TUGAS AKHIR - KI141502

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT* PADA *REVERSE SKYLINE QUERIES* MENGGUNAKAN *INCOMPLETE DATA*

TOSCA YOEL CONNERY
NRP 05111440000061

Dosen Pembimbing
Henning Titi Ciptaningtyas, S.Kom., M.Kom.
Bagus Jati Santoso, S.Kom., Ph.D.

DEPARTEMEN INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

TUGAS AKHIR – KI141502

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT* PADA *REVERSE SKYLINE QUERIES* MENGGUNAKAN *INCOMPLETE DATA*

TOSCA YOEL CONNERY
NRP 05111440000061

Dosen Pembimbing
Henning Titi Ciptaningtyas, S.Kom., M.Kom.
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 THESES – KI141502

DESIGN AND IMPLEMENTATION APPLICATION TO ANSWER WHY-NOT ON REVERSE SKYLINE QUERIES USING INCOMPLETE DATA

TOSCA YOEL CONNERY
NRP 05111440000061

Supervisors
Henning Titi Ciptaningtyas, S.Kom., M.Kom.
Bagus Jati Santoso, S.Kom., Ph.D.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT* PADA *REVERSE SKYLINE QUERIES* MENGGUNAKAN *INCOMPLETE DATA*

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
Pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh
TOSCA YOEL CONNERY
NRP: 05111440000061

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Henning Titi Ciptaningtyas, S.Kom., M.Kom. .....
   (NIP: 198407082010122004) (Pembimbing 1)
2. Bagus Jati Santoso, S.Kom., Ph.D. .....
   (NIP: 198611252018031001) (Pembimbing 2)

SURABAYA
JULI 2018

[Halaman ini sengaja dikosongkan]

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT* PADA *REVERSE SKYLINE QUERIES* MENGGUNAKAN *INCOMPLETE DATA*

Nama Mahasiswa	:	Tosca Yoel Connery
NRP	:	05111440000061
Jurusan	:	Departemen Informatika FTIK-ITS
Dosen Pembimbing 1	:	Henning Titi Ciptaningtyas, S.Kom., M.Kom
Dosen Pembimbing 2	:	Bagus Jati Santoso, S.Kom., Ph.D

ABSTRAK

Ada banyak sekali produk dengan spesifikasi berbeda-beda dijual dipasaran. Jumlah masyarakat yang menjadi konsumen dari produk-produk tersebut juga sangat banyak. Tiap konsumen mempunyai preferensi uniknya masing-masing sehingga daftar produk yang diminati oleh seorang konsumen akan berbeda dengan konsumen lainnya. Hal ini menyebabkan sebuah produk dengan spesifikasi tertentu hanya akan diminati oleh sebagian kalangan yang memiliki preferensi mendekati spesifikasi dari produk tersebut. Teknologi informasi yang telah berkembang seperti sekarang dapat digunakan dalam memecahkan masalah seperti ini demi membantu produsen dalam memaksimalkan jumlah konsumen yang bisa diperolehnya. Hal ini bisa dicapai dengan membuat sebuah sistem yang bisa memberikan rekomendasi penyesuaian spesifikasi produk agar mendapatkan perhatian konsumen yang diinginkan tanpa kehilangan satu pun konsumen yang telah dimilikinya dengan menggunakan preferensi dari tiap konsumen yang ada. Dalam bidang jaringan dan basis data, permasalahan ini disebut dengan problem why-not pada reverse skyline queries. Di dunia nyata, sering kali ditemui data preferensi yang tidak lengkap, baik karena rusak, hilang, ataupun privasi. Sistem ini bisa mengatasi masalah tersebut

dengan menggunakan metode ISkyline dan berhasil memberikan rekomendasi paling efisien tanpa kehilangan satupun konsumen yang telah ada.
Kata kunci: Data Engineering, Incomplete Data, Reverse Skyline Queries, Why-not

DESIGN AND IMPLEMENTATION APPLICATION TO ANSWER WHY-NOT QUESTIONS ON REVERSE SKYLINE QUERIES USING INCOMPLETE DATA

Student's Name	:	Tosca Yoel Connery
Student's ID	:	05111440000061
Department	:	Department of Informatics FTIK-ITS
First Advisor	:	Henning Titi Ciptaningtyas, S.Kom., M.Kom.
Secod Advisor	:	Bagus Jati Santoso, S.Kom., Ph.D.

ABSTRACT

There are many products with different specifications sold in the market. The number of people who become consumers of these products is also too much. Each consumer has their unique preference that makes their list product of interest differs from other consumers. It makes some product will only be attractive for some people who have a preference like its specification. Information technology that has developed as now can be used to solve problems like this to assist producers in maximizing the number of obtained consumers. It can be accomplished by a system that can give recommendation customizing product specifications with the intention to get the desired consumer attention, without losing any of its existing customers by using the customer preferences. In the field of networks and databases, this problem is called the why-not questions in reverse skyline queries. In the real world, often encountered incomplete data preference, either because of damage, loss or privacy. This system can solve the problem by using ISkyline method and managed to provide the most efficient recommendation without losing any existing customers.

Keywords: Data Engineering, Incomplete Data, Reverse Skyline Queries, Why-not

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

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 UNTUK MENJAWAB PERMASALAHAN *WHY-NOT* PADA *REVERSE SKYLINE QUERIES* 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.

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 dan Nabi Muhammad SAW.
2. Ibunda, Ayahanda dan Adik, yang selalu mendoakan penulis dan mendukung setiap pilihan yang penulis ambil
3. Ibu Henning Titi Ciptaningtyas, S.Kom., M.Kom. selaku pembimbing I yang selalu memberikan motivasi dan membimbing penulis selama pengerjaan tugas akhir.
4. Bapak Bagus Jati Santoso, S.Kom., Ph.D. selaku pembimbing II yang selama ini telah membantu dan membimbing penulis selama pengerjaan tugas akhir

5. Bapak Dr.Eng Darlis Herumurti, S.Kom., M.Kom. selaku Kepala Jurusan Teknik Informatika ITS, Bapak Dr. Radityo Anggoro, S.Kom., M.Sc. selaku koordinator TA, dan segenap dosen Teknik Informatika yang telah banyak memberikan ilmu kepada penulis.
6. I Putu Eka Wira Mahardika, Dwika Setya Muhammad, Setyassida Novian, Cahya Setya, Bayu Sektiaji dan Syukron Rifa'il Muhammad selaku teman seperjuangan di kampus yang telah banyak bertukar pikiran dengan penulis.
7. Rekan-rekan satu angkatan mahasiswa Informatika 2014 yang tidak lelah membantu penulis semasa masa studi.

Surabaya, Juni 2018

Tosca Yoel Connery

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Bab ini membahas garis besar penyusunan tugas akhir yang meliputi latar belakang, tujuan pembuatan, rumusan dan batasan permasalahan, metodologi penyusunan tugas akhir, dan sistematika penulisan.

1.1. Latar Belakang

Pada era yang berkembang pesat seperti sekarang ini, teknologi informasi merupakan hal penting yang menunjang segala bidang kehidupan manusia. Semua kegiatan dan pekerjaan masyarakat kini sudah mulai terintegrasi dengan teknologi dan basis data. Hal ini dilakukan dengan tujuan memaksimalkan hasil yang ingin dicapai. Untuk bisa memberikan manfaat tersebut, sebuah aplikasi sering kali melibatkan banyak data dalam proses komputasinya sehingga dibutuhkan algoritma dan struktur data yang tepat agar aplikasi tersebut bisa memberikan hasil yang benar dengan cepat. Sistem basis data standar akan mengakibatkan waktu eksekusi yang panjang sehingga diperlukan struktur data baru yang tepat berdasarkan tujuan aplikasi tersebut.

Salah satu contoh pemanfaatan teknologi jaringan dan basis data adalah sistem evaluasi untuk menemukan kekurangan sebuah produk sehingga masih ada konsumen yang tidak tertarik terhadap produk tersebut, serta mencari solusi yang efektif dan efisien untuk memperbaiki kekurangan tersebut. Hal ini sering disebut dengan problem *why-not* pada *reverse skyline queries*. *Why-not questions* telah menjadi perhatian yang cukup serius dalam dunia *data engineering* selama beberapa tahun terakhir.

Pada kenyataannya, sebuah data tidak selalu memiliki nilai yang lengkap. Ada keadaan dimana atribut yang dimiliki oleh sebuah data berada dalam keadaan kosong atau hilang. Hal ini disebut sebagai *incomplete data*. Hal ini akan sangat berpengaruh pada algoritma yang akan digunakan. Pada *skyline* terdapat sebuah relasi dominasi dimana jika a_1 mendominasi a_2 dan a_2 mendominasi a_3, maka a_1 akan secara otomatis akan mendominasi

a_3. Namun hal ini tidak berlaku jika ternyata datanya tidak lengkap. Data yang tidak lengkap bisa menyebabkan dominasi cyclic dimana a_1 mendominasi a_2, a_2 mendominasi a_3 dan a_3 mendominasi a_1.

Oleh karena itu, tujuan dari TA ini adalah untuk membuat aplikasi yang bisa memberikan semua aspek yang harus dimodifikasi dari sebuah produk yang dipasarkan agar mendapatkan lebih banyak konsumen dari daftar pelanggan yang telah dimiliki. Dengan adanya aplikasi ini produk yang dipasarkan dari sebuah perusahaan diharapkan bisa cocok dengan semua preferensi pengguna yang ada.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dijabarkan dengan:

1. Bagaimana algoritma yang tepat untuk menjawab masalah *why-not questions on reverse skyline queries* pada data yang tidak lengkap.
2. Menentukan struktur data yang paling efisien untuk permasalahan tersebut.

1.3. Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, yakni sebagai berikut:

1. Data yang paling baik adalah data yang memiliki selisih nilai yang lebih kecil dengan preferensi.
2. Nilai atribut yang akan diproses dalam algoritma ini bertipe numerik.
3. *Tools* yang digunakan untuk menyelesaikan tugas akhir ini adalah Python.
4. Data yang digunakan adalah data yang tidak lengkap (*incomplete data*).

1.4. Tujuan

Merancang dan mengimplementasikan sebuah algoritma baru untuk menjawab permasalahan *why-not* pada *reverse skyline queries* menggunakan *incomplete data* serta menentukan struktur data yang tepat untuk menyelesaikan permasalahan tersebut

1.5. Manfaat

Beberapa manfaat dari tugas akhir ini adalah:

1. Membantu perusahaan dalam mengevaluasi produk yang sedang tidak dipasarkan agar bisa menyesuaikan dengan lebih banyak *customer preference*, sehingga produk tersebut memiliki kemungkinan dibeli yang lebih tinggi.
2. Membantu perusahaan untuk mengambil keputusan yang paling efisien terhadap suatu produk.
3. Membantu perusahaan untuk memodifikasi produk tanpa kehilangan pelanggan yang telah dimiliki

1.6. Metodologi Pembuatan Tugas Akhir

Pembuatan tugas akhir dilakukan menggunakan metodologi sebagai berikut:

1. Studi literatur
   Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam perancangan dan implementasi algoritma ini, yaitu tentang *dynamic skyline*, *reverse skyline*, serta cara menemukan *skyline* untuk data yang tidak lengkap.

2. Analisis dan desain perangkat lunak
   Dimulai dengan memodalkan masalah ke dalam beberapa bentuk notasi matematika untuk mempermudah proses perancangan desain dari algoritma dan struktur data.

3. Implementasi perangkat lunak
   Algoritma ini diimplementasikan menggunakan bahasa pemrograman Python serta *website* berbasis Laravel untuk sistem antarmuka dengan pengguna.

4. Pengujian dan evaluasi
   Pengujian dan evaluasi dilakukan untuk mengetahui apakah sistem yang dibangun dengan menggunakan metode yang diusulkan telah bekerja dengan baik dan efisien. Untuk menguji keberhasilan dari sistem ini, maka akan dilakukan pengujian terhadap:

   1. Akurasi
      Pada pengujian ini akan dilihat seberapa tepat solusi yang diberikan oleh sistem ini dalam menjawab masalah *why-not* pada *reverse skyline queries*.

   2. Waktu eksekusi
      Pada pengujian ini akan dilihat seberapa cepat waktu eksekusi dari sistem ini jika dibandingkan dengan sistem lain.

   3. Penggunaan memori
      Pengujian ini akan berfokus pada pengukuran jumlah memori yang digunakan dalam proses komputasi.

5. Penyusunan buku tugas akhir
   Pada tahap ini dilakukan penyusunan laporan yang berisi dasar teori, dokumentasi dari perangkat lunak, dan hasil-hasil yang diperoleh selama pengerjaan tugas akhir. Sistematika penulisan buku tugas akhir secara garis besar antara lain:

   1. Pendahuluan
      a. Latar Belakang
      b. Rumusan Masalah
      c. Batasan Masalah
      d. Tujuan
      e. Manfaat
      f. Metodologi Pembuatan Tugas Akhir
      g. Sistematika Penulisan Laporan Tugas Akhir
   2. Tinjauan Pustaka
   3. Desain

4. Pengujian dan Evaluasi
5. Kesimpulan dan Saran
6. Daftar Pustaka

1.7. Sistematika Penulisan Laporan Tugas Akhir

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan

Pendahuluan, menjelaskan latar belakang, batasan masalah, tujuan dari pembuatan tugas akhir ini serta metodologi yang digunakan selama penyusunan.

Bab II Tinjauan Pustaka

Tinjauan Pustaka, memaparkan hasil studi literatur yang digunakan sebagai dasar untuk menyelesaikan tugas akhir ini, terdiri atas deskripsi mengenai perancangan algoritma, *skyline*, *data dominance*, *reverse skyline queries*, *incomplete data*, *why-not questions*, *dynamic skyline*, *dynamic dominance region*, *dynamic anti-dominance region*.

Bab III Desain

Pada tahap ini dijelaskan deskripsi dari sistem yang bertujuan untuk menjawab permasalahan *why-not* pada *reverse skyline queries* menggunakan *incomplete data.*

Bab IV Implementasi

Bab ini membahas implementasi dari perancangan sistem yang dilakukan pada tahap perancangan. Penjelasan implementasi meliputi implementasi program inti yang akan menerapkan gabungan kedua algoritma *why-not* pada *reverse skyline queries* dan *skyline query* untuk *incomplete data*, serta implementasi interface dalam bentuk website berbasis Laravel.

Bab V Pengujian dan Evaluasi

Pengujian dilakukan dengan cara memperhitungkan akurasi, waktu eksekusi dan memori yang digunakan dalam dalam proses komputasi.

