TUGAS AKHIR – KI1502

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT ON REACHING K SUBSCRIBERS*

I PUTU EKA WIRA MAHARDIKA
NRP 0511140000025

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 – KI1502

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT ON REACHING K SUBSCRIBERS*

I PUTU EKA WIRA MAHARDIKA
NRP 0511140000025

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 – KI1502

DESIGN AND IMPLEMENTATION APPLICATION TO ANSWER WHY-NOT ON REACHING K SUBSCRIBERS PROBLEM

I PUTU EKA WIRA MAHARDIKA
NRP 05111440000025

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 ON REACHING K SUBSCRIBERS*
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
I PUTU EKA WIRA MAHARDIKA
NRP: 05111440000025

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
JUNI 2018

[Halaman ini sengaja dikosongkan]

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT ON REACHING K SUBSCRIBERS*

Nama Mahasiswa : I Putu Eka Wira Mahardika
NRP : 05111440000025
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

Di era teknologi yang semakin berkembang pesat ini, data merupakan elemen penting dalam pertimbangan dalam pengambilan keputusan, misalnya pada sebuah perusahaan manufaktur. Perusahaan ini tentu ingin produknya menjangkau pelanggan sebanyak-banyaknya, namun dengan biaya yang serendah-rendahnya. Jika jumlah pelanggan tidak sesuai ekspektasi perusahaan, maka perusahaan harus memperbaiki produknya, namun tetap mempertimbangkan biaya perubahan yang sekecil mungkin. Permasalahan ini didefinisikan dalam penelitian ini sebagai "*Why Not on Reaching k Subscribers*".

Penelitian ini berusaha untuk menjawab tantangan tersebut dengan ilmu rekayasa data. Merujuk pada permasalahan perusahaan manufaktur sebelumnya, tentu data pelanggan adalah data dengan jumlah yang sangat banyak. Untuk mengoptimalkan kecepatan dan biaya komputasi, maka penelitian ini menawarkan algoritma *indexing* dengan struktur data graf yang mengadopsi algoritma Close Dominance Graph (CDG). Untuk menghitung biaya perubahan dalam rangka memperbaiki jumlah pelanggan, penelitian ini juga menawarkan solusi untuk menghitung biaya perubahan agar solusi yang ditawarkan dipastikan merupakan solusi terbaik.

Hasil uji coba menunjukkan bahwa algoritma yang ditawarkan dapat memberikan solusi terbaik dengan waktu eksekusi yang lebih baik dalam waktu eksekusi secara signifikan dibandingkan dengan algoritma konvensional *brute force*. Solusi untuk menghitung biaya perubahan juga dapat digunakan untuk himpunan data dengan rentang nilai yang berbeda-beda, sehingga algoritma yang ditawarkan dapat diimplementasikan untuk beragam jenis himpunan data.

Kata kunci: Rekayasa Data, Why-Not, Close Dominance Graph, Indexing

DESIGN AND IMPLEMENTATION APPLICATION TO ANSWER WHY-NOT ON REACHING K SUBSCRIBERS PROBLEM

Student's Name	**:**	**I Putu Eka Wira Mahardika**
Student's ID	**:**	**05111440000025**
Department	**:**	**Department of Informatics, Faculty of ICT - ITS**
First Advisor	**:**	**Henning Titi Ciptaningtyas, S.Kom., M.Kom.**
Second Advisor	**:**	**Bagus Jati Santoso, S.Kom., Ph.D.**

ABSTRACT

In this era of rapidly growing technology, data is an important element in consideration in decision making, for example in a manufacturing company. The company certainly wants its products to reach subscribers as many as possible, but at the lowest possible cost. If the number of subscribers does not match the expectations of the company, then the company should improve some attributes of its product, but still consider the cost of change as small as possible. This problem is defined in this research as "Why Not on Reaching k Subscribers".

This research seeks to address these challenges with data engineering. Referring to the problems of previous manufacturing companies, of course, customer data is data with a very large number. To optimize the speed and cost of computation, this research offers an indexing algorithm with a graph data structure that adopts the Close Dominance Graph (CDG) algorithm. To calculate the cost of change in order to improve the number of subscribers, this study also offers a solution to calculate the cost of change so that the solution offered is certainly the best solution.

The results of the test show that the proposed algorithm can provide the best solution with better execution time significantly compared with conventional brute force algorithm. Solutions for calculating change costs can also be used for data sets with different value ranges, so that the proposed algorithm can be implemented for different types of data sets.

Keywords: Data Engineering, Why-Not, Close Dominance Graph, Indexing

KATA PENGANTAR

Segala puji dan syukur bagi Tuhan Yang Maha Kuasa, yang telah melimpahkan nikmat dan anugerahnya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul

"DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT ON REACHING K SUBSCRIBERS*".

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. Tuhan Yang Maha Kuasa, karena atas karunia-Nya penulis dapat menyelesaikan Tugas Akhir dengan baik.
2. Kedua orang tua saya I Ketut Wirata serta Ni Made Suantari, terimakasih atas doa dan bantuan moral maupun material selama penulis menempuh pendidikan di Departemen Informatika ITS.
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.
6. Bapak Dr. Radityo Anggoro, S.Kom.,M.Sc. selaku koordinator TA, dan segenap dosen Teknik Informatika yang telah banyak memberikan ilmu kepada penulis.
7. Bapak dan Ibu Hadi beserta keluarga atas kesempatan tempat tinggal yang diberikan dan telah membimbing penulis selama 4 tahun di tanah rantau.
8. Rekan-rekan *Data Engineers* Dwika, Tosca, Ovan, Syukron, Sekbay, dan Cahya sebagai teman seperjuangan yang telah menemani penulis selama pengerjaan Tugas Akhir.
9. Fandy, Rina, Winda, Dharmawan, Widhi, dan Nobby yang telah menemani penulis ketika rindu kampung halaman.
10. Seluruh rekan TC 2014 yang saya cintai dan saya banggakan.
11. Rekan-rekan kontrakan Denny, Bagus Andi, dan Murata atas segala dukungan dan waktunya dalam menemani penulis selama 4 tahun di tanah rantau.
12. Rekan-rekan TPKH-ITS, terimakasih telah memberikan penulis banyak pelajaran dan pengalaman terutama untuk angkatan 2014 Laksmana yang saya cintai.

Penulis memohon maaf apabila terdapat kekurangan dalam penulisan Tugas Akhir ini. Kritik dan saran penulis harapkan untuk perbaikan dan pembelajaran di kemudian hari. Semoga Tugas Akhir ini dapat memberikan manfaat yang sebesar-besarnya.

