), *window size*(n), dan nilai (k) didapatkan hasil: 1) Pemakaian memory pada semua kasus cenderung sama pada tesiap kasus, 2) Pada kasus perubahan dimensi waktu eksekusi

query algoritme naive cenderung naik bila dimensi semakin besar, sedangkan algoritme event based fluktuatif tetapi tetap lebih kecil dari pada *naive*, 3) Dalam kasus perubahan window size terjadi perubahan kenaikan waktu eksekusi query yang derastis pada algoritme *naive* bila dimensi semakin besar, sedangkan algoritme event based fluktuatif tetapi tetap lebih kecil dari pada *naive*, 4) Di kasus perubahan nilai k waktu eksekusi algoritme naive cenderung constan bila k naik, sedangkan event based cerung naik secara signifikan.

## 6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

1. Mencoba untuk menemukan algoritma yang lebih cepat saat nilai $K$ tinggi.
2. Penelitian dapat dikemangkan dengan menambahkan kondisi data terdistribusi atau data yang tidak pasti.

# DAFTAR PUSTAKA


[1] X. Miao, Y. Gao, B. Zheng, G. Chen, dan H. Cui, "Top-k Dominating Queries on Incomplete Data," *IEEE 32nd International Conference on Data Engineering (ICDE) 2016*, vol. 28, no. 1, hal. 252 – 266, Jun. 2016.

[2] M. Kontaki, A. N. Papadopoulos, dan Y. Manolopoulos, "Continuous Top-k Dominating Queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, hal. 840 – 853, Feb. 2011.

[3] M. L. Yiu dan N. Mamoulis, "Multi-dimensional top-k dominating queries," *The International Journal on Very Large Data Bases*, vol. 18, no. 3, hal. 695–718.

[4] M. E. Khalefa, M. F. Mokbel, dan J. J. Levandoski, "Skyline Query Processing for Incomplete Data."

[5] "Welcome to Python.org," 02 Juli 2018. [Daring]. Tersedia pada: https://www.python.org/. [Diakses: 02 Juli 2018].

[6] "Welcome | Flask (A Python Microframework)," 02 Juli 2018. [Daring]. Tersedia pada: http://flask.pocoo.org/. [Diakses: 02 Juli 2018].

*(Halaman ini sengaja dikosongkan)*

# LAMPIRAN A

# CODE SUMBER ALGORITMA UTAMA

## Model Data pada Algoritme *Naive*

```
1  class Model2:
2      def __init__(self, id, data, exp):
3          self.id = id
4          self.data = data
5          self.score = 0
6          self.exp = exp
7          self.time_generate = 0
8          self.parent = 0
```

**Kode Sumber A.1:** model data pada algoritme *naive*

## Model Data pada Algoritme *Event Based*

```
1  class Model5:
2      def __init__(self, id, data, exp):
3          self.id = id
4          self.data = data
5          self.score = 0
6          self.exp = exp
7          self.start_generate = 0
8          self.end_generate = 0
9          self.parent = 0
10         self.event = None
```

**Kode Sumber A.2:** model data pada algoritme *event based*

## Fungsi *Dominate*

```
1  def dominate(obj1,obj2,dimensi):
2      """
3  ␣␣␣␣this␣function␣compare␣obj1␣with␣obj2
4  ␣␣␣␣if␣obj1␣dominate␣obj2␣return␣1
```

```
    ␣␣␣␣else␣if␣obj2␣dominate␣obj1␣return␣-1
    ␣␣␣␣else␣return␣0
    ␣␣␣␣"""
        dominate = 0
        for i in range(0,dimensi):
            if obj1[i] == '-' or obj2[i] == '-':
                continue
            if obj1[i] > obj2[i]:
                if dominate == -1:
                    dominate = 0
                    break
                dominate = 1
            elif obj1[i] < obj2[i]:
                if dominate == 1:
                    dominate = 0
                    break
                dominate = -1
        return dominate
```