Bab VI Kesimpulan dan Saran

Kesimpulan dan Saran, berisi tentang kesimpulan yang didapat dari proses pembuatan tugas akhir beserta saran-saran untuk pengembangan selanjutnya.

BAB II
TINJAUAN PUSTAKA

Bab ini membahas teori-teori yang mendukung pembuatan tugas akhir. Teori yang mendukung tersebut adalah deskripsi mengenai *skyline*, *data dominance*, *dynamic skyline*, *reverse skyline*, *dynamic anti-dominance region*, *skyline* pada *incomplete data*, *why-not question*, *virtual point* dan *shadow skyline*.

2.1. Daftar Istilah

Daftar istilah yang digunakan pada penelitian ini dapat dilihat pada Tabel 2.1.

Tabel 2.1 Tabel istilah

Simbol	**Keterangan**
$window\_query$	Cara yang digunakan untuk mendapatkan Wilayah diantara dua titik untuk memeriksa ada atau tidaknya titik lain yang mendominasi
DSL	Semua data yang menjadi *skyline* dari sebuah preferensi pengguna
DDR	Wilayah yang didominasi oleh *skyline*
$\overline{DDR}$	Wilayah yang tidak didominasi oleh *skyline*
$a \prec_q b$	Variabel *a* mendominasi variabel *b* terhadap titik acuan q
$Local\ skyline$	*Skyline* dari data yang memiliki representasi *bitmap* yang sama
$Candidate\ skyline$	Kumpulan *local skyline* yang akan menjadi kandidat sebagai *global skyline*
$Global\ skyline$	*Skyline* sebenarnya dari data yang tidak lengkap
$Query\ point$	Spesifikasi sebuah produk yang diperiksa
Why-not point	data *p closely dominates q* pada CDG
Λ	Seluruh titik yang didapatkan dari hasil *window_query*

2.2. *Skyline* dan *Data Dominance*

Skyline adalah sejumlah data yang tidak didominasi oleh data lain dalam sekumpulan data. *Skyline* bisa dianggap sebagai data yang memiliki nilai yang lebih baik jika dibandingkan dengan seluruh data disekelilingnya. Nilai yang lebih baik disini bisa diterjemahkan sebagai nilai yang lebih tinggi atau rendah, tergantung kebutuhan [1].

Data Dominance adalah data yang memiliki atribut yang paling tidak sama dengan atribut pada data lainnya, serta memiliki paling tidak satu atribut yang lebih unggul dari atribut pada data lainnya. Sebuah data *a* tidak bisa disebut mendominasi *b* jika ada paling tidak satu atribut di data *b* yang memiliki nilai lebih baik dari pada *a*, meskipun *a* memiliki atribut-atribut yang cenderung lebih baik dari pada *b*. Konsep inilah yang digunakan digunakan untuk mengeliminasi sekumpulan data untuk mendapatkan *skyline*-nya. Hanya data yang tidak terdominasi oleh data lain yang muncul sebagai *skyline*.

Tabel 2.2 Contoh dataset empat dimensi

ID	Atribut 1	Atribut 2	Atribut 3	Atribut 4
p_1	4	5	7	4
p_2	2	5	2	3
p_3	4	3	4	7
p_4	3	6	2	5
p_5	1	3	2	1

Dengan menggunakan anggapan bahwa nilai yang lebih tinggi adalah nilai yang lebih baik, maka pada Tabel 2.2, dapat disimpulkan bahwa produk p_1, p_3 dan p_4 tergolong sebagai *skyline* dari semua poduk. Produk p_2 didominasi oleh p_1 dan p_4. Sedangkan produk p_5 didominasi oleh semua produk lainnya.

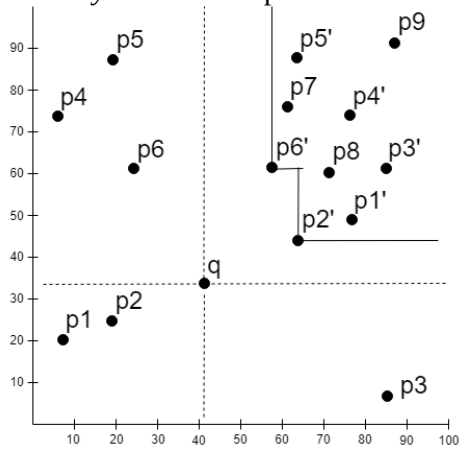
Pada *skyline* juga berlangsung sebuah relasi transitif dimana jika *a* mendominasi *b* dan *b* mendominasi *c*, maka dipastikan *a* juga mendominasi *c*. Hal ini akan sangat menghemat proses komputasi karena jika *a* diketahui mendominasi *b*, maka

semua data yang didominasi oleh *b* sudah pasti didominasi juga oleh *a*.

Dalam peneliatian ini, *skyline* akan digunakan sebagai konsep dasar untuk menentukan apakah sebuah produk yang diproduksi bisa menjadi salah satu barang yang diminati atau tidak.

2.3. *Dynamic Skyline*

Dynamic skyline adalah sebuah *skyline* yang dibuat berdasarkan sebuah data tertentu sebagai titik acuan. Secara umum, *DSL*(*c*) adalah garis yang membatasi *DDR*(*c*) dan $\overline{DDR}(c)$ [1]. Nilai atribut yang akan dibandingkan dalam *dynamic skyline* didapat dengan menggunakan semantik *around-by*, yaitu selisih nilai pada tiap atribut dalam data acuan terhadap atribut yang sama dari data disekelilingnya. Untuk mempermudah proses membandingkan data mana saja yang menjadi *dynamic skyline*, semua data disekitar acuan akan ditransformasikan kedalam satu kuadran yang sama dengan titik acuan sebagai pusatnya. Pada penelitian ini, *dynamic skyline* digunakan untuk akan digunakan untuk menentukan *skyline* dari setiap konsumen.



Gambar 2.1 Dynamic skyline terhadap titik q

Pada Gambar 2.1, semua titik yang ada disekitar *query-point q* ditransformasikan terhadap *q* kedalam kuadran yang sama.

Dari hasil transformasi, titik p_2' dan p_6' berhasil mendominasi titik-titik lainnya. Sehingga titik p_2 dan p_6 menjadi *dynamic skyline* dari q.

2.4. *Reverse Skyline*

Reverse skyline adalah sekumpulan data yang bertindak sebagai preferensi dan secara bersamaan menjadikan sebuah *query point* yang sama sebagai *dynamic skyline*-nya [1]. *Reverse skyline* digunakan untuk menentukan siapa saja konsumen yang tertarik dengan sebuah produk dengan spesifikasi tertentu.

Pada Gambar 2.1, p_2 dan p_6 adalah *dynamic skyline* dari q. Maka q adalah *reverse skyline* dari p_2 dan p_6. Dalam penelitian ini *reverse skyline* akan digunakan untuk menentukan semua konsumen yang tertarik dengan *query point* q.

2.5. *Dynamic Anti-Dominance Region*

Dynamic anti-dominance region atau $\overline{DDR}$ adalah wilayah yang terletak diantara titik acuan dan garis *skyline*. Sembarang titik baru yang diletakkan didalamnya akan secara otomatis menjadi *skyline* baru. $\overline{DDR}$ akan digunakan untuk menentukan wilayah aman (*safe region*) bagi perpindahan *query point* tanpa perlu khawatir didominasi oleh titik lain. Wilayah diluar $\overline{DDR}$ disebut dengan dynamic dominance region (*DDR*) yang berisi semua titik-titik yang didominasi oleh skyline [1].

2.6. *Skyline* pada *Incomplete Data*

Incomplete data adalah keadaan dimana terdapat nilai yang hilang pada atribut sebuah data. Perlu diingat ketika dua buah data dibandingkan dalam proses penentuan *skyline*, hanya dimensi dengan data yang lengkap dikedua sisilah yang bisa dibandingkan. Sedangkan dimensi yang tidak lengkap tidak perlu dibandingkan.

Hal ini mengakibatkan algoritma *skyline query* biasa tidak bisa lagi digunakan karena jika *a* mendominasi *b* dan *b* mendominasi *c*, bisa saja *c* justru mendominasi *a*. Hal ini disebut dengan *cyclic dominance relation* [2] [3].

Data yang tidak lengkap ini bisa disebabkan oleh beberapa hal seperti rusaknya perangkat penyimpanan yang digunakan, *data*

loss, ataupun karena privasi dimana pemilik data menolak untuk memperlihatkannya sehingga data tersebut tidak bisa dilibatkan dalam proses penentuan *skyline*.

Tabel 2.3 Contoh dataset tidak lengkap empat dimensi

ID	Atribut 1	Atribut 2	Atribut 3	Atribut 4
p_1	4	5	7	-
p_2	-	5	2	3
p_3	-	3	4	7
p_4	3	-	2	-
p_5	1	3	2	1

Sebagai contoh, pada Tabel 2.3 data p_1 dan p_2 hanya memiliki dua atribut yang lengkap dikedua data, yaitu atribut 2 dan atribut 3. Maka hanya kedua atribut inilah yang dibandingkan dalam proses penentuan dominasi. Berdasarkan tabel tersebut, dapat disimpulkan bahwa p_1 mendominasi p_2 karena atribut 3 pada p_1 memiliki nilai yang lebih baik daripada p_2 serta tidak ada atribut di p_2 yang lebih baik daripada p_1.

Akibat terjadinya relasi *cyclic* dan *non-transitive dominance relation* pada data yang tidak lengkap, maka proses penentuan *skyline* harus menggunakan algoritma *i-skyline*. Dalam algoritma ini, data yang diproses harus melalui beberapa tahap agar bisa benar-benar digolongkan sebagai *skyline* pada kondisi data yang tidak lengkap. Data yang dibandingkan harus menjadi *local skyline* pada tahap pertama dalam proses tersebut, kemudian menjadi *candidate skyline* ditahap kedua, dan akhirnya menjadi *global skyline* di tahap terakhir.

Pada tahap pertama atau *local skyline*, setiap data yang diperiksa akan dipisahkan kedalam *bucket-bucket* tertentu. Sebuah *bucket* tersusun dari data yang memiliki representasi *bitmap* yang sama dan setiap *bucket* memiliki representasi *bitmap* yang unik. Reprentasi *bitmap* mengacu pada ada atau tidaknya nilai pada sebuah atribut atau dimensi. Hasil *skyline* dari masing-masing *bucket* ini disebut dengan *local skyline*.

Jika sebuah data berhasil menjadi *local skyline*, maka akan diteruskan kedalam tahap *candidate skyline*. Pada tahap ini data yang sedang diproses akan dibandingkan dengan *local skyline* dari *bucket* lain. Jumlah *candidate skyline* dibatasi sebanyak *t* data agar tidak terlalu banyak data yang akan dibandingkan. Ketika jumlah *candidate skyline* telah melewati batas *t* maka akan dilakukan pembaruan *global skyline*.

Tahap terakhir adalah *global skyline*. Dalam proses ini, seluruh *candidate skyline* akan dibandingkan dengan *global skyline*. Semua *global skyline* dan *candidate skyline* yang tidak terdominasi akan bertahan sebagai *global skyline* serta variabel *candidate skyline* akan diinisialisasi kembali. Semua data yang berhasil bertahan hingga akhir dalam proses inilah yang bisa disebut sebagai *skyline* yang sebenarnya dari kumpulan *incomplete data*.

2.7. *Why-not Questions* pada *Incomplete Data*

Why-not questions adalah pertanyaan yang mempermasalahkan mengapa data atau objek tertentu yang diharapkan untuk menjadi bagian dari hasil *query* ternyata tidak muncul sebagai hasil. Jawaban dari pertanyaan ini akan menjelaskan bagian apa saja yang harus disesuaikan agar data tersebut bisa muncul sebagai hasil *query*.

Ada beberapa cara yang bisa digunakan untuk menjawab masalah *why-not questions*, yaitu dengan cara memperbaiki *why-not point* (preferensi pengguna yang tidak menjadi *RSL*), memperbaiki *query point* (nilai tiap atribut atau spesifikasi produk), maupun memperbaiki keduanya. Hal ini harus dilakukan dengan usaha seminimal mungkin serta tanpa mengakibatkan hilangnya pelanggan dari daftar pelanggan yang telah dimiliki sebelumnya.

Dengan menggunakan cara pertama, *why-not point ct* akan dipindahkan ke lokasi yang aman agar *query point q* tidak didominasi oleh titik lain. Hal yang harus dilakukan pertama kali adalah menentukan semua titik terluar yang berada diantara *why-not point* dan *query point*. Titik-titik terluar bisa didapat dengan

menjalankan sebuah *window query* yang berpusat di *ct* sesuai dengan persamaan

$$\Lambda \leftarrow window\_query(ct, q). \quad (2.1)$$

Dari hasil *window query* kemudian dipilih titik yang memenuhi persamaan

$$\forall e_1 \in F, \nexists e_2 \in \Lambda : e_2 \prec_q e_1 . \quad (2.2)$$

Lokasi yang bisa digunakan harus berada paling tidak tepat ditengah-tengah antara titik terluar dan *query point* pada tiap dimensi *ct*.

$$u_l^i = e_l^i + \frac{|e_l^i - q^i|}{2}, \forall_i \in \{1, \ldots, d\} \quad (2.3)$$

Cara berikutnya adalah dengan memperbaiki *query point* yang digunakan. Perpindahan *query point* harus dengan memperhatikan *safe region*(q). Wilayah ini adalah irisan dari semua wilayah $\overline{DDR}$ dari semua *RSL*(q) agar tidak menyebabkan hilangnya konsumen yang telah ada sebelumnya. Hal ini hanya bisa dilakukan jika *SR*(q) beririsan dengan $\overline{DDR}$(*ct*).

Cara terakhir dilakukan jika ternyata *dynamic anti-dominance region* dari data yang menjadi *why-not point* ternyata tidak memiliki area yang sama dengan irisan semua *dynamic anti-dominance region* dari *reverse skyline*, yaitu memindahkan *why-not point* serta *query point*-nya.