Surabaya, Juni 2018

I Putu Eka Wira Mahardika

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

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

1.1. Latar Belakang

Penelitian dan pengembangan performa, kualitas, dan pengguna an sistem basis data selama dekade terakhir ini telah mendapat perhatian lebih dalam beberapa tahun terakhir [1]. Berbagai macam algoritma *query* untuk membantu pengguna mencari data dengan nilai atribut terbaik seperti *Top-K Query*, *Skyline Query*, dan *Top-K Dominating Query* mulai dikembangkan. Dengan dikembangkannya berbagai macam algoritma *query* tersebut, maka mulai muncul berbagai permasalahan baru, salah satunya adalah dimana saat pengguna merasa hasil dari suatu *query* tidak sesuai dengan ekspektasi dan mulai bertanya kenapa data tertentu tidak muncul di hasil *query*. Permasalahan tersebut juga dikenal dengan "*Why-Not Question*". Permasalahan tersebut sudah beberapa kali diangkat untuk menjadi topik jurnal, harapannya dengan dikembangkannya algoritma yang dapat menjawab permasalahan tersebut dengan memperbaiki *query* awal agar dapat mencapai ekspektasi namun tetap memperhitungkan penalti dari perubahannya.

Berangkat dari permasalahan "*Why-Not Question*", terdapat masalah lain yang dapat diangkat untuk dicari jalan keluarnya. Diketahui bahwa pengguna memiliki sebuah produk dengan suatu nilai tertentu, dan terdapat sebuah himpunan data yang berisi data preferensi untuk masing-masing konsumen. Kemudian pengguna melakukan *query* untuk mengetahui berapa banyak data konsumen yang kriterianya cocok dengan produk tersebut. Permasalahan akan muncul saat jumlah konsumen hasil *query* tersebut lebih rendah dari ekspektasi pengguna. Permasalahan seperti ini selanjutnya akan disebut dengan istilah

"*Why-Not on Reaching k Subscribers*" dan konsumen tersebut akan diistilahkan sebagai *subscribers*. Perbedaan mendasar dari permasalahan "*Why-Not Question*" dengan "*Why-Not on Reaching k Subscribers*" terletak pada masukan permasalahan. Pada "*Why-Not Question*" yang menjadi masukan adalah data mana yang hilang dari hasil *query*, sedangkan pada "*Why-Not on Reaching k Subscribers*" adalah jumlah data yang diharapkan dari hasil *query*,

Berdasarkan uraian di atas maka dapat dilihat bahwa dibutuhkan sebuah algoritma untuk menjawab permasalahan "*Why-Not on Reaching k Subscribers*". Algoritma yang akan dirancang harapannya dapat memperbaiki nilai pada setiap atribut dari produk agar hasil *query* dapat mencapai jumlah data yang diharapkan pengguna. Dalam memperbaiki *query*, nilai penalti juga akan dipertimbangkan sebagai biaya dari suatu perubahan. Dengan begitu diharapkan juga hasil perbaikan nilai atribut juga memiliki nilai penalti serendah mungkin.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana mengindeks seluruh data *subscribers* untuk mendukung solusi atas permasalahan *Why-Not on Reaching k Subscribers*?
2. Bagaimana algoritma yang tepat untuk memperbaiki *query* agar mencapai ekspektasi pengguna aplikasi terhadap jumlah *subscribers*?
3. Bagaimana pembuatan mekanisme penalti yang tepat untuk mengukur kualitas atas solusi perbaikan yang diberikan?

1.3. Batasan Masalah

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

1. Algoritma ini hanya memproses nilai atribut yang bertipe numerik.
2. Himpunan data yang digunakan adalah data *real-life* dan sintetis.
3. Nilai atribut pada data dinormalisasi terlebih dahulu untuk mempermudah penghitungan nilai penalti.

1.4. Tujuan

Tugas akhir ini mempunyai beberapa tujuan, yaitu sebagai berikut:

1. Menemukan formulasi yang tepat dalam mengindeks seluruh data konsumen untuk mendukung solusi atas permasalahan *Why-Not on Reaching k Subscribers*
2. Merancang algoritma yang tepat untuk memperbaiki nilai atribut produk agar mencapai ekspektasi pengguna aplikasi terhadap jumlah konsumen.
3. Membuat mekanisme penalti yang tepat untuk mengukur kualitas atas solusi perbaikan yang diberikan.

1.5. Manfaat

Manfaat dari pembuatan tugas akhir ini adalah:

1. Memberikan solusi bagi produsen untuk memperbaiki produknya secara efektif.
2. Memberikan kontribusi dalam pengembangan dan penelitian di bidang rekayasa data.

1.6. Metodologi Pembuatan Tugas Akhir

Tahapan-tahapan yang dilakukan dalam pengerjaan tugas akhir ini adalah sebagai berikut:

1. Penyusunan proposal tugas akhir.

   Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil

pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

2. Studi literatur

   Pada studi literatur ini, akan dipelajari beberapa sejumlah referensi yang akan diperlukan untuk merancang algoritma yaitu mengenai *refined query*.

3. Analisis dan desain perangkat lunak

   Pada tahap ini permasalahan ke dimodelkan dalam beberapa bentuk notasi matematika untuk mempermudah proses perancangan desain dari algoritma dan struktur data.

4. Implementasi perangkat lunak

   Implementasi algoritma ini akan dibuat dalam bentuk program konsol yang dibangun dengan menggunakan bahasa pemrograman Python. IDE yang akan digunakan adalah Atom.

5. Pengujian dan evaluasi

   Pengujian dalam algoritma ini akan dilakukan dalam beberapa cara, antara lain:

   1. Pengujian Fungsionalitas

      Pengujian ini akan berfokus pada ketercapaian kebutuhan fungsionalitas aplikasi.

   2. Pengujian Waktu Eksekusi

      Pengujian ini akan berfokus pada seberapa lama waktu yang dibutuhkan untuk mengeksekusi algoritma dalam menjawab permasalahan *Why-Not on Reaching k Subscribers*.

3. Pengujian Penggunaan Memori

   Pengujian ini akan berfokus pada pengukuran besarnya memori yang digunakan saat aplikasi dijalankan.

6. Penyusunan buku tugas akhir

   Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat. 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. Analisis dan Perancangan Sistem
   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

Bab yang berisi mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, dan manfaat dari pembuatan tugas akhir. Selain itu metodologi yang

digunakan dan sistematika penulisan laporan akhir juga merupakan bagian dari bab ini.

Bab II Tinjauan Pustaka

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

Bab III Analisis dan Perancangan Sistem

Bab ini berisi tentang analisis permasalahan, deskripsi umum sistem, spesifikasi kebutuhan perangkat lunak, lingkungan perancangan, perancangan arsitektur sitem, diagram kelas, dan struktur data.