**Kode Sumber A.3:** fungsi *dominate*

## Fungsi *Main*

```
# import modul
import csv
from collections import deque
import time
import sys
# import algoritm
from algorithm.NaiveAlgorithm import NaiveAlgorithm
from algorithm.EventBased import EventBased
# import data model
from model.model2 import Model2
from model.model5 import Model5

if __name__ == "__main__":
    # Parameter
    if len(sys.argv) != 4:
```

```
16          print("cara␣penggunaan:␣python3␣main.py␣<bf|eva
                ↪ |na|bb>␣<jumlah␣K>␣<directory␣data>")
17          sys.exit(0)
18      algorithm_mode = sys.argv[1]
19      K = int(sys.argv[2])
20      file_path = sys.argv[3]
21      file_name = file_path.split('/')[-1].split('.')[0]
22      window_size = int(file_name.split('-')[1])
23      Dimension = int(file_name.split('-')[3])
24      jumlah_query = 100
25
26      if algorithm_mode == 'eva':
27          # Membuka file input
28          with open(file_path) as f:
29              file_data = csv.reader(f)
30              next(file_data)
31              obj_list = list()
32              for i, row in enumerate(file_data):
33                  obj = Model5(row[0], row[1:], i+
                        ↪ window_size+1)
34                  obj_list.append(obj)
35              window = deque()
36              for i in range(window_size):
37                  window.append(obj_list[i])
38              start_time = time.time()
39              EVA = EventBased(K, window, Dimension)
40              print("prepare␣time:␣---␣%s␣seconds␣---" %
41                  (time.time() - start_time))
42              insert_obj = obj_list[window_size:
                    ↪ window_size+jumlah_query]
43              start_time = time.time()
44              hasil = EVA.getTopK()
45              for i, row in enumerate(insert_obj):
46                  insert_time = time.time()
47                  EVA.update(row)
48                  hasil = EVA.getTopK()
49                  print("insert␣%d:␣---␣%s␣seconds␣---" %
50                      (i+1, (time.time() - insert_time)))
51              print("total␣time:␣---␣%s␣seconds␣---" %
52                  (time.time() - start_time))
53              print("hasil␣akhir:")
```

```
54              for row in hasil:
55                  print(row.id, row.score)
56      elif algorithm_mode == 'na':
57          # Membuka file input
58          with open(file_path) as f:
59              file_data = csv.reader(f)
60              next(file_data)
61              obj_list = list()
62              for i, row in enumerate(file_data):
63                  obj = Model2(row[0], row[1:], i+
                        ↪ window_size+1)
64                  obj_list.append(obj)
65              window = deque()
66              for i in range(window_size):
67                  window.append(obj_list[i])
68              start_time = time.time()
69              NA = NaiveAlgorithm(K, window, Dimension)
70              print("prepare␣time:␣---␣%s␣seconds␣---" %
71                  (time.time() - start_time))
72              insert_obj = obj_list[window_size:
                    ↪ window_size+jumlah_query]
73              start_time = time.time()
74              hasil = NA.getTopK()
75              for i, row in enumerate(insert_obj):
76                  insert_time = time.time()
77                  NA.update(row)
78                  hasil = NA.getTopK()
79                  print("insert␣%d:␣---␣%s␣seconds␣---" %
80                      (i+1, (time.time() - insert_time)))
81              print("total␣time:␣---␣%s␣seconds␣---" %
82                  (time.time() - start_time))
83              print("hasil␣akhir:")
84              for row in hasil:
85                  print(row.id, row.score)
```