2.8. *Virtual Points*

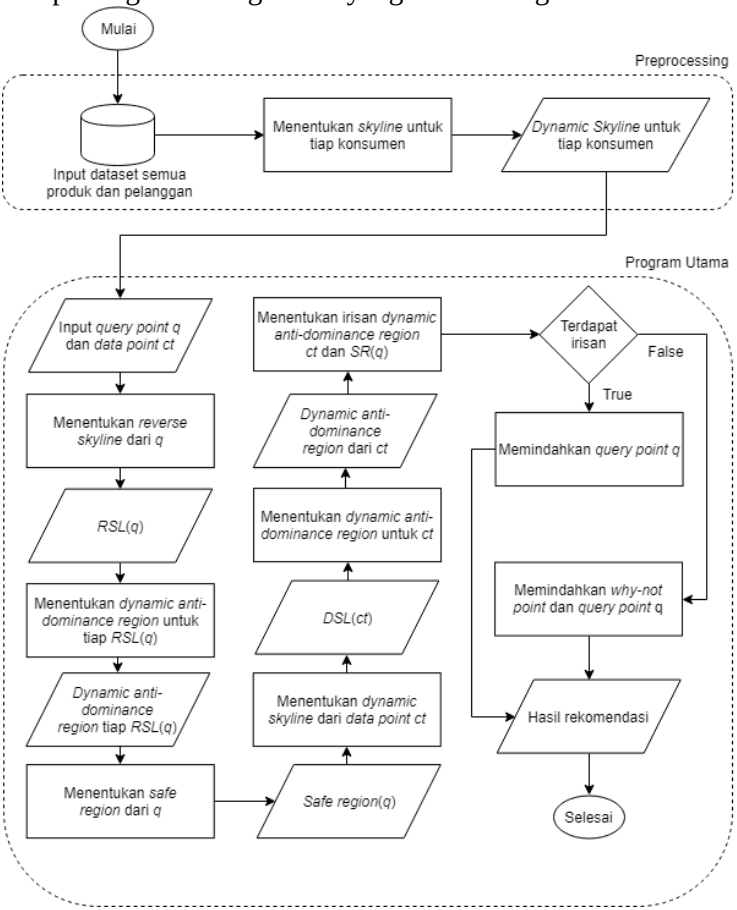
Ide utama penggunaan *virtual points* adalah untuk mengurangi jumlah proses perbandingan *candidate skyline*. Hal ini bisa dilakukan dengan menyaring secara lebih awal calon *candidate skyline* pada proses insersi *local skyline*. *Virtual point* merupakan data-data dari *bucket* lain yang berfungsi sebagai penyaring pada suatu *bucket* sehingga jumlah *local skyline* yang maju sebagai *candidate skyline* berkurang.

Sebelum sebuah data dimasukkan kedalam *local skyline* dari *bucket*-nya, data tersebut akan dibandingkan terlebih dahulu dengan *virtual point* yang telah ditetapkan pada *bucket* tersebut. Sebuah data tidak akan dijadikan *local skyline* jika terdominasi oleh *virtual point*-nya. *Virtual point* didapatkan ketika proses

insersi *candidate skyline* dimana *local skyline* maupun *candidate skyline* yang terdominasi dalam proses ini akan dijadikan *virtual point* pada *bucket* yang mendominasinya. Penggunaan *virtual point* pada penelitian ini akan membantu mengeliminasi data pada *local skyline* harusnya terdominasi oleh sebuah *candidate skyline*, namun *candidate skyline* tersebut telah lebih dulu terdominasi oleh data lain. *Candidate skyline* yang terdominasi akan dihapus dari *list*, hal ini bisa menyebabkan *local skyline* yang harusnya terdominasi oleh *candidate skyline* tersebut tetap menjadi *candidate skyline* baru. Hal inilah yang menyebabkan dibutuhkannya *virtual point* untuk menyaring *local skyline* yang akan masuk ke tahap berikutnya [1].

BAB III
DESAIN

Pada bab ini akan dipaparkan desain tugas akhir yang meliputi tahap-tahap penyelesaian tugas akhir, baik secara umum maupun algoritma-algoritma yang mendukung secara khusus.



Gambar 3.1 Gambaran umum sistem

3.1. Desain Umum Sistem

Tujuan dari sistem ini adalah mendapatkan sebuah rekomendasi perubahan nilai variabel pada spesifikasi produk dan preferensi pengguna jika produk tersebut tidak menjadi pilihan yang menarik bagi pengguna tersebut, dengan catatan dimana jika hasil modifikasi tersebut diterapkan, tidak boleh ada satupun konsumen yang hilang dari daftar pelanggan yang telah ada. Sebuah produk dapat dikatakan menarik jika memiliki spesifikasi yang tidak didominasi oleh produk lain terhadap preferensi konsumen tersebut. Setiap konsumen bisa memiliki preferensi yang berbeda dari konsumen lainnya sehingga daftar produk yang diminatinya juga akan beragam.

Algoritma yang diajukan ini untuk selanjutnya akan disebut dengan algoritma berbasis *safe region*. Secara umum langkah-langkah yang harus dilakukan akan dijelaskan sebagai berikut. Sistem akan mengolah semua data preferensi konsumen yang telah dimiliki dari daftar pelanggan untuk mendapatkan DSL dari tiap preferensi.

Tahap berikutnya, program akan menentukan *skyline* dari konsumen *ct* yang akan ditinjau ketertarikannya terhadap produk *q* dengan cara membandingkan semua data spesifikasi produk di pasaran dan produk *q*. Jika *q* muncul sebagai *DSL* dari *ct*, maka dapat diartikan bahwa konsumen *ct* menganggap produk *q* sebagai produk yang menarik. Dalam hal ini, tidak perlu dilakukan penyesuian sama sekali pada spesifikasi produk *q* dan preferensi konsumen *ct*. Namun jika ternyata produk *q* tidak menjadi *DSL* dari *ct*, maka perlu dilakukan penyesuaian untuk menjadikan produk *q* sebagai barang yang diminati oleh konsumen *ct*.

Kemudian program akan menentukan wilayah $\overline{DDR}(c_t)$ dan $SR(q)$. Perubahan dilakukan berdasarkan hasil irisan dari wilayah $SR(q)$ dan $\overline{DDR}(c_t)$. Jika kedua wilayah ini beririsan maka hanya perlu dilakukan penyesuaian pada spesifikasi produk *q*. Jika kedua wilayah tersebut tidak beririsan, maka perlu dilakukan penyesuaian pada produk *q* dan preferensi konsumen *ct*.

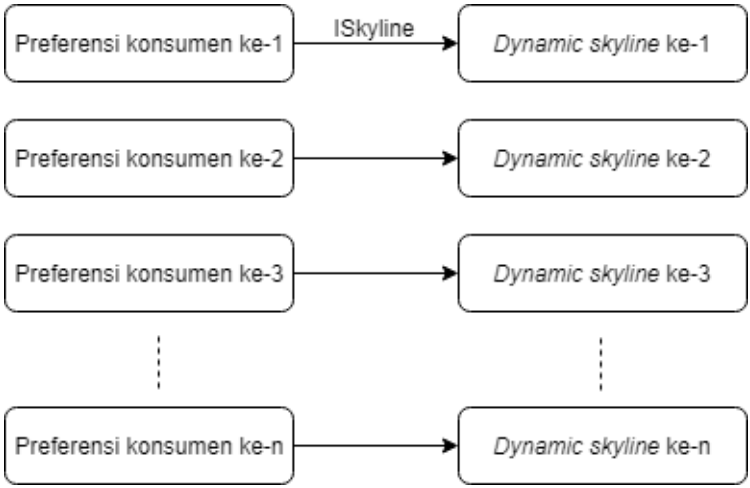
Program akan menerima input berupa sebuah preferensi user c_t serta sebuah spesifikasi produk q untuk kemudian ditentukan apakah diperlukan perubahan nilai pada spesifikasi produk q maupun preferensi user c_t tersebut. Pada tahap awal, masing-masing preferensi pengguna c sebagai pelanggan yang telah lebih dahulu eksis akan dijadikan acuan dalam proses penghitungan *skyline*. Tiap *skyline* akan digunakan untuk menentukan wilayah *dynamic anti-dominance region* ($\overline{DDR}$) dari c_t.

Irisan dari tiap dimensi dalam $\overline{DDR}(c_t)$ ini akan menghasilkan *safe region* dari dari q yang memungkinkan pergerakan nilai q pada tiap dimensi tanpa menghilangkan satupun *existing user*.

3.1.1. Preprocessing

Tahap pertama yang akan dijalakankan adalah *preprocessing*. Tahap ini meliputi proses penentuan *DSL* untuk tiap konsumen. Proses penentuan *DSL* untuk data yang tidak lengkap dilakukan dengan mengadopsi algoritma *ISkyline*. Tiap data harus bisa memenuhi tiga tahap *skyline* agar bisa disebut sebagai *skyline* pada data yang tidak lengkap. Tahap pertama adalah *local skyline* dimana sebuah data tidak boleh terdominasi oleh *virtual points* dan *local skyline* lain pada *bucket* yang sama. Data yang berhasil menjadi *local skyline* akan dibandingkan dengan semua *candidate skyline*. Setelah jumlah *candidate skyline* mencapai angka tertentu, maka proses pembaruan *global skyline* akan dilakukan. *Global skyline* ini adalah gabungan antara *candidate skyline* dan *global skyline* itu sendiri yang tidak didominasi oleh data lainnya, serta tidak didominasi oleh *shadow skyline*.

Perlu diperhatikan bahwa dalam hal ini spesifikasi produk dengan selisih paling kecil terhadap preferensi pengguna adalah produk yang paling menarik bagi tiap pengguna.



Gambar 3.2 Penentuan DSL tiap konsumen yang sudah ada

DSL ini nantinya akan digunakan dalam proses penentuan *RSL*(*q*) dimana sistem hanya perlu membandingkan *q* dengan produk-produk lain yang berhasil menjadi *skyline* dari tiap konsumen. proses ini secara umum akan lebih cepat dari pada

membandingkan produk *q* dengan semua produk yang ada dipasaran terhadap semua preferensi user secara berulang-ulang sebanyak jumlah konsumen yang ada dalam daftar pelanggan. Pada Tabel 3.1 dapat dilihat bahwa tiap preferensi konsumen akan menghasil satu buah *dynamic skyline* yang akan digunakan sebagai acuan penentuan wilayah *SR*(*q*).

3.1.2. Proses Utama

Setelah tahap *preprocessing* selesai, sistem akan menggunakan hasil dari proses tersebut didalam proses utama. Pada proses ini sistem akan memberikan rekomendasi perubahan paling efisien yang bisa dilakukan.

3.1.2.1. Menentukan *Safe Region* dari q dan *Dynamic Anti-Dominance Region* dari ct

Query point q akan dibandingkan dengan semua *skyline* dari data konsumen. Proses yang dilakukan akan sama dengan proses pencarian *skyline* sebelumnya. Kemudian dilakukan pengecekan konsumen mana saja yang menjadikan *q* sebagai *dynamic skyline*-nya. Semua konsumen memenuhi kriteria tersebut akan dijadikan *reverse skyline* dari *q*. Untuk menentukan *safe region* dari *q*, diperlukan irisan $\overline{DDR}$ dari seluruh *reverse skyline q*.

Berikutnya, seluruh data produk *p* akan dihitung kembali *skyline*-nya dengan menggunakan *ct* sebagai titik acuannya. Kemudian dari titik-titik *skyline* ini bisa ditentukan $\overline{DDR}$ dari *ct*.

3.1.2.2. Menentukan Perubahan Nilai Paling Efisien

Untuk menentukan perubahan apa yang harus dilakukan, terlebih dahulu dilakukan pengecekan apakah $\overline{DDR}(ct)$ beririsan dengan $SR(q)$. Jika terdapat irisan diantara keduanya, maka tindakan yang harus dilakukan hanyalah mengubah nilai dai *query point q*. Namun jika keduanya tidak memiliki irisan, perlu dilakukan perubahan nilai pada *query point q* dan *why-not point ct* secara bersamaan. Dalam beberapa kasus, *query point q* bisa saja tidak bisa diubah karena wilayah $SR(q)$ hanya mencakup titik *q* itu sendiri. Hal ini disebabkan oleh beberapa konsumen yang tertarik terhadap *q* namun memiliki preferensi yang sangat berbeda sehingga wilayah $\overline{DDR}$ yang dari tiap konsumen hanya berpotongan pada titik *q* itu sendiri.

3.2. Algoritma Pembanding

Solusi yang ditawarkan dari hasil penelitian ini perlu dibandingkan dengan algoritma lain agar bisa ditentukan seberapa efektif solusi baru tersebut. Algoritma *brute force* dijadikan pembanding dengan cara mencoba satu persatu tiap kemungkinan perubahan yang terjadi. Perubahan akan tetap berada dalam

wilayah $SR(q)$ sehingga jumlah $RSL(q)$ tidak akan berkurang akibat kehilangan konsumen. Dari tiap kemungkinan perubahan yang ada, akan dicek perubahan mana saja yang dapat menyebabkan q muncul sebagai *DSL* dari q serta berapa biaya yang diperlukan untuk melakukan tiap perubahan tersebut. Sistem kemudian akan mencari perubahan dengan biaya yang paling kecil untuk direkomendasikan.

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Dalam bab ini juga akan dijelaskan tentang lingkungan yang digunakan dalam implementasi sistem.

4.1. Lingkungan Implementasi Sistem

Spesifikasi perangkat keras serta perangkat lunak yang digunakan dalam tahap implementasi perangkat lunak tugas akhir ini seperti dijelaskan pada Tabel 4.1.

4.2. Implementasi *Preprocessing*

Secara garis besar, pengimplementasian *preprocessing* sesuai dengan rancangan pada bab desain. *Preprocessing* digunakan untuk mempersiapkan seluruh *dynamic skyline* dari tiap preferensi pengguna. Dengan ditentukannya seluruh *dynamic skyline* dari tiap preferensi pengguna lebih awal, maka proses pencarian *RSL*(q) bisa dilakukan dengan hanya membandingkan hasil transformasi *query point* ke semua *dynamic skyline* dari tiap preferensi pengguna. Jika *query point* bisa bertahan tanpa terdominasi, maka preferensi pengguna tersebut dijadikan sebagai anggota dari *RSL*(q).

Tabel 4.1 Lingkungan Implementasi Sistem

Spesifikasi	**Deskripsi**
CPU	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
Memori Primer	32GB
Memori Sekunder	1TB
Arsitektur Prosesor	64-bit
Sistem Operasi	Ubuntu 16.04.4 LTS
Perangkat Pengembang	Python 3

Hal ini tentunya sangat memangkas biaya komputasi tanpa perlu membandingkan secara berulang-ulang seluruh data produk demi mendapatkan *RSL*(*q*) untuk tiap data *q* yang ingin diproses. Dalam bagian *preprocessing* ini juga akan dilibatkan beberapa fungsi yang juga akan digunakan dalam program utama seperti fungsi untuk menyisipkan *local skyline*, *candidate skyline* dan *global skyline*.