Bab IV Implementasi

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

Bab V Pengujian dan Evaluasi

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

Bab VI Kesimpulan dan Saran

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

BAB II
TINJAUAN PUSTAKA

Bab ini menjelaskan teori-teori yang berkaitan dengan Desain Dan Implementasi Aplikasi Untuk Menjawab Permasalahan *Why-Not on Reaching k Subscribers* yang diajukan untuk tugas akhir ini. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap perangkat lunak yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

Penelitian ini menggunakan beberapa istilah penting untuk menyederhanakan penulisan. Daftar istilah yang digunakan pada penelitian ini dapat dilihat pada Tabel 2.1.

Tabel 2.1 Daftar Istilah

Simbol	**Keterangan**
n	jumlah data pada suatu himpunan data
r_i	data ke-i, dimana *i = 1*, …, *n*
$p \prec q$	data *p* mendominasi *q*
$p \rightarrow q$	data *p closely dominates q* pada CDG
$A(r)$	sebuah kumpulan data a_i, dimana $i = 1, 2, \ldots$ dan $a_i \prec r$
Q	*query awal* yang belum dilakukan proses *query refinement*
Q'	hasil *query refinement* dari query awal *Q*
O_i	tahap ke-*i* dalam proses *query refinement*
d	jumlah dimensi
$ancscore(r)$	jumlah data yang mendominasi r + 1 atau jumlah data pada $A(r)$ + 1
$C(r)$	sebuah himpunan data c_i, dimana $i = 1, 2, \ldots$ dan $r \rightarrow c_i$
$D(r)$	sebuah himpunan data d_i, dimana $i = 1, 2, \ldots$ dan $r \prec d_i$
$\overline{sky}(R)$	hasil *skyline query* dari kumpulan data *R*

Simbol	Keterangan
k	jumlah ekspektasi *subscribers*
$Skyline(R)$	hasil *skyline query* dari kumpulan data *R*
q_o	*query* awal sebelum dilakukan *query refinement*
q'	hasil *query refinement*
$CAND$	himpunan kandidat q'
CDG	*close dominance graph* yang dibentuk dari suatu himpunan data
$ROOT$	himpunan *vertex* pada CDG yang tidak didominasi oleh *vertex* manapun pada CDG dan menjadi titik awal dalam penelusuran CDG
$LAYER$	himpunan *vertex* yang menjadi lapisan ROOT virtual pada CDG
$layerdist$	jarak antar lapisan *LAYER*
$LAYER_l$	himpunan *vertex* yang menjadi lapisan ke-*l* pada *LAYER*, dimana *l* = *layerdist*, *layerdist* × 2, *layerdist* × 3, …
P_o	himpunan nilai atribut p_{o_i} pada q_o, dimana i = {1,2,…,*d*}
P'	himpunan nilai atribut p'_i pada q', dimana i = {1,2,…,*d*}
p_{min}	nilai atribut terkecil pada suatu himpunan data
p_{max}	nilai atribut terbesar pada suatu himpunan data
w_i	himpunan koefisien toleransi perubahan nilai atribut p_i pada q' dimana i = {1,2,…,*d*}
$Penalty(q_o, q')$	nilai penalti *refined query* q' dari perubahan nilai atribut terhadap *query* awal q_o yang dihitung dengan persamaan 3.2

2.1. Query Refinement

Query Refinement merupakan suatu pendekaatan untuk menghasilkan *query* baru agar dapat menjawab permasalahan pada hasil *query* seperti *Why* dan *Why-Not*, *query* baru tersebut selanjutnya akan disebut *refined query*. Hasil riset terakhir pada [1] tentang pendekatan *query-refinement* menjelaskan 2 kriteria untuk refined query yang baik. Pertama *query* tersebut bersifat *similar* — memiliki sedikit perubahan jika dibandingkan dengan *query* aslinya. Kriteria kedua adalah *refined query* yang baik bersifat *precise* — memiliki sedikit data tambahan pada hasil *query*nya.

Karena jumlah kemungkinan *refined query* terhadap sebuah *query* sangat banyak, jadi diperlukan suatu metrik (alat ukur) untuk membandingkan kualitas dari sebuah *refined query*. Metrik tersebut akan bermanfaat agar algoritma yang memproduksi *refined query* hanya mengembalikan sebuah *query* terbaik sebagai solusinya. Sesuai dengan 2 kriteria dari *refined query* yang baik, maka kriteria tersebut dapat dijadikan sebagai metrik atas kualitas *refined query*.

Yang pertama adalah metrik ketidakmiripan. *Refined query* yang baik seharusnya semirip mungkin dari *query* aslinya. Diketahui sebuah *query* asli *Q* dan refined query *Q'*, kita bandingkan tingkat kemiripan antara *Q* dan *Q'* dengan mengukur jarak/selisih dari perubahan yang paling minimal. Dengan demikian *query* tersebut dikatakan lebih mirip (atau lebih tidak mirip) terhadap satu sama lain jika jarak perbahannya kecil (begitu pula sebaliknya). Karena hasil dari *Q* dan *Q'* bersifat *union-compatible* (seluruh atribut dalam klausa *select* pada *Q* dan *Q'* adalah sama), kita hanya mempertimbangkan klausa *from* dan *where* sebagai operator perubahan untuk mengubah *Q* menjadi *Q'*. Empat kunci dalam operasi perubahan *query* adalah: (O_1) ubah beberapa nilai konstan di predikat *select* pada klausa *where*, (O_2) tambahkan sebuah predikat *select* pada klausa *where*, (O_3) tambahkan/hilangkan predikat *join* pada klausa *where*, dan (O_4) tambahkan/hilangkan *relation* pada klausa *from*. Perlu diketahui bahwa tidak ada perubahan operator secara eksplisit saat

menghilangkan sebuah predikat *select* seperti yang dimodelkan pada O_1. Proses menghilangkan predikat *select* secara efisiensi setara dengan mengubah jarak dari nilai *select* untuk menutupi domain dari seluruh atribut. Selanjutnya, saat O_4 digunakan untuk menghilangkan relasi R_i pada klausa *from*, seluruh predikat *select* dan *join* yang berasosiasi dengan R_i juga dihilangkan sebagai bagian dari operasi perubahan.

Dinyatakan w_i sebagai biaya untuk operasi perubahan Oi, $i \in [1,4]$. Dengan begitu dapat diasumsikan bahwa $w_1 < w_2 < w_3 < {w_4}^2$. Dinyatakan n_i sebagai total dari operasi O_i yang digunakan untuk merubah Q menjadi Q', $i \in [1,4]$. Jarak perubahan untuk transformasi ini diketahui sebagai $\sum_{1 \le i \le 4}(w_i \times n_i)$. Mengacu pada jarak perubahan minimum untuk mengubah Q menjadi Q' sebagai ukuran untuk perbedaan antara Q dan Q', yang dapat dihitung secara efisien untuk Q dan Q'.