**Kode Sumber A.4:** fungsi *main*

## Algoritme *Naive*

```
1  import heapq
2  from .Helper import dominate
3  from multiprocessing import Pool
4
5  class NaiveAlgorithm:
6      def __init__(self, K, windows, dimension):
7          self.K = K
8          self.TopK = []
9          self.windows = windows
10         self.dimension = dimension
11         self.data_lenght = len(windows)
12         self.now = len(windows)
13
14         # Compute from scrach
15         p = Pool(6)
16         hasil = p.map(self.computeFromScratch, list(
               ↪ self.windows))
17         for i in range(self.data_lenght):
18             self.windows[i] = hasil[i]
19
20         # sort window by score
21         self.sorted_window = sorted(
22             self.windows, key=lambda object: object.
                   ↪ score, reverse=True)
23
24         # get Top-K from sorted window
25         self.TopK = []
26         localK = 0
27         tao = self.data_lenght
28         for row in self.sorted_window:
29             if localK < self.K:
30                 self.TopK.append(row)
31                 localK += 1
32                 if tao > row.score:
33                     tao = row.score
34             elif tao == row.score and localK == self.K:
35                 self.TopK.append(row)
36             else:
37                 break
38
```

```
39      def getTopK(self):
40          return self.TopK
41
42      def update(self, item):
43          # Update time
44          self.now += 1
45          # take item
46          itemleft = self.windows.popleft()
47          for row in self.windows:
48              if itemleft.parent <= 0:
49                  break
50              dominate_status_out = dominate(
51                  itemleft.data, row.data, self.dimension
                        ↪ )
52              if dominate_status_out == -1:
53                  row.score -= 1
54                  itemleft.parent -=1
55
56          # compute who dominate item
57          for row in self.windows:
58              dominate_status = dominate(
59                  item.data, row.data, self.dimension)
60              if dominate_status == 1:
61                  item.score += 1
62                  row.parent += 1
63              elif dominate_status == -1:
64                  row.score += 1
65                  item.parent += 1
66
67          # insert item into windows
68          self.windows.append(item)
69
70          self.sorted_window = sorted(
71              self.windows, key=lambda object: object.
                    ↪ score, reverse=True)
72
73          # get Top-K from sorted window
74          self.TopK = []
75          localK = 0
76          tao = self.data_lenght
77          for row in self.sorted_window:
```

```
78                 if localK < self.K:
79                     self.TopK.append(row)
80                     localK += 1
81                     if tao > row.score:
82                         tao = row.score
83                 elif tao == row.score and localK == self.K:
84                     self.TopK.append(row)
85                 else:
86                     break
87 
88         def computeFromScratch(self, item):
89             for j in range(self.data_lenght):
90                 dominate_status = dominate(
91                     item.data, self.windows[j].data, self.
                        ↪ dimension)
92                 if dominate_status == 1:
93                     item.score += 1
94                 if dominate_status == -1:
95                     item.parent += 1
96             return item
97             # print(item.score, item.parent)
```