Fungsi utama yang digunakan dalam *preprocessing* adalah fungsi generate_all_dsl. Fungsi ini akan melakukan iterasi pada semua preferensi pengguna untuk menentukan *dynamic skyline* pada tiap preferensi pengguna. *DSL* yang didapat dari proses inilah yang akan dijadikan dasar penentuan *RSL*(*q*) untuk sembarang nilai *q* yang diinputkan nantinya. Pseudocode yang diimplementasikan untuk fungsi generate_all_dls dapat dilihat pada Gambar 4.1.

Fungsi generate_all_dsl()	
Variabel Global	-
Masukan	-
Keluaran	-

```
1.  customer_skyline ← {}
2.  for all pref ∈ list_customer do
3.      for all spec ∈ product_specification do
4.          is_skyline ← Insert_Local_Skyline(transform(spec), bitmap)
5.          if is_skyline = TRUE then
6.              Insert_Candidate_Skyline(transform(spec), bitmap)
7.          end if
8.          update global_skyline when candidate_skyline reach a certain number
9.      customer_skyline = customer_skyline U global_skyline
```

Gambar 4.1 Pseudocode fungsi mendapatkan seluruh DSL

4.3. Implementasi Program Utama

Algoritma program utama diimplementasikan sesuai dengan rancangan yang telah dijelaskan pada bab desain. Program utama terdiri dari beberapa bagian. Beberapa fungsi dari program utama akan digunakan baik pada tahap preprocessing maupun pada program utama itu sendiri.

4.3.1. Fungsi Insersi *Local Skyline*

Pada bagian ini akan dijelaskan pseudocode tahap pertama dalam algoritma *ISkyline*. Sebuah point *P* akan ditinjau apakah titik tersebut mendominasi atau didominasi oleh titik lain yang ada didalam *local_skyline* dari *bucket* yang sama. Jika *P* didominasi oleh *local_skyline*, maka proses akan langsung dihentikan dan mengembalikan nilai *False* sebagai pertanda bahwa *P* tidak berhasil menjadi *local_skyline* pada *bucket B*. Jika ternyata *P* tidak didominasi oleh *local_skyline*, maka *P* kemudian dibandingkan dengan *virtual point* dari *bucket B*. Jika *virtual point* juga tidak mendominasi *P*, maka *P* akan menjadi *local skyline* dan semua data yang terdominasi dalam *virtual point* dihapus.

Jika ternyata *P* hanya didominasi oleh *virtual point*, maka *P* akan dimasukkan kedalam *shadow skyline* dan *node n* akan ditandai sebagai telah diupdate. Semua *shadow skyline* yang terdominasi kemudian akan dihapus. Jika ternyata *P* tidak didominasi sama sekali oleh apapun, maka *P* akan langsung dijadikan *local skyline* serta semua *local skyline* dan *shadow skyline* yang terdominasi akan dihapus.

Fungsi insert_local_skyline(*Point P, Bitmap B*)	
Variabel Global	*local_skyline, shadow_skyline, virtual_point*
Masukan	*Point P, Bitmap B*
Keluaran	Boolean
1. **if** *P* is not dominated by any point in the *local_skyline* list on bucket *B* 2. Insert *P* into *local_skyline* list of bucket *B*	

```
3.          Delete all real points that are dominated by P from
            the local_skyline and shadow_skyline list of bucket B
4.          return true
5.    else if P is dominated only by virtual_point then
6.          Insert P into shadow_skyline of bucket B
7.          N.updated_flag ← true
8.          Delete all points that are dominated by P from the
            shadow_skyline list
9.    end if
10.   return false
```

Gambar 4.2 Pseudocode fungsi insersi local skyline

4.3.2. Fungsi Insersi *Candidate Skyline*

Tahap kedua dalam algoritma *ISkyline* adalah *candidate skyline*. Semua data yang berhasil menjadi *local skyline* akan langsung dibandingkan dengan *candidate skyline* lainnya. Setiap titik yang terdominasi baik *candidate skyline* maupun *P* akan dimasukkan ke *virtual point*.

Fungsi insert_candidate_skyline(*Point P, Bitmap B*)	
Variabel Global	*candidate_skyline*
Masukan	*Point P, Bitmap B*
Keluaran	-

```
1.    for each point Q ∈ candidate skyline where P and Q are
      compatible do
2.       if P dominates Q then
3.          Delete Q from candidate_skyline
4.          Insert_Virtual_Point(P, bitmap Q)
5.       else if Q  dominates P then
6.          Insert_Virtual_Point(Q, bitmap P)
7.       end if
8.       check if P is dominated by virtual_point in bucket B
9.    end for
10.   if P is not dominated by any point, then insert P in
      candidate_skyline list
```

Gambar 4.3 Pseudocode fungsi insersi candidate skyline

4.3.3. Fungsi *Update Global Skyline*

Ini adalah tahap terakhir dari ISkyline. Algoritma ini dijalankan setiap kali jumlah data dalam variabel *candidate_skyline* telah mencapai jumlah tertentu. Algoritma ini membandingkan semua data yang adapa pada *global_skyline* dan *candidate_skyline*. Tiap data *global_skyline* yang tereliminasi akan dihapus. Seluruh *candidate_skyline* akan yang tereliminasi oleh *shadow_skyline* juga akan dihapus. Pada bagian terakhir, *candidate_skyline* dan *global_skyline* akan ditentukan digabungkan sebagai *global_skyline* yang baru.

Fungsi update_global_skyline()	
Variabel Global	*global_skyline, candidate_skyline, shadow_skyline*
Masukan	-
Keluaran	-

1. **for each** pair of comparable points $P \in$ *global_skyline* and $Q \in$ *candidate_skyline* **do**
2. **if** *P* dominates *Q* or *Q* dominates *P*, **then** mark the dominated point
3. **end for**
4. Delete all marked points from *candidate_skyline* and *global_skyline* lists
5. **for each** point $P \in$ *global_skyline* **do**
6. **for each** *n_updated_flag* with *true* value **do**
7. **if** any point in *shadow_skyline* on bitmap *n* dominates *P*, then delete *P* from the *global_skyline* list
8. **end for**
9. **end for**
10. **for each** point $Q \in$ *candidate_skyline* **do**
11. **for each** node *N* with comparable bitmap to *Q*
12. **if** any point in *N shadow_skyline* list dominates *Q*, **then** delete *Q* from *candidate_skyline* list
13. **end for**

```
14. end for
15. global_skyline ← global_skyline U candidate_skyline
16. set all updated flags to false
```

Gambar 4.4 Pseudocode fungsi memperbarui global skyline

4.3.4. Fungsi Insersi *Virtual Point*

Pada bagian ini akan dijelaskan pseudocode untuk melakukan insersi *virtual_point*. Fungsi ini dijalankan ketika ada data yang terdominasi ketika proses insersi *candidate skyline*. Data yang terdominasi inilah yang akan dijadikan *virtual point* pada *bucket* dari data yang mendominasinya. Proses insersi *virtual point* adalah sebagai berikut

Fungsi insert_virtual_point(*Point P, Bitmap B*)	
Variabel Global	*local_skyline, virtual_point, shadow_skyline*
Masukan	*Point P, Bitmap B*
Keluaran	-

```
1.  for each local_skyline L that has bitmap B do
2.      if L is dominated by P then
3.          move L to shadow_skyline[B]
4.  for each virtual_point V that has bitmap B do
5.      if V is dominated by P and have complete
        dimensions that are superset of, or same as P's
6.          remove V
```

Gambar 4.5 Pseudocode fungsi insersi virtual point

4.3.5. Fungsi Untuk Mendapatkan *DDR Prime Ct*

Implementasi fungsi ini digunakan untuk mendapatkan *dynamic anti-dominance region* dari *ct*. Fungsi menerima sebuah input preferensi pengguna *ct* yang akan digunakan sebagai acuan untuk menentukan nilai maksimum dan minimum dari tiap dimensi. Rentang antara nilaik maksimum dan minimum pada tiap dimensi tersebut akan menjadi wilayah yang bisa digunakan sebagai batas perubahan *ct* yang dianjurkan. Jika *ct* diletakkan

dalam wilayah ini, maka *query point q* akan menajdi *dynamic skyline* dari *ct*. Algoritma ini dapat dilihat pada

Fungsi generate_ddr_prime_ct(*Point Ct*)	
Variabel Global	*product_list, node, local_skyline, candidate_skyline, global_skyline, shadow_skyline, virtual_point, bitmap, safe_region*
Masukan	*Point Ct*
Keluaran	-

1. find all *global_skyline* of *ct*
2. compare *query_point q* to all *global_skyline* of *ct*
3. **if** *q* is not dominated **then**
4. stop program //*q* is already in *RSL*(*ct*)
5. **else**
6. define top and bottom corner of each dimension

Gambar 4.6 Pseudocode fungsi mendapatkan DDR prime ct

4.3.6. Fungsi Untuk Mendapatkan *Safe Region Q*

Implementasi fungsi ini ditujukan untuk mencari irisan dari seluruh $\overline{DDR}$ dari tiap *RSL*. Setiap $\overline{DDR}$ merupakan wilayah aman perpindahan dari tiap preferensi pengguna, sehingga perpindahan *query point q* ke sembarang tempat didalam wilayah *SR*(*q*) tidak akan menyebabkan hilangnya *RSL*.

Fungsi generate_safe_region()	
Variabel Global	*customer_skyline, query_point, safe_region*
Masukan	-
Keluaran	-

1. **for all** *customer_skyline*
2. *SR*(*q*) ← *null*
3. **for each** *cl* ∈ *RSL*(*q*) **do**
4. **for each** comparable data in $\overline{DDR}$ and transformed value of *q* **do**
5. *bottom* = max($\overline{DDR}$, transformed *query_point*)

6.	*top* = min($\overline{DDR}$, transformed *query_point*)
7.	*safe_region = safe_region U [bottom, top]*
8.	

Gambar 4.7 Pseudocode fungsi mendapatkan safe region q

4.4. Implementasi Algoritma Pembanding (*Brute Force*)

Algoritma *brute force* diimplementasikan sesuai dengan rancangan yang telah dijelaskan pada bab perancangan. Algoritma *Brute Force* memiliki tujuan yang sama dengan algoritma program utama, namun dengan metode yang lebih sederhana. Program akan melakukan iterasi kesemua perubahan yang mungkin terjadi pada *ct*, kemudian melakukan apakah perubahan tersebut menyebabkan produk *q* muncul sebagai *skyline*. Perubahan yang bisa dilakukan pada *ct* adalah nilai rentang nilai antara *ct* itu sendiri hingga *q* pada setiap dimensinya. Pseudocode program *brute force* dapat dilihat pada Gambar 4.8.

Algoritma *Brute Force*	
Masukan	*q, ct*
Keluaran	*ct′*

1. *cheapest_cost* ← ∞
2. *cheapest_index* ← None
3. *candidate_combination* ← all possible combination of ct movement that makes *q* as *DSL*
4. **for each** *cc* ∈ *candidate_combination* **do**
5. **if** cost of *candidate_combination* < *cheapest_cost* **then**
6. cheapest_index ← cc
7. *best_cand* ← *best_cand*

Gambar 4.8 Pseudocode algoritma *brute force*

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan hasil uji coba yang dilakukan pada sistem yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi data uji coba dan skenario uji coba yang meliputi uji performa serta analisa setiap pengujian.

5.1. Data Uji Coba

Tahap uji coba pada penelitian ini menggunakan 3 jenis data, yaitu data *forest covertype* (FC), data independen (IND), dan data *anticorrelated* (ANT). Masing-masing jenis data memiliki variasi pada jumlah data (*n*), jumlah dimensi (*d*) dan jumlah konsumen yang terdaftar (*u*).

5.1.1. Data *Forest Covertype* (FC)

Himpunan data *forest covertype* (FC) berisi data hasil prediksi tipe tutupan hutan hanya dari variabel kartografi (tidak ada data penginderaan jauh). Jenis tutupan hutan yang sebenarnya untuk pengamatan yang diberikan (30 x 30 meter sel) ditentukan dari data *Resource Information System* (RIS) Wilayah 2 US *Forest Service* (USFS). Variabel independen berasal dari data yang awalnya diperoleh dari US *Geological Survey* (USGS) dan data USFS. Data dalam bentuk mentah (tidak diskalakan) dan berisi kolom data biner (0 atau 1) untuk variabel independen kualitatif (area padang gurun dan tipe tanah).

Data *forest covertype* (FC) adalah himpunan data yang diambil dari sumber sebenarnya. Penggunaan data ini bertujuan untuk menguji performa algoritma pada data dengan persebaran dan rentang nilai atribut sebenarnya yang bersifat saling berkaitan. Sebelum diolah dalam sistem, data FC akan dinormalkan terlebih dahulu karena rentang nilai antara satu dimensi dengan dimensi lainnya bisa sangat jauh berbeda.

5.1.2. Data Independen (IND)

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 algoritma jika berhadapan dengan data yang nilai atributnya tidak memiliki keterkaitan dengan apapun. Rentang nilai yang digunakan untuk setiap atribut adalah 1 – 100.

5.1.3. Data *Anticorrelated* (ANT)

Data *anticorrelated* (ANT) adalah himpunan data sintetis yang memiliki persebaran nilai atribut yang bagus disalah satu dimensinya, namun bisa saja jelek pada dimensi lainnya. Hal ini akan menyebabkan data *anticorrelated* memiliki kemungkinan untuk mempunyai jumlah *skyline* yang sangat banyak. Penggunaan data ini bertujuan untuk menguji performa algoritma jika berhadapan dengan data yang nilai atributnya saling bertolak belakang dan memiliki relasi dominasi antar data paling rendah dibandingkan dengan jenis data lainnya. Rentang nilai yang digunakan untuk setiap atribut adalah 1 – 100.

5.2. Skenario dan Evaluasi Pengujian Performa

Uji coba ini dilakukan untuk menguji apakah program yang telah diimplementasikan dapat berjalan dengan sebagaimana mestinya. Skenario uji coba performa berfokus pada pengujian seberapa baik program dapat memberikan rekomendasi perubahan nilai atas spesifikasi yang diberikan. Sebagai perbandingan, seluruh skenario ini akan dikerjakan dengan menggunakan algoritma berbasis *safe region* serta algoritma *brute force*. Hal yang akan menjadi penilaian dalam pengujian ini adalah penggunaan memori serta dan waktu eksekusi. Waktu yang digunakan untuk *preprocessing* tidak akan dihitung.

5.2.1. Uji Coba Jumlah Data

Pengujian ini dilakukan dengan memvariasikan jumlah data yang diproses. Jumlah dimensi dan jumlah konsumen akan tetap sama dalam pengujian ini. Skenario yang dilakukan dapat dilihat pada Tabel 5.1.