Yang kedua adalah metrik ketidaktepatan. *Refined query* yang baik seharusnya hanya menghasilkan data yang dihasilkan pada *query* aslinya dan dengan tambahan data yang memang diinginkan (dalam hal ini data yang hilang pada permasalahan *Why-Not*). Data-data yang tidak diinginkan untuk tampil di dalam hasil *query* seharusnya dapat diminimalisir.

Skyline refined query adalah *refined query* yang memiliki nilai terbaik (paling rendah) pada metrik ketidakmiripan maupun pada metrik ketidaktepatan. Dari seluruh hasil *refined query* yang mungkin akan menjadi solusi, maka *Skyline refined query* adalah solusi terbaik [1].

Penelitian ini menggunakan konsep query refinement untuk menjawab permasalahan *Why-Not on Reaching k Subscribers* agar solusi yang dihasilkan dapat mencapai metrik ketidakmiripan dan ketidaktepatan sebaik mungkin.

2.2. *Query* Berbasis Preferensi

Preferensi telah menjadi peran utama dalam berbagai disiplin ilmu, termasuk pada ilmu komputer seperti kecerdasan buatan, interaksi manusia dan komputer, dan basis data. Pada basis data, representasi dari preferensi memiliki dua kategori utama yaitu

preferensi kualitatif dan preferensi kuantitatif. Preferensi kualitatif dispesifikasikan menggunakan predikat biner yang dapat membandingkan antara suatu data dengan data lainnya secara relatif. Sedangkan preferensi kuantitatif diekspresikan dengan menetapkan skor pada suatu data atau pada elemen *query* yang mendeskripsikan sebuah kumpulan data, yang dimana skor tersebut mengekspresikan derajat ketertarikan (*degree of interest*). Pada preferensi kuantitatif, suatu data *a* dikatakan lebih disukai (*prefered*) apabila skor untuk data *a* lebih tinggi dari skor untuk data *b* [2].

2.3. Dominasi Data

Dominasi data adalah suatu kondisi dimana sebuah objek memiliki nilai atribut lebih baik secara keseluruhan objek. Sebagai contoh, objek *a* dikatakan mendominasi objek *b* jika seluruh nilai atribut pada *a* tidak ada yang lebih buruk dari *b* dan setidaknya satu atribut pada *a* bernilai lebih baik daripada *b* [3].

2.4. Graf

Dalam ilmu matematika dan ilmu komputer, terutama pada teori graf, graf adalah sebuah struktur dari sekumpulan objek yang memiliki relasi dengan objek lainnya. Objek tersebut dalam abstraksi matematika disebut dengan *vertex* (disebut juga simpul, *node*, atau titik) dan setiap pasangan relasi antar *vertex* dinamakan *edge* (disebut juga garis atau busur). Biasanya, graf digambarkan dalam bentuk diagram sebagai serangkaian titik sebagai *vertex*, dan dihubungkan dengan garis sebagai *edge* [4].

Pada penelitian ini graf digunakan sebagai struktur data untuk menunjang performa algoritma.

2.5. Min-Max Algorithm

Algoritma Min-Max merupakan salah satu metode dalam melakukan normalisasi data. Normalisasi data adalah sebuah metode untuk mengatasi ketidakseimbangan nilai yang terdapat pada suatu himpunan data. Nilai-nilai yang tidak seimbang tersebut harus diubah ke dalam suatu rentang yang sama. Cara kerja algoritma ini adalah dengan melakukan penskalaan data pada

rentang tertentu, dimana rentang yang digunakan umumnya adalah 0-1. Persamaan untuk algoritma ini adalah sebagai berikut [5]:

$$X_n = \frac{X_0 - X_{min}}{X_{max} - X_{min}} \quad (2.1)$$

dimana,

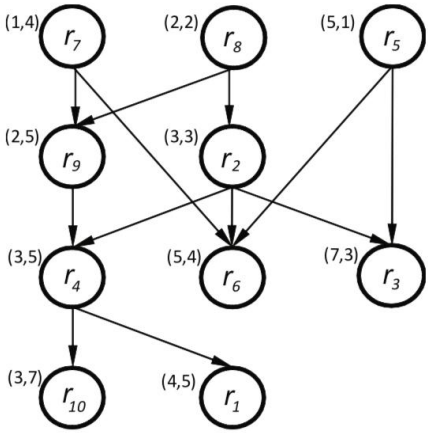
X_n = nilai baru untuk variabel X
X_0 = nilai awal dari variabel X
X_{min} = nilai paling kecil pada himpunan data
X_{max} = nilai paling besar pada himpunan data

2.6. Close Dominance Graph (CDG)

Close Dominance Graph (CDG) adalah sebuah kerangka kerja yang memodelkan himpunan berdasarkan *close dominance relation* antar data. Diketahui sebuah set R yang berisi kumpulan data dalam *multidimensional space*. $A(x)$ merepresentasikan himpunan pada R yang mendominasi data $x \in R$, contohnya $A(x) = \{y | y \in R, y \prec x\}$. Sebuah data $x \in R$ dikatakan *closely dominated* oleh data lain $s \in R$, jika dan hanya jika $s \in A(r)$ dan s tidak didominasi oleh data lain $s \in A(r)$. Relasi ini dilambangkan dengan $s \rightarrow r$.

Close Dominance Graph (CDG) dari R adalah sebuah graf berarah, dimana R sebagai kumpulan *vertex*, dimana setiap dua *vertex* s dan r terhubung oleh relasi berarah dari s ke r, jika dan hanya jika terdapat $s \rightarrow r$.

Jika terdapat $y \prec x$, maka disana terdapat setidaknya satu relasi berarah dari y menuju x. Jadi kesimpulannya, CDG bersifat *acyclic*.



Gambar 2.1 Contoh CDG dari 10 data [6]

Pada Gambar 2.1 diilustrasikan 10 data dengan 2 dimensi atribut, dimana r_4 didominasi oleh $A(r_4) = \{r_2, r_7, r_8, r_9\}$. Karena r_2 dan r_9 tidak mendominasi data lain pada $A(r_4)$, maka terbentuk relasi $r_2 \rightarrow r_4$ dan $r_9 \rightarrow r_4$ pada CDG [6]. Penelitian ini menggunakan konsep CDG sebagai struktur data untuk menunjang performa algoritma.

2.7. Python

Python adalah bahasa pemrograman umum tingkat tinggi yang dapat diterapkan ke berbagai kelas masalah. Bahasa ini dilengkapi dengan pustaka standar besar yang mencakup area seperti pemrosesan string (*regular expression*, Unicode, menghitung perbedaan antar file), protokol Internet (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, pemrograman CGI), rekayasa perangkat lunak (pengujian unit, pencatatan, pembuatan profil, penguraian kode Python), dan antarmuka sistem operasi (panggilan sistem, sistem file, soket TCP / IP) [7]. Pada penelitian ini Python digunakan sebagai bahasa pemrograman untuk mengimplementasikan algoritma.