**Kode Sumber A.5:** algoritme *naive*

## Algoritme *Event Based*

```
1  import math
2  import heapq
3  from collections import deque
4  from .Helper import dominate
5  from multiprocessing import Pool
6 
7 
8  class Event:
9      def __init__(self, item, score, ept, egt):
10         self.item = item
11         self.shadowScore = score
12         self.ept = ept
13         self.egt = egt
```

```
14
15      def __lt__(self, other):
16          return self.ept < other.ept
17
18
19  class EventBased:
20      def __init__(self, K, windows, dimension):
21          self.K = K
22          self.TopK = []
23          self.windows = windows
24          self.dimension = dimension
25          self.data_lenght = len(windows)
26          self.delete_item = deque()
27          self.now = len(windows)
28          self.EventQueue = []
29
30          ## COMPUTE ALL FROM SCRATCH
31          p = Pool(12)
32          hasil = p.map(self.computeFromScratch, list(
                ↪ self.windows))
33          for i in range(self.data_lenght):
34              self.windows[i] = hasil[i]
35
36          for i in range(self.data_lenght):
37              self.windows[i].end_generate = self.now
38
39          ## SORT ALL AND TAKE TOP-K
40          sorted_window = sorted(
41              self.windows, key=lambda object: object.
                    ↪ score, reverse=True)
42          localK = 0
43          tao = self.data_lenght
44          for row in sorted_window:
45              if localK < self.K:
46                  self.TopK.append(row)
47                  localK += 1
48                  if tao > row.score:
49                      tao = row.score
50              elif tao == row.score and localK == self.K:
51                  self.TopK.append(row)
52              else:
```

```
53                  ## JIKA TIDAK MASUK TOP-K CREATE
                        ↪ SCHEDULE
54                  self.scheduleEvent(row, row.score, self
                        ↪ .now)
55
56      def getTopK(self):
57          return self.TopK
58
59      def update(self, item):
60          self.now += 1
61          self.windows.append(item)
62
63          # DELETE ITEM
64          itemleft = self.windows.popleft()
65          if itemleft.parent > 0:
66              self.delete_item.append(itemleft)
67
68          # UPDATE TOP-K DOMINATING SCORE
69          for i, row in enumerate(self.TopK):
70              if row.exp > self.now:
71                  self.TopK[i] = self.computeTheRest(row)
72              else:
73                  self.TopK.pop(i)
74
75          self.TopK = sorted(
76              self.TopK, key=lambda object: object.score,
                    ↪  reverse=True)
77          # CHECK AFTER UPDATE IT STILL HAS LENGHT K
78          if len(self.TopK) > self.K:
79              topKextention = self.TopK[self.K:]
80              topKextention.reverse()
81              for row in topKextention:
82                  if row.score < self.TopK[self.K-1].
                        ↪ score:
83                      self.scheduleEvent(
84                          self.TopK[-1], self.TopK[-1].
                                ↪ score, self.now)
85                      self.TopK.pop()
86
87          ## COMPUTE FROM SCRATCH
```

```
88          isGetReference, point = self.findReferencePoint
                ↪ (item)
89          ## INSERT TOPK JIKA TIDAK SCHEDULE ULANG
90          if isGetReference:
91              event = point.event
92              score = event.shadowScore + self.now -
                    ↪ event.egt - 1
93              if score >= self.TopK[-1].score:
94                  self.computeTheRest(item)
95                  if not self.InsertTopKD(item):
96                      self.scheduleEvent(item, item.score
                            ↪ , self.now)
97              else:
98                  self.scheduleEvent(item, score, self.
                        ↪ now)
99          else:
100             if not self.InsertTopKD(item):
101                 self.scheduleEvent(item, item.score,
                        ↪ self.now)
102
103         ## SEE IS THERE ANY EVENT NOW
104         while self.EventQueue:
105             if self.EventQueue[0].ept == self.now:
106                 event = heapq.heappop(self.EventQueue)
107                 if event.item.exp > self.now:
108                     event.item = self.computeTheRest(
                            ↪ event.item)
109                     if not self.InsertTopKD(event.item)
                            ↪ :
110                         self.scheduleEvent(
111                             event.item, event.item.
                                    ↪ score, self.now)
112             else:
113                 break
114         # print(self.now, [(x.id, x.score, x.exp) for x
                ↪  in self.TopK])
115
116     def scheduleEvent(self, item, score, now):
117         Exp1 = None
118         for row in self.TopK:
119             if Exp1 == None or Exp1 > row.exp:
```

```
120                 Exp1 = row.exp
121         ept = min((math.ceil((self.TopK[-1].score -
                ↪ score)/2)+now), Exp1)
122         if ept >= now:
123             event = Event(item, score, ept, now)
124             item.event = event
125             heapq.heappush(self.EventQueue, event)
126             return True
127         else:
128             return False
129 
130     def InsertTopKD(self, item):
131         ## JIKA LEBIH BESAR DARI PADA TOP-K YANG
                ↪ TERKECIL MAKA MASUK TOP-K
132         if item.score == self.TopK[-1].score or len(
                ↪ self.TopK) < self.K:
133             self.TopK.append(item)
134             i = len(self.TopK)-1
135             while i > 0:
136                 if item.score > self.TopK[i-1].score:
137                     self.TopK[i] = self.TopK[i-1]
138                     i -= 1
139                 else:
140                     break
141             self.TopK[i] = item
142             return True
143         elif item.score > self.TopK[-1].score:
144             self.TopK.append(item)
145             i = len(self.TopK)-1
146             while i > 0:
147                 if item.score > self.TopK[i-1].score:
148                     self.TopK[i] = self.TopK[i-1]
149                     i -= 1
150                 else:
151                     break
152             self.TopK[i] = item
153             # CHECK AFTER UPDATE IT STILL HAS LENGHT K
154             if len(self.TopK) > self.K:
155                 topKextention = self.TopK[self.K:]
156                 topKextention.reverse()
157                 for row in topKextention:
```

```
158                     if row.score < self.TopK[self.K-1].
                            ↪ score:
159                         self.scheduleEvent(
160                             self.TopK[-1], self.TopK
                                    ↪ [-1].score, self.now
                                    ↪ )
161                         self.TopK.pop()
162             return True
163         else:
164             return False
165
166     def computeTheRest(self, item):
167         start = len(self.delete_item) - (self.now -
                ↪ item.start_generate)
168         if start < 0:
169             start = 0
170
171         for i in range(start, len(self.delete_item)):
172             dominate_status = dominate(
173                 item.data, self.delete_item[i].data,
                        ↪ self.dimension)
174             if dominate_status == 1:
175                 item.score -= 1
176                 self.delete_item[i].parent -= 1
177             elif dominate_status == -1:
178                 self.delete_item[i].score -= 1
179                 item.parent -= 1
180
181         item.start_generate = self.now
182         for row in list(self.delete_item):
183             if row.parent > 0 or row.score > 0:
184                 break
185             else:
186                 self.delete_item.popleft()
187
188         if item.end_generate == 0:
189             start = 0
190         else:
191             start = self.data_lenght - (self.now - item
                    ↪ .end_generate)
192         for j in range(start, self.data_lenght):
```

```
193             dominate_status = dominate(
194                 item.data, self.windows[j].data, self.
                        ↪ dimension)
195             if dominate_status == 1:
196                 item.score += 1
197                 self.windows[j].parent += 1
198         item.end_generate = self.now
199 
200         return item
201 
202     def findReferencePoint(self, item):
203         isGetReference = 0
204         countDominate = 0
205         for row in self.windows:
206             countDominate += 1
207             dominate_status = dominate(
208                 item.data, row.data, self.dimension)
209             if dominate_status == 1:
210                 item.score += 1
211                 row.parent += 1
212             elif dominate_status == -1:
213                 if row.event:
214                     if isSameDimension(item.data, row.
                            ↪ data, self.dimension):
215                         isGetReference = 1
216                         break
217         item.end_generate = self.now - self.data_lenght
                ↪  + countDominate
218         item.start_generate = self.now
219         return isGetReference, row
220 
221     def computeFromScratch(self, item):
222         for j in range(self.data_lenght):
223             dominate_status = dominate(
224                 item.data, self.windows[j].data, self.
                        ↪ dimension)
225             if dominate_status == 1:
226                 item.score += 1
227             if dominate_status == -1:
228                 item.parent += 1
229         return item
```

```
230
231 def isSameDimension(data1, data2, dimension):
232     status = True
233     for i in range(dimension):
234         if data1[i] == '-' and data2[i] == '-':
235             continue
236         elif data1[i] == '-' or data2[i] == '-':
237             status = False
238             break
239     return status
```