Tabel 5.1 Skenario perbandingan jumlah data

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)
A01	30.000	3	10
A02	50.000	3	10
A03	100.000	3	10
A04	200.000	3	10

5.2.2. Uji Coba Jumlah Dimensi

Pengujian ini dilakukan dengan memvariasikan jumlah dimensi yang diproses. Jumlah data dan jumlah konsumen akan tetap sama dalam pengujian ini. Skenario yang dilakukan dapat dilihat pada Tabel 5.2.

Tabel 5.2 Skenario perbandingan jumlah dimensi

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)
B01	30.000	2	10
B02	30.000	3	10
B03	30.000	4	10

5.2.3. Uji Coba Jumlah Konsumen

Pengujian ini dilakukan dengan memvariasikan jumlah data konsumen. Jumlah data dan jumlah dimensi akan tetap sama selama pengujian ini. Skenario yang dilakukan dapat dilihat pada Tabel 5.3.

Tabel 5.3 Skenario perbandingan jumlah konsumen

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)
C01	30.000	3	5
C02	30.000	3	10

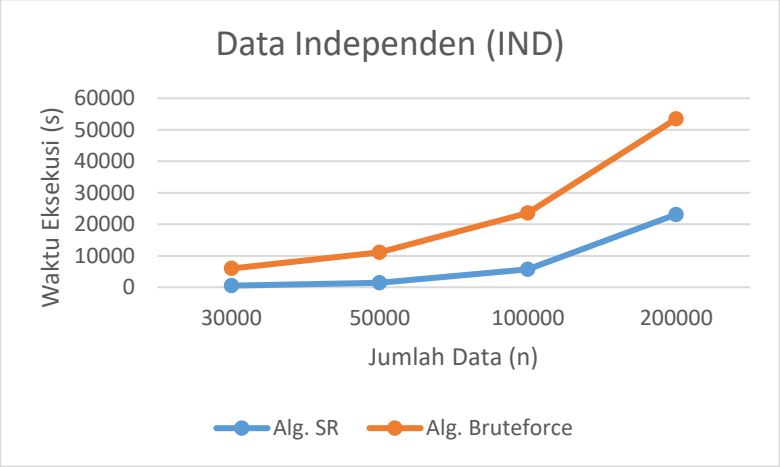
C03	30.000	3	50
C04	30.000	3	100
C05	30.000	3	200

5.3. Analisis Hasil Uji Coba

Bagian ini menjelaskan hasil uji coba yang telah dilakukan beserta analisanya. Uji coba yang dilakukan sesuai dengan skenario uji coba yang telah didefinisikan sebelumnya. Hasil uji coba dibagi menjadi tiga bagian, yaitu uji coba jumlah data, uji coba jumlah dimensi, dan uji coba jumlah konsumen.

5.3.1. Hasil Uji Coba Jumlah Data

Uji coba dilakukan sesuai dengan skenario pada Tabel 2.1. Algoritma berbasis *safe region* selalu lebih cepat dari pada algoritma *brute force*. Data *forest covertype* selalu memiliki jumlah *RSL* yang banyak sehingga proses penghitungan *safe region* menggunakan waktu yang lebih lama.

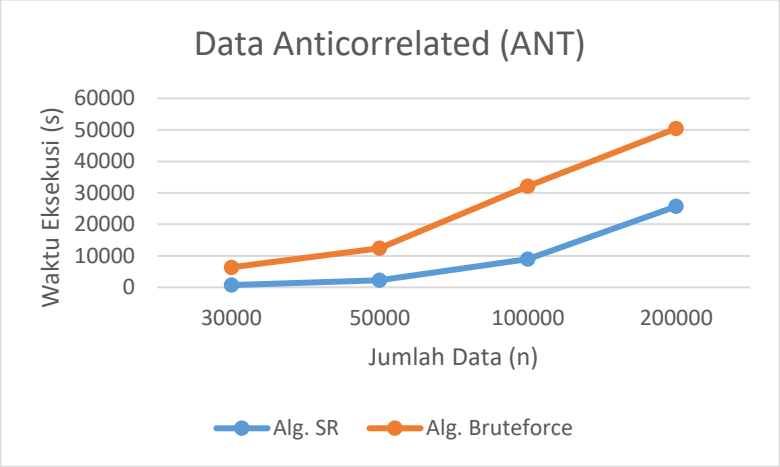


Gambar 5.1 Grafik perbandingan waktu eksekusi terhadap jumlah data pada data independen

Tabel 5.4 Hasil perbandingan waktu eksekusi terhadap jumlah data pada data independen

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)

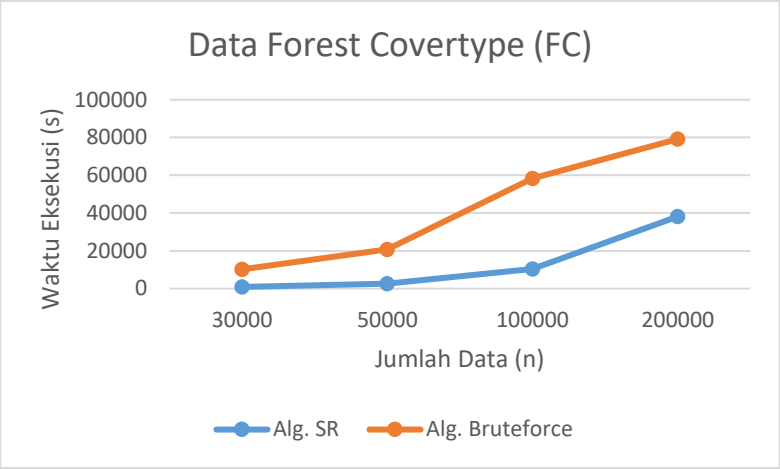
				SR	BF
A01	30.000	3	10	502	5946
A02	50.000	3	10	1445	11034
A03	100.000	3	10	5713	23582
A04	200.000	3	10	23109	53475



Gambar 5.2 Grafik perbandingan waktu eksekusi terhadap jumlah data pada data anticorrelated

Tabel 5.5 Hasil perbandingan waktu eksekusi terhadap jumlah data pada data anticorrelated

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)	
				SR	BF
A01	30.000	3	10	722	6371
A02	50.000	3	10	2278	12412
A03	100.000	3	10	8973	32132
A04	200.000	3	10	25707	50441



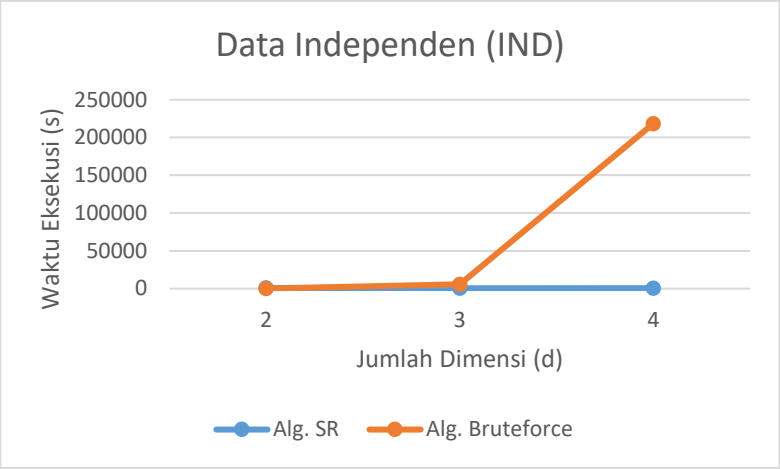
Gambar 5.3 Grafik perbandingan waktu eksekusi terhadap jumlah data pada data forest covertype

Tabel 5.6 Hasil perbandingan waktu eksekusi terhadap jumlah data pada data forest covertype

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)	
				SR	BF
A01	30.000	3	10	904	10226
A02	50.000	3	10	2661	20739
A03	100.000	3	10	10447	58345
A04	200.000	3	10	38135	79144

5.3.2. Hasil Uji Coba Jumlah Dimensi

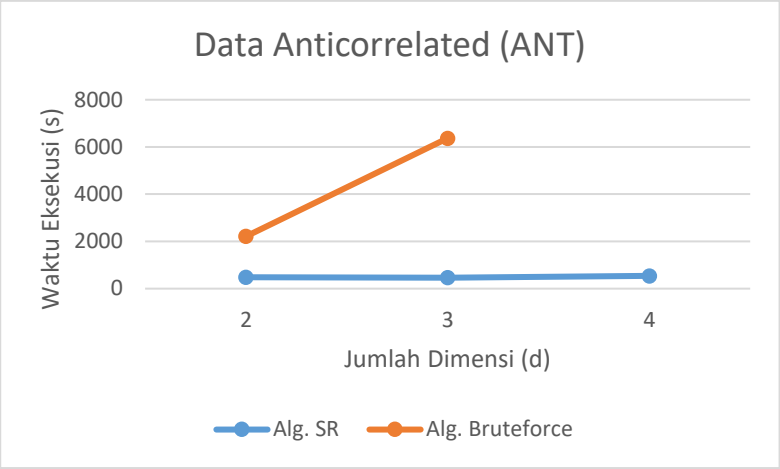
Uji coba dilakukan sesuai dengan skenario pada Tabel 5.2. Berdasarkan hasil yang diperoleh, dapat disimpulkan bahwa jumlah dimensi sangat mempengaruhi waktu eksekusi pada algoritma *brute force*, namun tidak terlalu berpengaruh pada algoritma berbasis *safe region*. Makin banyak jumlah dimensi, makin banyak kemungkinan yang harus diuji oleh algoritma *brute force*.



Gambar 5.4 Grafik perbandingan waktu eksekusi terhadap jumlah dimensi pada data independen

Tabel 5.7 Hasil perbandingan waktu eksekusi terhadap jumlah dimensi pada data independen

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)	
				SR	BF
B01	30.000	2	10	981	114
B02	30.000	3	10	502	5946
B03	30.000	4	10	559	218486

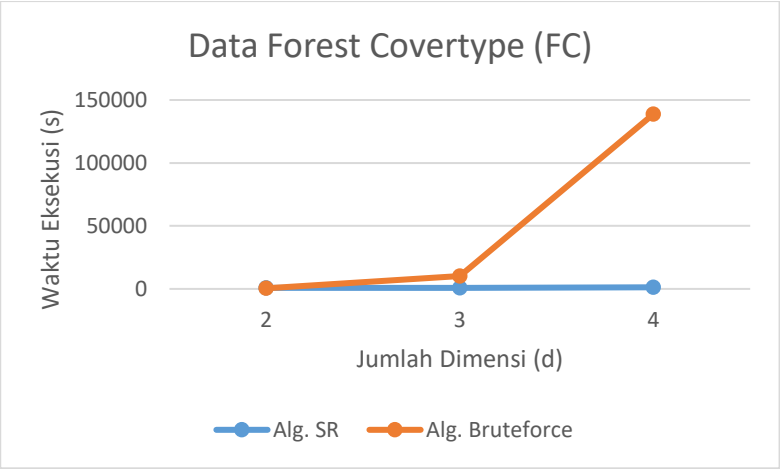


Gambar 5.5 Grafik perbandingan waktu eksekusi terhadap jumlah dimensi pada data anticorrelated

Untuk data anticorrelated dengan metode *brute force*, uji coba dengan jumlah dimensi 4 menghabiskan waktu yang sangat lama. Sehingga pengujian waktu eksekusi terhadap perubahan dimensi tidak lengkap untuk algoritma *brute force*.

Tabel 5.8 Hasil perbandingan waktu eksekusi terhadap jumlah dimensi pada data anticorrelated

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)	
				SR	BF
B01	30.000	2	10	483	2212
B02	30.000	3	10	463	6371
B03	30.000	4	10	541	



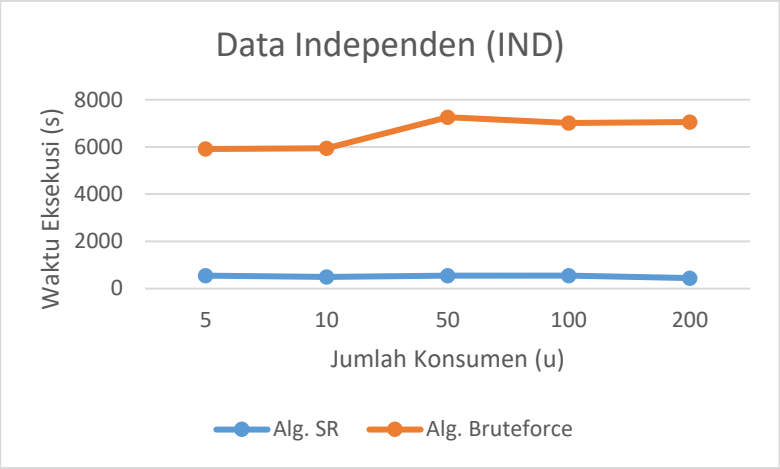
Gambar 5.6 Grafik perbandingan waktu eksekusi terhadap jumlah dimensi pada data forest covertype

Tabel 5.9 Hasil perbandingan waktu eksekusi terhadap jumlah dimensi pada data forest covertype

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)	
				SR	BF
B01	30.000	2	10	830	470
B02	30.000	3	10	904	10226
B03	30.000	4	10	1278	138788

5.3.3. Hasil Uji Coba Jumlah Konsumen

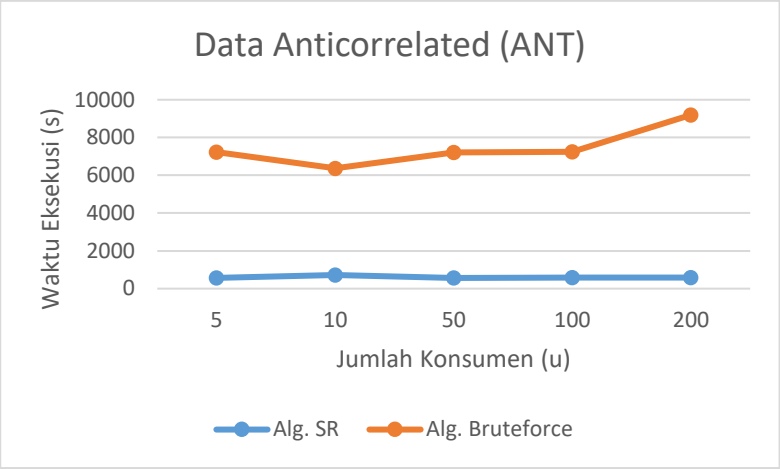
Uji coba dilakukan sesuai dengan skenario pada Tabel 5.3. Hasil uji coba menggambarkan bahwa jumlah konsumen tidak terlalu mempengaruhi waktu eksekusi. Hal ini terjadi karena sistem yang dibuat hanya menggunakan jumlah konsumen yang terdaftar pada proses *preprocessing*. Hasil uji coba dapat dilihat pada Tabel 5.10, Tabel 5.11 dan Tabel 5.12.