2.8. Apache HTTP *Server*

Apache HTTP *Server* adalah *server web* lintas-platform *open-source* dan gratis, yang dirilis di bawah ketentuan Apache License 2.0. Apache dikembangkan dan dikelola oleh komunitas pengembang terbuka di bawah naungan Apache Software Foundation.

Apache mendukung berbagai fitur, banyak diimplementasikan sebagai modul yang dikompilasi yang memperluas fungsionalitas inti. Hal ini dapat berkisar dari dukungan bahasa pemrograman sisi *server* untuk skema otentikasi. Beberapa antarmuka bahasa umum mendukung Perl, Python, Tcl dan PHP. Modul otentikasi populer termasuk *mod_access*, *mod_auth*, *mod_digest*, dan *mod_auth_digest*, penerus *mod_digest* [8]. Pada penelitian ini Apache digunakan sebagai *web server* pada tahap implementasi. Versi yang digunakan adalah Apache 2.4.

2.9. PHP

PHP (akronim untuk PHP: *Hypertext Preprocessor*) adalah bahasa *scripting open source* yang digunakan secara luas dan untuk tujuan umum yang sangat cocok untuk pengembangan *web* dan dapat dimasukkan ke dalam HTML.

Yang membedakan PHP dari sesuatu seperti *client-side* JavaScript adalah bahwa kode dijalankan di *server*, menghasilkan HTML yang kemudian dikirim ke *client*. *Client* akan menerima hasil menjalankan skrip tersebut, tetapi tidak akan tahu apa *source code*-nya. Kita bahkan dapat mengkonfigurasi *server web* untuk memproses semua file HTML dengan PHP, dan kemudian benar-benar tidak ada cara bagi pengguna untuk mengetahui apa yang kita miliki [9]. Pada penelitian ini PHP digunakan sebagai bahasa pemrograman untuk menghubungkan antarmuka pengguna dengan algoritma berbasis CDG. PHP yang digunakan adalah versi 7.1.16.

2.10. Vis.JS

Vis.JS merupakan pustaka visualisasi berbasis peramban yang dinamis. Pustaka ini dirancang agar mudah digunakan, untuk menangani sejumlah data yang besar dan dinamis, dan untuk

memungkinkan manipulasi dan interaksi dengan data. Pustaka ini terdiri dari komponen DataSet, Timeline, Network, Graph2d dan Graph3d [10]. Pada penelitian ini Vis.JS digunakan sebagai pustaka untuk melakukan visualisasi struktur data. Vis.JS yang digunakan adalah versi 4.21.0.

2.11. Metode Brute Force

Dalam ilmu komputer, metode pencarian *Brute Force* atau pencarian lengkap, juga dikenal sebagai hasilkan dan tes (*generate and test*), adalah teknik pemecahan masalah yang sangat umum yang terdiri dari penghitungan secara sistematis pada semua kandidat yang mungkin untuk solusi dan memeriksa apakah masing-masing kandidat memenuhi pernyataan masalah.

Metode *Brute Force* cenderung sederhana untuk diterapkan, dan akan selalu menemukan solusi jika ada, biayanya sebanding dengan jumlah solusi kandidat – yang mana dalam praktiknya cenderung tumbuh sangat cepat karena ukuran masalah meningkat. Oleh karena itu, metode *Brute Force* biasanya digunakan ketika ukuran masalah terbatas, atau ketika terdapat heuristik pada spesifikasi masalah yang dapat digunakan untuk mengurangi kandidat solusi ke ukuran yang lebih kecil. Metode ini juga digunakan ketika kesederhanaan implementasi lebih penting daripada kecepatan [11]. Pada penelitian ini metode *Brute Force* digunakan sebagai landasan dasar untuk membangun algoritma pembanding.

[Halaman ini sengaja dikosongkan]

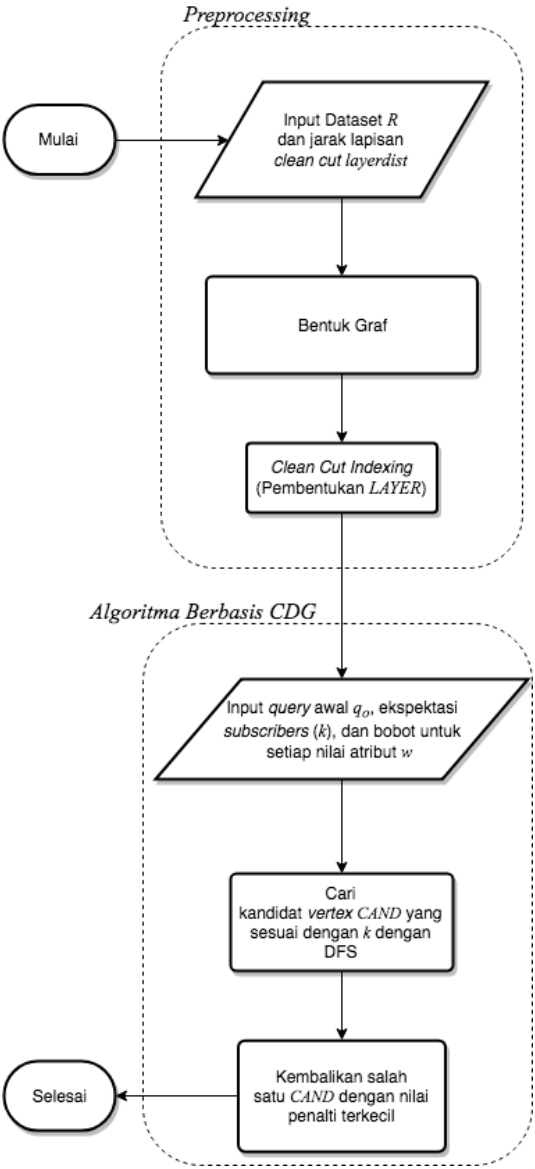
BAB III
DESAIN DAN PERANCANGAN

Pada bab ini akan dijelaskan mengenai perancangan sistem perangkat lunak yang akan dibuat. Perancangan akan dibagi menjadi empat proses utama, yaitu:

1. *Data preprocessing* untuk mengindeks data berdasarkan dominasinya terhadap data lainnya menggunakan struktur data graf dan menerapkan konsep *Close Dominance Graph* (CDG).
2. Penerapan algoritma untuk menjawab permasalahan *Why-Not on Reaching k Subscribers* dari struktur data graf yang telah dibentuk pada proses *data preprocessing*.
3. Perancangan algoritma pembanding dengan metode *Brute Force*.
4. Perancangan arsitektur aplikasi.