**Kode Sumber A.6:** algoritme *event based*

# LAMPIRAN B

# CODE SUMBER PENDUKUNG

## Implementasi Server Flask Dengan SocketIO

```
1  from flask import Flask
2  from flask_socketio import SocketIO, send
3  import csv
4  from collections import deque
5  from algorithm.EventBased import EventBased
6  import json
7
8  from model.model5 import Model5
9
10 app = Flask(__name__)
11 app.config['SECRET_KEY'] = 'secret!'
12 socketio = SocketIO(app)
13 class Output:
14     def __init__(self, id, score):
15         self.id = id
16         self.score = score
17
18 @app.route("/")
19 def hello():
20     result = [{'id':row.id , 'score':row.score} for row
           ↪  in EVA.getTopK()]
21     socketio.emit('update',json.dumps(result))
22     return (json.dumps(result))
23
24 @socketio.on('message')
25 def handleMessage(msg):
26     print('Message:␣' + msg)
27     send(msg, broadcast=True)
28
29 @app.route("/insert")
30 def insert():
31     EVA.update(insert_obj[-1])
32     result = [{'id': row.id, 'score': row.score} for
           ↪ row in EVA.getTopK()]
33     socketio.emit('update', json.dumps(result))
34     return ("ini␣insert")
35
```

```
36 if __name__ == '__main__':
37     # print("Preparing␣Server")
38     file_path = 'data/TEST-4-d-5.csv'
39     file_name = file_path.split('/')[-1].split('.')[0]
40     window_size = int(file_name.split('-')[1])
41     Dimension = int(file_name.split('-')[3])
42     K = 2
43     with open(file_path) as f:
44         file_data = csv.reader(f)
45         next(file_data)
46         obj_list = list()
47         for i, row in enumerate(file_data):
48             obj = Model5(row[0], row[1:], i+window_size
                   ↪ +1)
49             obj_list.append(obj)
50         window = deque()
51         for i in range(window_size):
52             window.append(obj_list[i])
53         global EVA
54         EVA = EventBased(K, window, Dimension)
55         global insert_obj
56         insert_obj = obj_list[:]
57     print("server␣siap")
58
59     # app.run(host='127.0.0.1',port=8000, debug=True)
60     socketio.run(app)
```