Gambar 5.7 Grafik perbandingan waktu eksekusi terhadap jumlah konsumen pada data independen

Tabel 5.10 Hasil perbandingan waktu eksekusi terhadap jumlah konsumen pada data independen

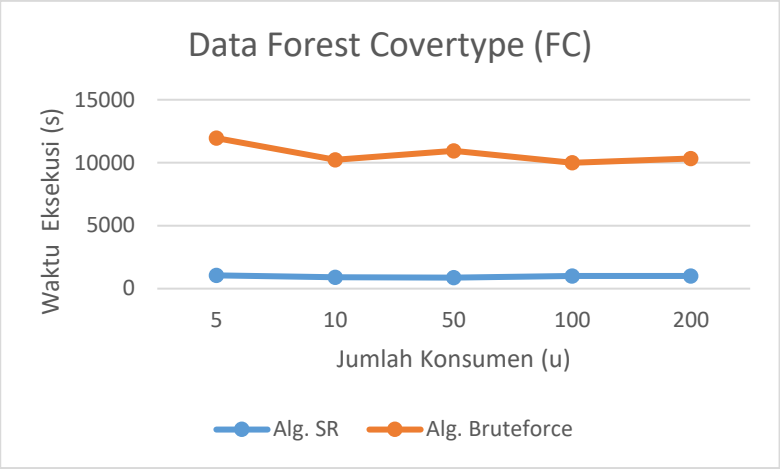
Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)	
				SR	BF
C01	30.000	3	5	549	5913
C02	30.000	3	10	502	5946
C03	30.000	3	50	547	7259
C04	30.000	3	100	551	7013
C05	30.000	3	200	447	7049



Gambar 5.8 Grafik perbandingan waktu eksekusi terhadap jumlah konsumen pada data anticorrelated

Tabel 5.11 Hasil perbandingan waktu eksekusi terhadap jumlah konsumen pada data anticorrelated

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)	
				SR	BF
C01	30.000	3	5	574	7235
C02	30.000	3	10	722	6371
C03	30.000	3	50	565	7212
C04	30.000	3	100	584	7236
C05	30.000	3	200	581	9195



Gambar 5.5.9 Grafik perbandingan waktu eksekusi terhadap jumlah konsumen pada data forest covertype

Tabel 5.12 Hasil perbandingan waktu eksekusi terhadap jumlah konsumen pada data forest covertype

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Jumlah Konsumen (*u*)	Waktu Eksekusi (*s*)	
				SR	BF
C01	30.000	3	5	1053	11950
C02	30.000	3	10	904	10226
C03	30.000	3	50	874	10944
C04	30.000	3	100	1014	9994
C05	30.000	3	200	1011	10338

5.3.4. Hasil Uji Coba Memori yang Terpakai

Berikut adalah tabel penggunaan memori dari hasil uji coba yang telah dilakukan.

Tabel 5.13 Penggunaan memori untuk jumlah data yang divariasikan

n	*d* = 3, *u* = 10					
	IND		ANT		FC	
	SR	BF	SR	BF	SR	BF
30K	401M	397M	406M	397M	406	397M
50K	404M	397M	410M	397M	410	397M
100K	418M	396M	419M	397M	418	397M
200K	436M	396M	437M	397M	436	396M

Tabel 5.14 Penggunaan memori untuk jumlah dimensi yang divariasikan

d	*n* = 30.000, *u* = 10					
	IND		ANT		FC	
	SR	BF	SR	BF	SR	BF
2	406M	396M	406M	396M	406M	396M
3	401M	397M	406M	397M	406M	397M
4	407M	407M	402M	428M	408M	401M

Tabel 5.15 Penggunaan memori untuk jumlah konsumen yang divariasikan

u	*n* = 30.000, *d* = 3					
	IND		ANT		FC	
	SR	BF	SR	BF	SR	BF
5	406M	396M	406M	397M	406M	397M
10	401M	397M	406M	397M	406M	397M
50	406M	396M	401M	397M	406M	397M
100	406M	396M	407M	397M	406M	397M
200	406M	396M	407M	397M	406M	397M

Dari tabel tersebut, dapat disimpulkan bahwa algoritma *brute force* cenderung menggunakan memori dengan jumlah yang sama. Sedangkan algoritma berbasis *safe region* akan bervariasi tergantung dimensi, jumlah data, dan jumlah konsumen. Semakin banyak jumlah data, jumlah dimensi, atau jumlah konsumen yang

digunakan, maka akan semakin banyak pula memori yang terpakai. Hal yang menyebabkan algoritma *brute force* cenderung menggunakan jumlah memori yang sama adalah karena algoritma ini akan melakukan pengecekan langsung untuk tiap perubahan kemungkinan yang bisa terjadi tanpa melibatkan semua data spesifikasi produk yang ada.

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1. Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap pembuatan algoritma, dapat diambil kesimpulan sebagai berikut:

1. Algoritma yang tepat untuk menjawab masalah *why-not questions on reverse skyline queries* pada data yang tidak lengkap adalah dengan mengkombinasikan algoritma *ISkyline* untuk menentukan *skyline* dan memberikan rekomendasi berdasarkan *safe region* dari *query point*.
2. Struktur data yang paling efisien adalah dengan membagi data kedalam *bucket-bucket* tertentu sehinga proses pencarian *skyline* bisa dilakukan denga lebih cepat karena tidak perlu membandingkan dengan semua data yang ada.

6.2. Saran

Saran yang diberikan untuk pengembangan penelitian ini adalah:

1. Penelitian dapat dikembangkan untuk melakukan komputasi perubahan nilai terhadap banyak konsumen sekaligus.
2. Penelitian dapat dikembangkan dengan kondisi himpunan data yang dinamis ataupun tidak pasti.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] M. S. Islam, R. Zhou and C. Liu, "On answering why-not questions in reverse skyline queries," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)* , Brisbane, 2013.

[2] M. E. Khalefa, M. F. Mokbel and J. J. Levandoski, "Skyline Query Processing for Incomplete Data," in *2008 IEEE 24th International Conference on Data Engineering* , Cancun, 2008.

[3] R. Bharuka and P. S. Kumar, "Finding skylines for incomplete data," *ADC '13 Proceedings of the Twenty-Fourth Australasian Database Conference,* vol. 137, no. Australian Computer Society, pp. 109-117, 2013.

[4] J. A. Blackard, D. J. Jean and C. W. Anderson, "UCI Machine Learning Repositories," 1 Agustus 1998. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/covertype. [Accessed 12 Februari 2018].

[Halaman ini sengaja dikosongkan]