3.1. Desain Metode Algoritma Utama Secara Umum

Pada Tugas Akhir ini, diagram alur kerja sistem secara umum akan disajikan dalam bentuk *flowchart* yang dapat dilihat pada Gambar 3.1.

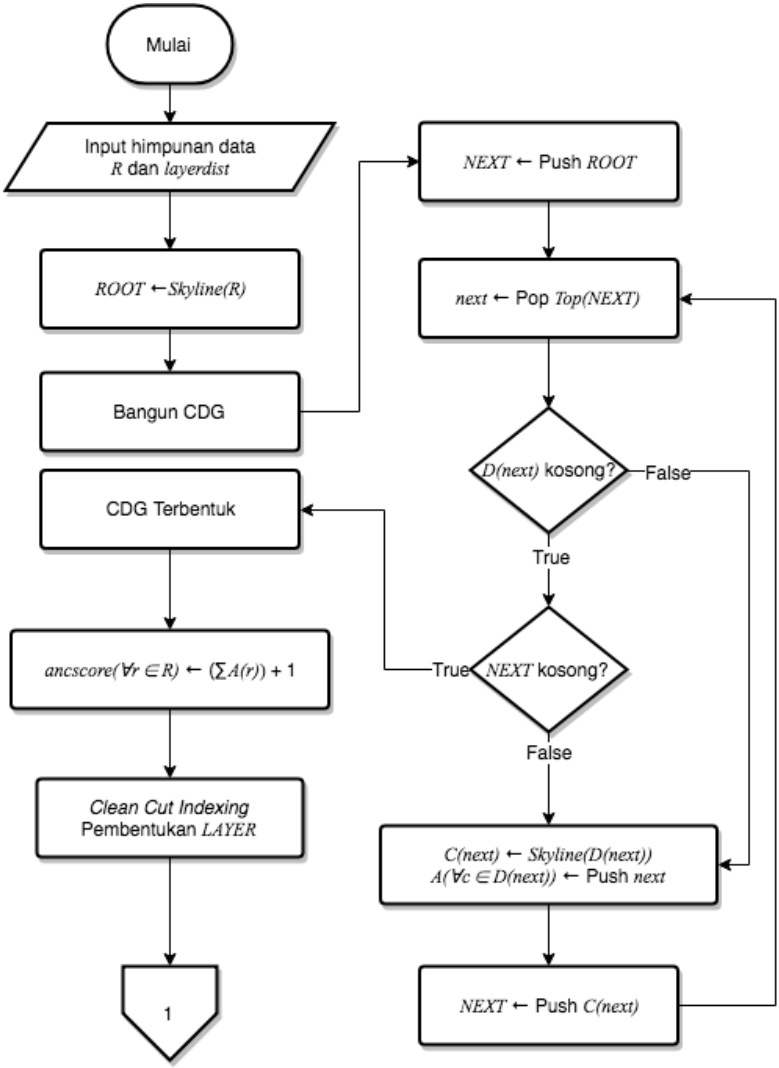


Gambar 3.1 Ilustrasi Alur Program

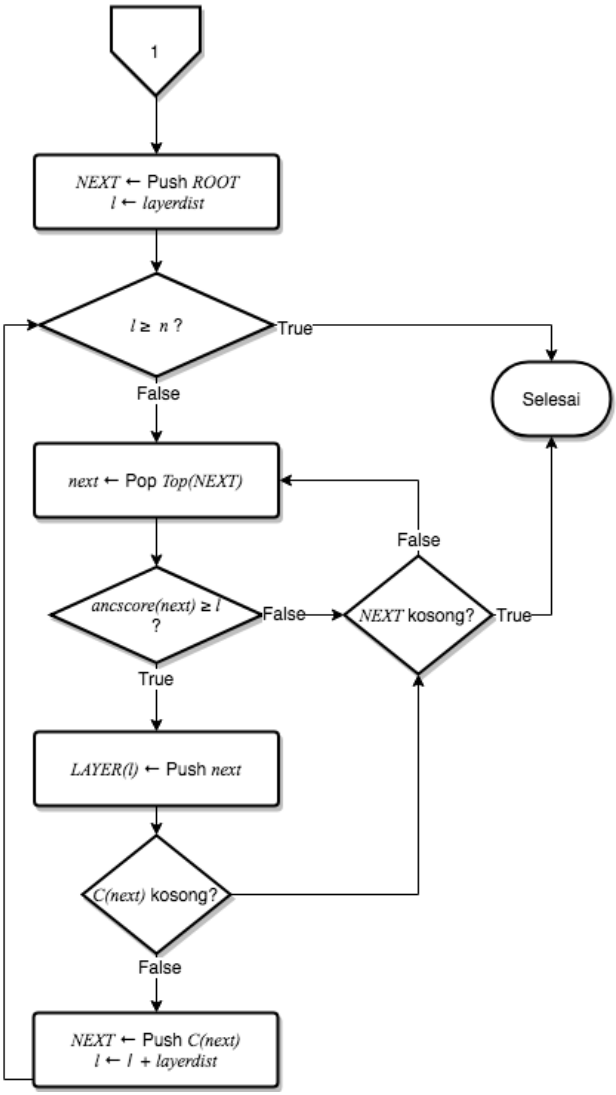
Secara garis besar alur program dibagi menjadi dua tahap, yaitu *Preprocessing* dan program utama (selanjutnya disebut Algoritma Berbasis CDG). Pada masing-masing tahap tersebut didukung oleh beberapa algoritma dasar dalam rekayasa data seperti pencarian *skyline* pada suatu set data dan pengecekan dominasi data. Himpunan data yang akan digunakan akan dilakukan tahap *Preprocessing* terlebih untuk melakukan *indexing* dengan membentuk struktur data graf, yaitu *Close Dominance Graph* (CDG). Proses *indexing* kemudian dilanjutkan dengan menciptakan lapisan *clean cut* yang selanjutnya akan disebut dengan *LAYER*. Setalah CDG dan *LAYER* terbentuk, maka barulah dilanjutkan ke tahap Algoritma Berbasis CDG. Pada Algoritma Berbasis CDG, pengguna menginputkan sebuah *query* preferensi asli q_0, preferensi perubahan nilai atribut pada *query* λ_P, dan ekspektasi *subscribers k*. Program akan menelusuri graf yang sebelumnya telah dibentuk dengan algoritma DFS hingga menemukan data yang sesuai dengan ekspektasi pengguna *k*. Data yang akan dipilih sebagai solusi terbaik adalah data dengan nilai penalti yang paling rendah. Tahap terakhir adalah visualisasi data yang menampilkan hasil pembentukan CDG ke tampilan *web*.