**Kode Sumber B.1:** Implementasi Flask server mengunakan socketio

## Implementasi React Mengunakan SocketIO Client

```
1 import React, { Component } from 'react';
2 import Grid from '@material-ui/core/Grid';
3 import Paper from '@material-ui/core/Paper';
4 import io from 'socket.io-client';
5 import { BarChart } from 'react-easy-chart';
6
7 import ButtonAppBar from './components/ButtonAppBar';
8 import SimpleTable from './components/SimpleTable';
```

```
 9  import './App.css';
10
11  const root = {
12    flexGrow: 1,
13    }
14
15  const paper = {
16      textAlign: 'center',
17      marginTop: 12,
18      margin: 20,
19      minHeight:600
20    }
21
22  class App extends Component {
23    constructor(props){
24      super(props);
25      this.state = {
26        data:[]
27      }
28
29      this.socket = io('http://127.0.0.1:5000');
30    }
31
32    componentDidMount(){
33      this.socket.on('connect', function () {
34        this.send('User has connected!');
35      });
36
37      this.socket.on('update', (newData) => {
38        // let data = newData.map((score, id) => ({ id,
              ↪ score }));
39        let data = JSON.parse(newData);
40        this.setState({data});
41      });
42
43      // let newData = [8,10,28,15,6];
44      // let data = newData.map((score,id) => ({id, score
            ↪ }));
45      // this.setState({data});
46    }
47
```

```
48    render() {
49      let chartData = this.state.data.map(item => ({x:
            ↪ item.id,y:item.score}));
50      console.log(chartData)
51      return (
52        <div className="App">
53          <ButtonAppBar />
54          <div style={root}>
55            <Grid container spacing={24}>
56              <Grid item xs={4}>
57                <Paper style={paper}>
58                  <h1>Tabel Nilai</h1>
59                  <SimpleTable data={this.state.data}/>
60                </Paper>
61              </Grid>
62              <Grid item xs={8}>
63                <Paper style={paper}>
64                    <h1>Grafik Nilai</h1>
65                    <BarChart
66                      axisLabels={{ x: 'My␣x␣Axis', y: '
                            ↪ My␣y␣Axis' }}
67                      axes
68                      grid
69                      colorBars
70                      height={650}
71                      width={950}
72                      data={chartData}
73                    />
74                </Paper>
75              </Grid>
76            </Grid>
77          </div>
78        </div>
79      );
80    }
81  }
82
83  export default App;
```

**Kode Sumber B.2:** Implementasi React mengunakan SocketIO Client

## BIODATA PENULIS

**Bayu Sektiaji**, lahir pada tanggal 23 Februari 1996 di Mojokerto. Penulis merupakan seorang mahasiswa yang sedang menempuh studi S1 di Departemen Informatika Institut Teknologi Sepuluh Nopember (ITS). Memiliki beberapa hobi antara lain jogging, renang, baca buku, jalan - jalan, dan belajar hal baru. Penulis merupakan administrator dari Laboratorium Pemrograman (LP Informatika ITS). Penulis juga aktif berorganisasi di dalam kampus dengan menjadi staff pada himpunan. Pernah menjadi Koordinator National Logic Competition pada Rangkaian acara SCHEMATICS 2016. Disamping menjalankan pendidikan dan organisasi di kampus penulis juga mengerjakan projek-projek web apps di luar seperti Ujian Sekolah SMP se-Surabaya, dan PPDB SMP se-Surabaya. Penulis sendiri Juga pernah magang di perusahan multinasional yaitu di IBM (Internasional Business Machine) Indonesia di Jakarta. Penulis dapat dihubungi melalui email sektibayu@gmail.com.