LAMPIRAN 1

Kode sumber untuk program *preprocessing*.

```
#!/usr/bin/python

import sys
import numpy as np

start_time = time.time()

marker = 1
node = {}
local_skyline = {}
candidate_skyline = []
global_skyline = []
shadow_skyline = {}
virtual_point = {}
why_not_points = {}
n_updated_flag = {}
data_length = 0
t = 5
rsl = []
safe_region = []
number_of_preference = 0
customer_skyline = {}
customer_index = 0
query_point = []
list_customer = []
ct = []
ct_has_skyline = True
product_list = "TESTING_FC_D3_N100.txt"
user_preference = "TESTING_USER_D3_N10.txt"
intersection = []
ct_cost = []
q_cost = []
jumlah_rsl = 0
list_rsl = []


def insert_local_skyline(current_specs, bitmap):
```

```
38.     global local_skyline
39.     global shadow_skyline
40.     global virtual_point
41.
42.     for i in range(0, len(local_skyline[bitmap])):

43.         dominating_local = False
44.         dominated_by_local = False
45.         for j in range(1, len(current_specs)):
46.             if(current_specs[j] != 'null' and local
    _skyline[bitmap][i][j] != 'null'):
47.                 if(current_specs[j] < local_skyline
    [bitmap][i][j]):
48.                     dominating_local = True
49.                 elif(current_specs[j] > local_skyli
    ne[bitmap][i][j]):
50.                     dominated_by_local = True
51.         if(dominating_local == True and dominated_b
    y_local == False):
52.             local_skyline[bitmap][i][-
    1] = 'delete'
53.         elif(dominating_local == False and dominate
    d_by_local == True):
54.             for k in range(0, i+1):
55.                 local_skyline[bitmap][k][-
    1] = 'ok'
56.             return False
57.     dominated = 0
58.     for i in range(0, len(virtual_point[bitmap])):

59.         dominating_virtual = False
60.         dominated_by_virtual = False
61.         for j in range(1, len(current_specs)):
62.             if(current_specs[j] != 'null' and virtu
    al_point[bitmap][i][j] != 'null'):
63.                 if(current_specs[j] < virtual_point
    [bitmap][i][j]):
64.                     dominating_virtual = True
65.                 elif(current_specs[j] > virtual_poi
    nt[bitmap][i][j]):
66.                     dominated_by_virtual = True
67.         if(dominating_virtual == False and dominate
    d_by_virtual == True):
```

```
68.             dominated = 1
69.             break
70.     if(dominated == 0):
71.         content = list(current_specs)
72.         content.append('ok')
73.         local_skyline[bitmap].append(content)
74.         for i in sorted(local_skyline[bitmap], reve
    rse=True):
75.             if (i[-1] == 'delete'):
76.                 local_skyline[bitmap].remove(i)
77.         for i in range(0, len(shadow_skyline[bitmap
    ])):
78.             dominating_shadow = False
79.             dominated_by_shadow = False
80.             for j in range(1, len(current_specs)):

81.                 if(current_specs[j] != 'null' and s
    hadow_skyline[bitmap][i][j] != 'null'):
82.                     if(current_specs[j] < shadow_sk
    yline[bitmap][i][j]):
83.                         dominating_shadow = True
84.                     elif(current_specs[j] > shadow_
    skyline[bitmap][i][j]):
85.                         dominated_by_shadow = True

86.             if(dominating_shadow == True and domina
    ted_by_shadow == False):
87.                 shadow_skyline[bitmap][i][-
    1] = 'delete'
88.         for i in sorted(shadow_skyline[bitmap], rev
    erse=True):
89.             if (i[-1] == 'delete'):
90.                 shadow_skyline[bitmap].remove(i)
91.         return True
92.     elif(dominated == 1):
93.         n_updated_flag[bitmap] = True
94.         for i in range(0, len(shadow_skyline[bitmap
    ])):
95.             dominating_shadow = False
96.             dominated_by_shadow = False
97.             for j in range(1, len(current_specs)):
```

```
98.                 if(current_specs[j] != 'null' and s
    hadow_skyline[bitmap][i][j] != 'null'):
99.                     if(current_specs[j] < shadow_sk
    yline[bitmap][i][j]):
100.                                dominating_shadow =
    True
101.                            elif(current_specs[j] >
    shadow_skyline[bitmap][i][j]):
102.                                dominated_by_shadow
    = True
103.                    if(dominating_shadow == True and
     dominated_by_shadow == False):
104.                        shadow_skyline[bitmap][i][-
    1] = 'delete'
105.                for i in sorted(shadow_skyline[bitma
    p], reverse=True):
106.                    if (i[-1] == 'delete'):
107.                        shadow_skyline[bitmap].remov
    e(i)
108.                content = list(current_specs)
109.                content.append('ok');
110.                shadow_skyline[bitmap].append(conten
    t)
111.            return False
112.
113.
114.        def insert_candidate_skyline(current_specs,
    bitmap):
115.            global candidate_skyline
116.            list_bit_inserted = []
117.            dominated = 0
118.
119.            for i in range(0, len(candidate_skyline)
    ):
120.                dominating_candidate = False
121.                dominated_by_candidate = False
122.                for j in range(1, len(current_specs)
    ):
123.                    if(current_specs[j] != 'null' an
    d candidate_skyline[i][j] != 'null'):
124.                        if(current_specs[j] < candid
    ate_skyline[i][j]):
```

```
125.                             dominating_candidate = T
    rue
126.                         elif(current_specs[j] > cand
    idate_skyline[i][j]):
127.                             dominated_by_candidate =
     True
128.                 if(dominating_candidate == True and
    dominated_by_candidate == False):
129.                     candidate_skyline[i][-
    1] = 'delete'
130.                     if(candidate_skyline[i][-
    2] not in list_bit_inserted):
131.                         insert_virtual_point(current
    _specs, candidate_skyline[i][-2])
132.                         list_bit_inserted.append(can
    didate_skyline[i][-2])
133.                 elif(dominating_candidate == False a
    nd dominated_by_candidate == True):
134.                     content = list(candidate_skyline
    [i][:-2])
135.                     insert_virtual_point(content, bi
    tmap)
136.                     dominated = 1
137.             candidate_skyline = [i for i in candidat
    e_skyline if i[-1] == 'ok']
138.             if(dominated == 0):
139.                 content = list(current_specs)
140.                 content.append(bitmap)
141.                 content.append('ok')
142.                 candidate_skyline.append(content)
143.
144.
145.
146.         def insert_virtual_point(current_specs, bitm
    ap):
147.             global local_skyline
148.             global virtual_point
149.             global shadow_skyline
150.             #MEMBANDINGKAN DENGAN LOCAL SKYLINE
151.             #Move all dominated local_skyline N to s
    hadow_skyline
152.             for i in range(0, len(local_skyline[bitm
    ap])):
```

```
153.                 dominating_local = False
154.                 dominated_by_local = False
155.                 for j in range(1, len(current_specs)
    ):
156.                     if(current_specs[j] != 'null' an
    d local_skyline[bitmap][i][j] != 'null'):
157.                         if(current_specs[j] < local_
    skyline[bitmap][i][j]):
158.                             dominating_local = True

159.                         elif(current_specs[j] > loca
    l_skyline[bitmap][i][j]):
160.                             dominated_by_local = Tru
    e
161.                 if(dominating_local == True and domi
    nated_by_local == False):
162.                     local_skyline[bitmap][i][-
    1] = 'delete'
163.             for i in reversed(local_skyline[bitmap])
    :
164.                 if (i[-1] == 'delete'):
165.                     shadow_skyline[bitmap].append(i)

166.                     local_skyline[bitmap].remove(i)

167.                     shadow_skyline[bitmap][-1][-
    1] = 'ok'
168.
169.             for i in range(0, len(virtual_point[bitm
    ap])):
170.                 dominating_virtual = False
171.                 dominated_by_virtual = False
172.                 superset_check = 0
173.
174.                 for j in range(1, len(current_specs)
    ):
175.                     if(current_specs[j] != 'null' an
    d virtual_point[bitmap][i][j] != 'null'):
176.                         if(current_specs[j] < virtua
    l_point[bitmap][i][j]):
177.                             dominating_virtual = Tru
    e
```

```
178.                         elif(current_specs[j] > virt
    ual_point[bitmap][i][j]):
179.                             dominated_by_virtual = T
    rue
180.                     if(current_specs[j] != 'null'):

181.                         superset_check += 1
182.                     elif(current_specs[j] == 'null'
    and virtual_point[bitmap][i][j] == 'null'):
183.                         superset_check += 1
184.                 if(dominating_virtual == True and do
    minated_by_virtual == False and superset_check == l
    en(current_specs)):
185.                     virtual_point[bitmap][i][-
    1] = 'delete'
186.
187.             for i in reversed(virtual_point[bitmap])
    :
188.                 if(i[-1] == 'delete'):
189.                     virtual_point[bitmap].remove(i)

190.             content = list(current_specs)
191.             content.append('ok')
192.             virtual_point[bitmap].append(content)
193.
194.
195.         def update_global_skyline():
196.             global global_skyline
197.             global candidate_skyline
198.             global shadow_skyline
199.             global data_length
200.             for c in range(0, len(candidate_skyline)
    ):
201.                 for g in range(0, len(global_skyline
    )):
202.                     dominating_global = False
203.                     dominating_candidate = False
204.                     for i in range(1, data_length):

205.                         if(candidate_skyline[c][i] !
    = 'null' and global_skyline[g][i] != 'null'):
206.                             if(candidate_skyline[c][
    i] < global_skyline[g][i]):
```

```
207.                                 dominating_global =
    True
208.                             elif(candidate_skyline[c
    ][i] > global_skyline[g][i]):
209.                                 dominating_candidate
     = True
210.                     if(dominating_global == True and
     dominating_candidate == False):
211.                         global_skyline[g][-
    1] = 'delete'
212.                     elif(dominating_global == False
    and dominating_candidate == True):
213.                         candidate_skyline[c][-
    1] = 'delete'
214.
215.             for i in reversed(global_skyline):
216.                 if(i[-1] == 'delete'):
217.                     global_skyline.remove(i)
218.             for i in reversed(candidate_skyline):
219.                 if(i[-1] == 'delete'):
220.                     candidate_skyline.remove(i)
221.             for g in range(0, len(global_skyline)):

222.                 for i in n_updated_flag:
223.                     if (n_updated_flag[i] == True an
    d i in shadow_skyline):
224.                         for s in range(0, len(shadow
    _skyline[i])):
225.                             dominating_global = Fals
    e
226.                             dominating_shadow = Fals
    e
227.                             for j in range(1, data_l
    ength):
228.                                 if(global_skyline[g]
    [j] != 'null' and shadow_skyline[i][s][j] != 'null'
     ):
229.                                     if(global_skylin
    e[g][j] < shadow_skyline[i][s][j]):
230.                                         dominating_s
    hadow = True
231.                                     elif(global_skyl
    ine[g][j] > shadow_skyline[i][s][j]):
```

```
232.                                         dominating_g
    lobal = True
233.                             if(dominating_global ==
    True and dominating_shadow == False):
234.                                 global_skyline[g][-
    1] == 'delete'
235.
236.             for c in range(0, len(candidate_skyline)
    ):
237.                 for i in node:
238.                     for s in range(0, len(shadow_sky
    line[i])):
239.                         dominating_candidate = False

240.                         dominating_shadow = False
241.                         for j in range(1, data_lengt
    h):
242.                             if(candidate_skyline[c][
    j] != 'null' and shadow_skyline[i][s][j] != 'null')
    :
243.                                 if(candidate_skyline
    [c][j] < shadow_skyline[i][s][j]):
244.                                     dominating_shado
    w = True
245.                                 elif(candidate_skyli
    ne[c][j] > shadow_skyline[i][s][j]):
246.                                     dominating_candi
    date = True
247.                         if(dominating_candidate == T
    rue and dominating_shadow == False):
248.                             candidate_skyline[c][-
    1] = 'delete'
249.
250.             for i in reversed(global_skyline):
251.                 if(i[-1] == 'delete'):
252.                     global_skyline.remove(i)
253.             for i in reversed(candidate_skyline):
254.                 if(i[-1] == 'delete'):
255.                     candidate_skyline.remove(i)
256.             for i in candidate_skyline:
257.                 global_skyline.append(i)
258.
259.             for i in n_updated_flag:
```

```
260.                 n_updated_flag[i] == False
261.
262.
263.
264.         def calculate_rsl_q(customer_skyline, query_
    point):
265.             global data_length
266.             for dict_index in customer_skyline:
267.                 transformed_query_point = []
268.                 for q in range(0, data_length):
269.                     transformed_value = abs(float(qu
    ery_point[q+1]) - customer_skyline[dict_index][-
    2][q])
270.                     transformed_query_point.append(t
    ransformed_value)
271.                 for data_index in range(0, len(custo
    mer_skyline[dict_index]) - 2):
272.                     dominating_q = False
273.                     dominating_customer = False
274.                     for i in range(0, data_length):

275.                         if(customer_skyline[dict_ind
    ex][data_index][i+1] != 'null'):
276.                             if(customer_skyline[dict
    _index][data_index][i+1] < transformed_query_point[
    i]):
277.                                 dominating_q = True

278.                             elif(customer_skyline[di
    ct_index][data_index][i+1] > transformed_query_poin
    t[i]):
279.                                 dominating_customer
    = True
280.                     if(dominating_q == True and domi
    nating_customer == False):
281.                         customer_skyline[dict_index]
    [-1] = 'delete'
282.                     elif(dominating_q == False and d
    ominating_customer == True):
283.                         customer_skyline[dict_index]
    [data_index][-1] = 'delete'
284.                 if(customer_skyline[dict_index][-
    1] == 'ok'):
```

```
285.                     for i in range(len(customer_skyl
    ine[dict_index]) - 3, -1, -1):
286.                         if(customer_skyline[dict_ind
    ex][i][-1] == 'delete'):
287.                             customer_skyline[dict_in
    dex].remove(customer_skyline[dict_index][i])
288.                 if(len(customer_skyline[dict_index])
     <= 2):
289.                     customer_skyline[dict_index][-
    1] = 'delete'
290.             return customer_skyline
291.
292.
293.         def Prepare_Data(line, customer):
294.             global bitmap
295.             global node
296.             global local_skyline
297.             global candidate_skyline
298.             global global_skyline
299.             global shadow_skyline
300.             global virtual_point
301.             global n_updated_flag
302.             global data_length
303.
304.             current_spec = line.split()
305.             data = []
306.             transformed_data = []
307.             transformed_data.append(current_spec[0])

308.             data.append(current_spec[0])
309.             for i in range(1, data_length+1):
310.                 if(current_spec[i] == "null"):
311.                     bitmap += "0"
312.                     data.append(current_spec[i])
313.                     transformed_data.append(current_
    spec[i])
314.                 else:
315.                     bitmap += "1"
316.                     difference = abs(float(current_s
    pec[i]) - customer[i-1])
317.                     data.append(float(current_spec[i
    ]))
```

```
318.                     transformed_data.append(float(di
    fference))
319.             if bitmap not in node:
320.                 node[bitmap] = []
321.                 local_skyline[bitmap] = []
322.                 shadow_skyline[bitmap] = []
323.                 virtual_point[bitmap] = []
324.                 n_updated_flag[bitmap] = False
325.             return transformed_data
326.
327.
328.         data_length = len(ct)
329.         fu = open(user_preference)
330.         for list_user in fu:
331.             temp = [float(x) for x in list_user.spli
    t()]
332.             list_customer.append(temp)
333.
334.         for x in range(0, len(list_customer)):
335.             fp = open(product_list)
336.             node.clear()
337.             local_skyline.clear()
338.             candidate_skyline.clear()
339.             global_skyline.clear()
340.             shadow_skyline.clear()
341.             virtual_point.clear()
342.             number_of_preference += 1
343.             for line in fp:
344.                 bitmap = ""
345.                 transformed_data = Prepare_Data(line
    , list_customer[x])
346.                 is_skyline = insert_local_skyline(tr
    ansformed_data, bitmap)
347.                 if is_skyline == True:
348.                     insert_candidate_skyline(transfo
    rmed_data, bitmap)
349.                     if(len(candidate_skyline) > t):

350.                         update_global_skyline()
351.                         candidate_skyline.clear()
352.             fp.close()
353.             update_global_skyline()
354.             if(len(global_skyline) > 0):
```

```
                customer_skyline[str(customer_index)] = list(global_skyline)
                customer_skyline[str(customer_index)].append(list(list_customer[x]))
                customer_skyline[str(customer_index)].append("ok")
                customer_index += 1
        fu.close()
        ddr_prime_ct = generate_ddr_prime_ct(ct)
        safe_region = generate_safe_region_q()


        if(ct_has_skyline == True):
            intersection_status = check_intersection(safe_region, ddr_prime_ct)
        else:
            intersection_status = False
        if(intersection_status == True):
            move_query_point()
        else:
            Move_Why_Not_And_Query_Point()
```

[Halaman ini sengaja dikosongkan]