3.1.1. *Preprocessing*

Dalam mengawali proses pengerjaan program, maka harus disiapkan terlebih dahulu data yang akan digunakan dalam proses pengerjaan. Dalam rekayasa data, *Preprocessing* merupakan tahapan yang penting untuk dilakukan karena akan menunjang algoritma pada Algoritma Berbasis CDG. Jumlah data yang sangat banyak dan kebutuhan untuk mendapatkan solusi dalam waktu yang singkat merupakan tantangan pada tahap *Preprocessing*. Ilustrasi alur pada tahap *Preprocessing* dapat dilihat pada Gambar 3.2 dan Gambar 3.3.



Gambar 3.2 Ilustrasi Alur *Preprocessing* (Bagian 1)

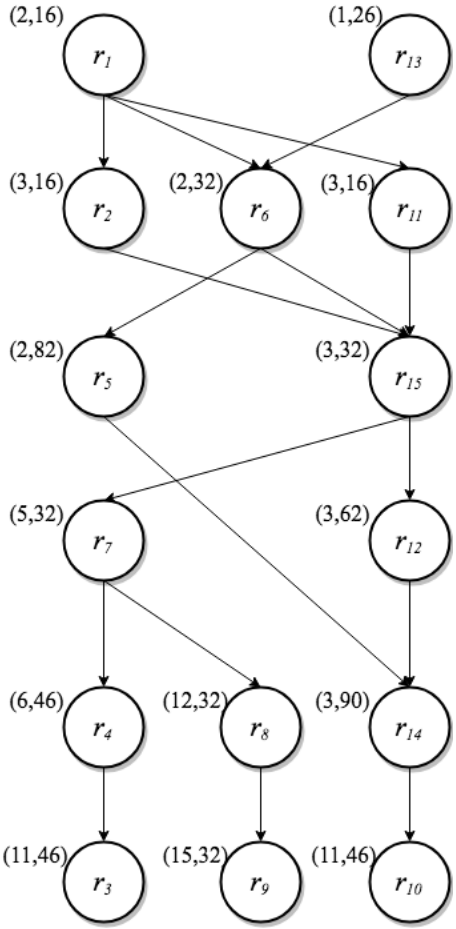


Gambar 3.3 Ilustrasi Alur *Preprocessing* (Bagian 2)

Algoritma *indexing* pada penelitian ini menggunakan struktur data graf (CDG). Diberikan himpunan data *R* dengan ukuran dua dimensi, maka setiap data *r* pada himpunan data akan menjadi *vertex* dan *close dominance relation* setiap *vertex* akan menjadi *edge*. Setiap data *r* pada himpunan data *R* memiliki skor *ancscore(r)* yang merupakan jumlah *vertex* lain yang mendominasi *r*. Atau dapat dikatakan skor *ancscore(r)* adalah total jumlah data yang mendominasi *vertex* tersebut ditambah 1. Nilai *ancscore(r)* ini akan menjadi patokan untuk mengetahui berapa banyak data yang yang mendominasi *r*. Contoh himpunan data *R* dapat dilihat pada Tabel 3.1. Hasil dari CDG*(R)* dapat dilihat pada Gambar 3.4 dan untuk *ancscore(r)* dapat dilihat pada Tabel 3.2. Pada Gambar 3.4 sebuah data dilambangkan dengan lingkaran dengan angka pada pojok kanan atas sebagai nilai atribut a_1 dan a_2 secara berurutan. *Close dominance relation* antar *vertex* dilambangkan dengan garis panah.

Tabel 3.1 Contoh himpunan data *R* dengan 10 data

r	a_1	a_2
r_1	2	16
r_2	3	16
r_3	4	126
r_4	6	46
r_5	2	82
r_6	2	32
r_7	5	32
r_8	12	32
r_9	15	32
r_{10}	11	46
r_{11}	3	16
r_{12}	3	62
r_{13}	1	26
r_{14}	3	90
r_{15}	3	92



Gambar 3.4 Contoh hasil graf dari himpunan data *R*

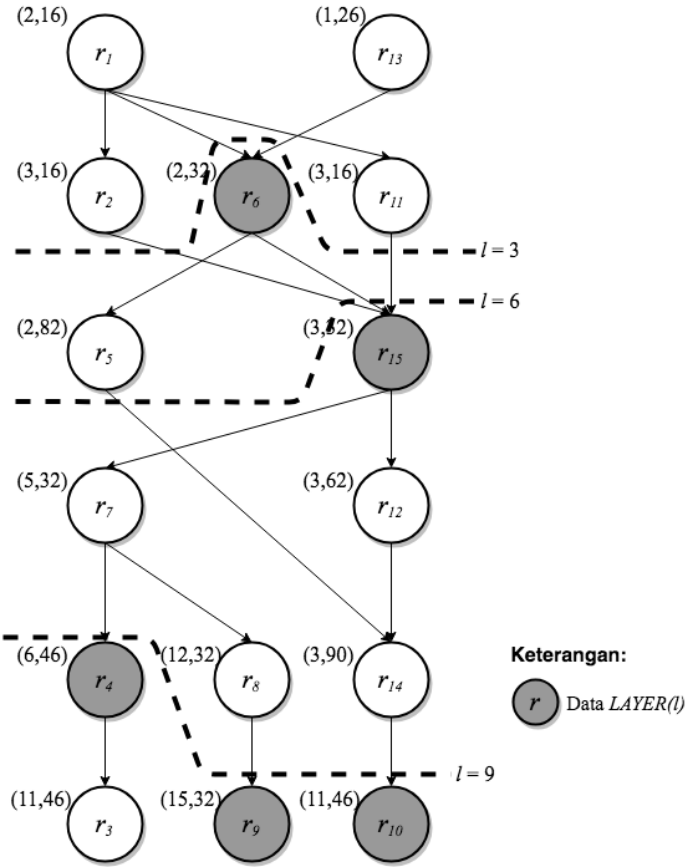
Tabel 3.2 Contoh hasil nilai *ancscore(r)*

r	***ancscore(r)***
r_1	1
r_2	2
r_3	10
r_4	8
r_5	4
r_6	3
r_7	7
r_8	8
r_9	9
r_{10}	9
r_{11}	2
r_{12}	7
r_{13}	1
r_{14}	9
r_{15}	6

Untuk mempercepat algoritma berbasis CDG, pada tahap *Preprocessing* akan dibentuk lagi lapisan *clean cut* (*LAYER*) sebesar *layerdist*. *LAYER(l)* adalah sekumpulan vertex *r* yang memiliki skor *ancestor(r)* $\geq l$ dan *r* tidak didominasi data r' dimana $r \in R$ dan $r' \in LAYER(l)$. Pada algoritma berbasis CDG, salah satu *LAYER* akan menjadi *root* bayangan untuk penelusuran DFS. Contoh hasil *LAYER* pada himpunan data R, dimana *layerdist* = 3, dapat dilihat pada Tabel 3.3 dan Gambar 3.5.