LAMPIRAN 2

Kode sumber algoritma program utama.

```
1.  #!/usr/bin/python
2.
3.  def calculate_rsl_q(customer_skyline, query_point):

4.      global data_length
5.      for dict_index in customer_skyline:
6.          transformed_query_point = []
7.          for q in range(0, data_length):
8.              transformed_value = abs(float(query_poi
    nt[q+1]) - customer_skyline[dict_index][-2][q])
9.              transformed_query_point.append(transfor
    med_value)
10.         for data_index in range(0, len(customer_sky
    line[dict_index]) - 2):
11.             dominating_q = False
12.             dominating_customer = False
13.             for i in range(0, data_length):
14.                 if(customer_skyline[dict_index][dat
    a_index][i+1] != 'null'):
15.                     if(customer_skyline[dict_index]
    [data_index][i+1] < transformed_query_point[i]):
16.                         dominating_q = True
17.                     elif(customer_skyline[dict_inde
    x][data_index][i+1] > transformed_query_point[i]):

18.                         dominating_customer = True

19.             if(dominating_q == True and dominating_
    customer == False):
20.                 customer_skyline[dict_index][-
    1] = 'delete'
21.             elif(dominating_q == False and dominati
    ng_customer == True):
22.                 customer_skyline[dict_index][data_i
    ndex][-1] = 'delete'
```

```
23.         if(customer_skyline[dict_index][-
    1] == 'ok'):
24.             for i in range(len(customer_skyline[dic
    t_index]) - 3, -1, -1):
25.                 if(customer_skyline[dict_index][i][
    -1] == 'delete'):
26.                     customer_skyline[dict_index].re
    move(customer_skyline[dict_index][i])
27.         if(len(customer_skyline[dict_index]) <= 2):

28.             customer_skyline[dict_index][-
    1] = 'delete'
29.     return customer_skyline
30.
31. def generate_safe_region_q():
32.     global customer_skyline
33.     global query_point
34.     global safe_region
35.     global data_length
36.     global jumlah_rsl
37.     global ct
38.     global start_time
39.
40.     query_point = query_point.split()
41.     calculate_rsl_q(customer_skyline, query_point)

42.
43.     q = []
44.     for i in range(0, data_length):
45.         q.append(float(query_point[i+1]))
46.
47.     safe_region = []
48.     for dict_index in customer_skyline:     #c is d
    ictionary index
49.         if(customer_skyline[dict_index][-
    1] == 'ok' and len(customer_skyline)>2):
50.             jumlah_rsl += 1
51.             list_rsl.append(dict_index)
52.             ddr_prime = []
53.             for data_index in range(0, len(customer
    _skyline[dict_index])-2):
54.                 data = []
55.                 for i in range(0, data_length):
```

```
56.                     if(customer_skyline[dict_index]
    [data_index][i+1] != 'null'):
57.                         top = customer_skyline[dict
    _index][-
    2][i] + customer_skyline[dict_index][data_index][i+
    1]
58.                         bottom = customer_skyline[d
    ict_index][-
    2][i] - customer_skyline[dict_index][data_index][i+
    1]
59.                     else:
60.                         top = 'null'
61.                         bottom = 'null'
62.                     max_min_value = [top, bottom]
63.                     data.append(max_min_value)
64.                 ddr_prime.append(data)
65.             data = []
66.             for i in range(0, data_length):
67.                 diff = abs(float(q[i]) - customer_s
    kyline[dict_index][-2][i])
68.                 top = customer_skyline[dict_index][
    -2][i] + diff
69.                 bottom = customer_skyline[dict_inde
    x][-2][i] - diff
70.                 max_min_value = [top, bottom]
71.                 data.append(max_min_value)
72.             ddr_prime.append(data)
73.             used_ddr_prime = []
74.             for data_index in range(0, len(ddr_prim
    e)):
75.                 q_dimension_counter = 0
76.                 for i in range(0, data_length):
77.                     if(ddr_prime[data_index][i][0]
    != 'null'):
78.                         if(ddr_prime[data_index][i]
    [0] >= float(q[i]) and ddr_prime[data_index][i][1]
    <= float(q[i])):
79.                             q_dimension_counter +=
    1
80.                     else:
81.                         q_dimension_counter += 1
82.
83.
```

```
84.                 if(q_dimension_counter == data_leng
    th):
85.                     used_ddr_prime.append(ddr_prime
    [data_index])
86.             ddr_prime = list(used_ddr_prime)
87.
88.             if(len(safe_region) == 0):
89.                 safe_region = list(ddr_prime)
90.             else:
91.                 new_safe_region = []
92.                 for safe_index in range(0, len(safe
    _region)):
93.                     for ddr_index in range(0, len(d
    dr_prime)):
94.                         intersect_status = True
95.                         intersect_data = []
96.                         for i in range(0, data_leng
    th):
97.                             #top
98.                             if(ddr_prime[ddr_index]
    [i][0] == 'null' and safe_region[safe_index][i][0]
    == 'null'):
99.                                 top = 'null'
100.                            elif(ddr_prime[d
    dr_index][i][0] == 'null'):
101.                                top = safe_r
    egion[safe_index][i][0]
102.                            elif(safe_region
    [safe_index][i][0] == 'null'):
103.                                top = ddr_pr
    ime[ddr_index][i][0]
104.                            else:
105.                                top = min(dd
    r_prime[ddr_index][i][0], safe_region[safe_index][i
    ][0])
106.
107.                            #bottom
108.                            if(ddr_prime[ddr
    _index][i][1] == 'null' and safe_region[safe_index]
    [i][1] == 'null'):
109.                                bottom = 'nu
    ll'
```

```
110.                                    elif(ddr_prime[d
    dr_index][i][1] == 'null'):
111.                                        bottom = saf
    e_region[safe_index][i][1]
112.                                    elif(safe_region
    [safe_index][i][1] == 'null'):
113.                                        bottom = ddr
    _prime[ddr_index][i][1]
114.                                    else:
115.                                        bottom = max
    (ddr_prime[ddr_index][i][1], safe_region[safe_index
    ][i][1])
116.                                    if(top != 'null'
     and bottom != 'null'):
117.                                        if(bottom >
    top):
118.                                            intersec
    t_status = False
119.                                    max_min_value =
    [top, bottom]
120.                                    intersect_data.a
    ppend(max_min_value)
121.                                if(intersect_status
    == True):
122.                                    new_safe_region.
    append(intersect_data)
123.                        if(len(new_safe_region) > 0)
    :
124.                            safe_region = list(new_s
    afe_region)
125.            if(len(safe_region) == 0):
126.                generate_cost()
127.                T = move_why_not_point(ct, q)
128.
129.                print("")
130.                print("   RESULT : MOVING WHY-
    NOT POINT")
131.                print("   Q      : " + str(T["q"]))

132.                print("   CT     : " + str(T["ct"]))

133.
134.                exit()
```

```
135.             return safe_region
136.
137.
138.
139.
140.         def generate_ddr_prime_ct(ct):
141.             global product_list
142.             global node
143.             global local_skyline
144.             global candidate_skyline
145.             global global_skyline
146.             global shadow_skyline
147.             global virtual_point
148.             global bitmap
149.             global query_point
150.             global t
151.             global data_length
152.             global ct_has_skyline
153.
154.             ct_has_skyline = True
155.
156.             fp = open(product_list)
157.             local_skyline.clear()
158.             candidate_skyline.clear()
159.             global_skyline.clear()
160.             shadow_skyline.clear()
161.             virtual_point.clear()
162.             node.clear()
163.             for line in fp:
164.                 bitmap = ""
165.                 transformed_data = Prepare_Data(line
    , ct)
166.                 is_skyline = insert_local_skyline(tr
    ansformed_data, bitmap)
167.                 if(is_skyline == True):
168.                     insert_candidate_skyline(transfo
    rmed_data, bitmap)
169.                     if(len(candidate_skyline) > t):

170.                         update_global_skyline()
171.                         candidate_skyline.clear()
172.             update_global_skyline()
173.             candidate_skyline.clear()
```

```
            fp.close()

            q = query_point.split()
            for i in range(0, data_length):
                q[i+1] = abs(float(q[i+1]) - ct[i])

            q.append("qp")
            q.append("ok")

            q_is_skyline = True
            for data_index in range(0, len(global_sk
    yline)):
                smaller = False
                greater = False
                for i in range(0, data_length):
                    if(q[i+1] < ct[i]):
                        smaller = True
                    elif(q[i+1] > ct[i]):
                        greater = True
                if(smaller == True and greater == Fa
    lse):
                    pass#tetap True
                elif(smaller == False and greater ==
     True):
                    q_is_skyline = False

            if(len(global_skyline) == 0):
                elapsed_time = time.time() - start_t
    ime
                ct_has_skyline = False
            if(q_is_skyline == True):
                #HENTIKAN PROGRAM
                print("   TIDAK PERLU DILAKUKAN PENY
    ESUAIAN")
                exit()
            else:
                ddr_prime_ct = []
                for data_index in range(0, len(globa
    l_skyline)):
                    data = []
                    for i in range(0, data_length):
```

```
                        if(global_skyline[data_index
    ][i+1] != 'null'):
                            top = ct[i] + global_sky
    line[data_index][i+1]
                            bottom = ct[i] - global_
    skyline[data_index][i+1]
                        else:
                            top = 'null'
                            bottom = 'null'
                        max_min_value = [top, bottom
    ]
                        data.append(max_min_value)
                    ddr_prime_ct.append(data)
            return ddr_prime_ct


        def check_intersection(safe_region, ddr_prim
    e_ct):
            global intersection
            global data_length

            intersection = []
            for safe_index in range(0, len(safe_regi
    on)):
                for ddr_index in range(0, len(ddr_pr
    ime_ct)):
                    intersect_data = []
                    intersect_status = True
                    null_counter = 0
                    for i in range(0, data_length):

                        #top
                        if(safe_region[safe_index][i
    ][0] == 'null' and ddr_prime_ct[ddr_index][i][0] ==
     'null'):
                            top = 'null'
                        elif(safe_region[safe_index]
    [i][0] == 'null'):
                            top = ddr_prime_ct[ddr_i
    ndex][i][0]
                        elif(ddr_prime_ct[ddr_index]
    [i][0] == 'null'):
```

```
237.                             top = safe_region[safe_i
    ndex][i][0]
238.                         else:
239.                             top = min(safe_region[sa
    fe_index][i][0], ddr_prime_ct[ddr_index][i][0])
240.
241.                         #bottom
242.                         if(safe_region[safe_index][i
    ][1] == 'null' and ddr_prime_ct[ddr_index][i][1] ==
     'null'):
243.                             bottom = 'null'
244.                         elif(safe_region[safe_index]
    [i][1] == 'null'):
245.                             bottom = ddr_prime_ct[dd
    r_index][i][1]
246.                         elif(ddr_prime_ct[ddr_index]
    [i][1] == 'null'):
247.                             bottom = safe_region[saf
    e_index][i][1]
248.                         else:
249.                             bottom = max(safe_region
    [safe_index][i][1], ddr_prime_ct[ddr_index][i][1])

250.
251.                         max_min_value = [top, bottom
    ]
252.                         intersect_data.append(max_mi
    n_value)
253.
254.                         if(top != 'null' and bottom
    != 'null'):
255.                             if(bottom > top):
256.                                 intersect_status = F
    alse
257.                         elif(top == 'null' and botto
    m == 'null'):
258.                             null_counter += 1
259.
260.                     if(intersect_status == True and
    null_counter != data_length):
261.                         intersection.append(intersec
    t_data)
262.             if(len(intersection) > 0):
```

```
263.                 return True
264.             else:
265.                 return False
266.
267.         def move_query_point():
268.             global intersection
269.             global query_point
270.             global q_cost
271.             global data_length
272.             modified_value = []
273.             distance_value = []
274.             for data_index in range(0, len(intersect
    ion)):
275.                 nearest_point = []
276.                 nearest_distance = []
277.                 for i in range(0, data_length):
278.                     top_diff = abs(float(query_point
    [i+1]) - intersection[data_index][i][0])
279.                     bottom_diff = abs(float(query_po
    int[i+1]) - intersection[data_index][i][1])
280.                     if(bottom_diff < top_diff):
281.                         nearest_point.append(interse
    ction[data_index][i][1])
282.                         nearest_distance.append(bott
    om_diff)
283.                     else:
284.                         nearest_point.append(interse
    ction[data_index][i][0])
285.                         nearest_distance.append(top_
    diff)
286.                 modified_value.append(nearest_point)

287.                 distance_value.append(nearest_distan
    ce)
288.             cheapest_index = None
289.             current_cost = 99999999999
290.             for data_index in range(0, len(modified_
    value)):
291.                 total_cost = 0
292.                 for i in range(0, data_length):
293.                     total_cost += (distance_value[da
    ta_index][i] * q_cost[i])
294.                 if(total_cost < current_cost):
```

```
295.                     cheapest_index = data_index
296.             recommendation = modified_value[cheapest
    _index]
297.
298.
299.
300.         def move_why_not_point(ct, q):      #q here
    is transformed q
301.             global data_length
302.             global ct_cost
303.             global node
304.             global local_skyline
305.             global candidate_skyline
306.             global global_skyline
307.             global shadow_skyline
308.             global virtual_point
309.
310.             A = []
311.             fp = open(product_list)
312.             for line in fp:
313.                 product = line.split()
314.                 transformed_point = []
315.                 for i in range(0, data_length):
316.                     if(product[i+1] != 'null'):
317.                         transformed_value = ct[i] +
    abs(ct[i] - float(product[i+1]))
318.                         transformed_point.append(tra
    nsformed_value)
319.                     else:
320.                         transformed_point.append(ct[
    i])
321.                 A.append(transformed_point)
322.
323.             for data_index in range(0, len(A)):
324.                 status = True
325.                 for i in range(0, data_length):
326.                     if(A[data_index][i] > q[i]):
327.                         status = False
328.                 if(status == False):
329.                     A[data_index][-1] = 'delete'
330.             for i in reversed(A):
331.                 if(i[-1] == 'delete'):
332.                     A.remove(i)
```

```
333.            for data_index in range(0, len(A)):
    #transformasikan terhadap q
334.                for i in range(0, data_length):
335.                    #cari jarak, bukan titik
336.                    A[data_index][i] = abs(q[i] - A[
    data_index][i])
337.                A[data_index].append('ok')
338.            for data_index in range(0, len(A)):
339.                for data_index_2 in range(0, len(A))
    :
340.                    greater = False
341.                    smaller = False
342.                    for i in range(0, data_length):

343.                        if(A[data_index][i] < A[data
    _index_2][i]):
344.                            smaller = True
345.                        elif(A[data_index][i] > A[da
    ta_index_2][i]):
346.                            greater = True
347.                    if(smaller == True and greater =
    = False):
348.                        A[data_index_2][-
    1] = 'delete'
349.                    elif(smaller == False and greate
    r == True):
350.                        A[data_index][-
    1] = 'delete'
351.            for i in reversed(A):
352.                if(i[-1] == 'delete'):
353.                    A.remove(i)
354.            M = []
355.            for data_index in range(0, len(A)):
356.                data = []
357.                for i in range(0, data_length):
358.                    temp = q[i] - (A[data_index][i]
    / 2)
359.                    data.append(temp)
360.                M.append(data)
361.            current_cost = 99999999999
362.            cheapest_index = None
363.            for data_index in range(0, len(M)):
364.                total_cost = 0
```

```
365.                 for i in range(0, data_length):
366.                     if(M[data_index][i] != 'null'):
367.                         total_cost += (abs(M[data_in
    dex][i] - ct[i]) * ct_cost[i])
368.                 if(total_cost < current_cost):
369.                     cheapest_index = data_index
370.                     current_cost = total_cost
371.             for i in range(0, data_length):
372.                 if(M[cheapest_index][i] == 'null'):
373.                     M[cheapest_index][i] = ct[i]
374.             best_value = {}
375.             best_value["q"] = list(q)
376.             best_value["ct"] = list(M[cheapest_index
    ])
377.             best_value["cost"] = current_cost
378.             return best_value
379.
380.
381.         def Move_Why_Not_And_Query_Point():
382.             global safe_region
383.             global query_point
384.             global ct
385.             global product_list
386.             global ct_cost
387.             global q_cost
388.             global start_time
389.
390.
391.             E = []      #corner_points
392.             for safe_index in range(0, len(safe_regi
    on)):
393.                 corner_points = []
394.                 for i in range(0, len(safe_region[sa
    fe_index])):
395.                     top_diff = abs(safe_region[safe_
    index][i][0] - float(ct[i]))
396.                     bottom_diff = abs(safe_region[sa
    fe_index][i][1] - float(ct[i]))
397.                     if(top_diff <= bottom_diff):
398.                         corner_points.append(safe_re
    gion[safe_index][i][0])
```

```
399.                     elif(top_diff > bottom_diff):
400.                         corner_points.append(safe_re
    gion[safe_index][i][1])
401.                 E.append(corner_points)
402.             Q = []
403.             for data_index in range(0, len(E)):
404.                 data = []
405.                 for i in range(0, len(E[data_index])
    ):
406.                     transformed_value = float(ct[i])
     + abs(float(ct[i]) - E[data_index][i])
407.                     data.append(transformed_value)
408.                 data.append('ok')
409.                 Q.append(data)
410.             for data_index in range(0, len(E)):
411.                 greater = False
412.                 smaller = False
413.                 for data_index_2 in range(0, len(E))
    :
414.                     for i in range(0, len(E[data_ind
    ex])):
415.                         if(Q[data_index][i] > Q[data
    _index_2][i]):
416.                             greater = True
417.                         elif(Q[data_index][i] < Q[da
    ta_index_2][i]):
418.                             smaller = True
419.                     if(greater == True and smaller =
    = False):
420.                         Q[data_index_2][-
    1] = 'delete'
421.             for i in reversed(Q):
422.                 if(i[-1] == 'delete'):
423.                     Q.remove(i)
424.
425.             Mc = []
426.             for data_index in range(0, len(Q)):
427.                 T = move_why_not_point(ct, Q[data_in
    dex][:-1])
428.                 Mc.append(T)
429.
430.             cheapest_index = None
431.             cheapest_cost = 99999999999
```

```
            for data_index in range(0, len(Mc)):
                if(Mc[data_index]["cost"] < cheapest
    _cost):
                    cheapest_index = data_index
            print("")
            print("   RESULT : MOVING WHY-
    NOT AND QUERY-POINT")
            print("   Q      : " + str(Mc[cheapest_i
    ndex]["q"]))
            print("   CT     : " + str(Mc[cheapest_i
    ndex]["ct"]))
```

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Tosca Yoel Connery, akrab dipanggil Tosca, lahir pada tanggal 31 Agustus 1995 di Pulau Gadang. Penulis telah menempuh pendidikan di SDN 001 Bagansiapiapi, SMPS Perg. Wahidin Bagansiapiapi, serta SMAN Plus Provinsi Riau. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Departemen Informatika Institut Teknologi Sepuluh Nopember.

Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, diantaranya adalah menjabat sebagai Staff Departemen Pengembangan Profesi Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS periode 2015/2016 dan kemudian dilanjutkan sebagai Kepala Departemen Pengembangan Profesi HMTC ITS periode 2016/2017. Penulis juga aktif dalam kepanitiaan Schematics, antara lain sebagai staff dalam Biro National Seminar of Technology dan Biro Desain Dokumentasi dan Dekorasi masing-masing satu periode. Penulis melaksanakan kerja praktik di Bank Indonesia pada periode Juni-Agustus 2017.

Dalam menyelesaikan pendidikan sarjana, penulis mengambil topik *Data Engineering* yang dibawahi Laboratorium Komputasi Berbasis Jaringan (KBJ). Penulis dapat dihubungi melailui *email* toscayoelconnnery@gmail.com.