Tabel 3.3 Contoh hasil tabel *LAYER* dari himpunan data *R*, dengan *layerdist* sebesar 3

l	***LAYER(l)***
3	r_6
6	r_{15}
9	r_4, r_9, dan r_{10}



Gambar 3.5. Contoh hasil visualisasi *LAYER* dari himpunan data *R*, dengan *layerdist* sebesar 3

3.1.2. Metode Pengukuran Biaya Perubahan

Untuk mengukur toleransi pengguna terhadap perubahan nilai atribut pada *query* aslinya q_0, pada penelitian ini didefinisikan metode untuk mengukur biaya perubahan nilai pada atribut *query*. Dinyatakan himpunan nilai atribut pada query awal q_o dengan P_o, dan himpunan nilai atribut pada *refined query* dengan P'. Secara sederhana biaya perubahan nilai atribut ΔP adalah selisih antara nilai atribut baru $p' \in P$ dengan atribut pada query awal $p_o \in P$, atau dalam notasi matematika dinyatakan dengan $\Delta P = p' - p_o$. Permasalahan selanjutnya adalah pengguna pasti memiliki preferensi atas atribut mana yang sebaiknya diubah, maka dibutuhkan pembobotan w_i untuk setiap atributnya. Dengan beberapa pernyataan tersebut, dapat dirumuskan persamaan dasar pengukuran penalti sebagai berikut:

$$Penalty(q_o, q') = \sum_{i=1}^{d} w_i \, \Delta p_i \quad (3.1)$$

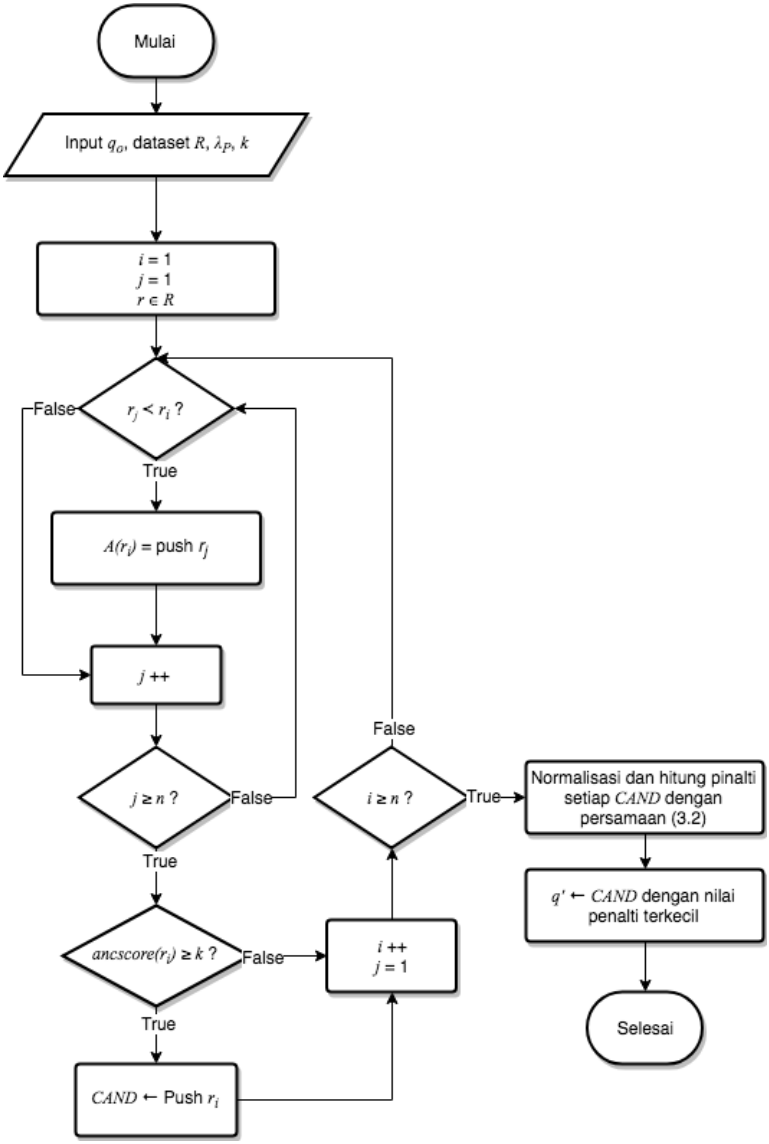
Nilai d pada persamaan (3.1) mengacu pada jumlah dimensi pada himpunan data R dengan P' adalah sebuah set berisi sekumpulan nilai atribut $P' = \{p'_1, p'_2, \dots, p'_d\}$ dari hasil *query refinement* q'. Pada himpunan data yang sebenarnya, persamaan tersebut tidak dapat diterapkan karena setiap dimensi dapat memiliki rentang nilai yang berbeda. Untuk itu, nilai p_o dan p' harus dinormalisasi terlebih dahulu dengan algoritma *Min-Max* sesuai pada persamaan (2.1). Maka, persamaan (3.1) dapat diturunkan sebagai berikut:

$$Penalty(q_o, q') = \sum_{i=1}^{d} w_i \left(\frac{p_{o_i} - p_{min}}{p_{max} - p_{min}} - \frac{p'_i - p_{min}}{p_{max} - p_{min}} \right) \quad (3.2)$$

3.1.3. Algoritma Berbasis CDG

Pada bagian ini dijelaskan desain metode dari algoritma untuk melakukan query refinement terhadap permasalahan *Why-Not on Reaching k Subscribers*. Secara garis besar algoritma berbasis CDG mengolah data yang telah dilakukan *indexing* berupa struktur data CDG dan dibentuk lapisan *clean cut LAYER* dengan masing-masing lapisan l berjarak *layerdist* untuk dapat mengembalikan *query refinement* q' dengan nilai penalti terkecil. Masukan pada algoritma berbasis CDG adalah himpunan data yang telah dipersiapkan pada tahap *Preprocessing* CDG, *query* awal q_o, ekspektasi jumlah subscribers k, dan koefisien bobot untuk perubahan setiap nilai atribut $\lambda_P = \{\lambda_{p_1}, \lambda_{p_2}, \ldots, \lambda_{p_d}\}$. Algoritma dimulai dengan mencari *LAYER(l)* dimana $l \leq k$ kemudian pilih nilai *l* terdekat dari *k*. Jadikan *LAYER(l)* sebagai *ROOT*, dan mulai penelusuran dengan algoritma DFS hingga menemukan sebuah set kandidat solusi $CAND = \{cand_1, cand_2, \ldots\}$ dengan $ancscore(cand_i) \geq k$. Ilustrasi alur algoritma berbasis CDG dapat dilihat pada Gambar 3.6.

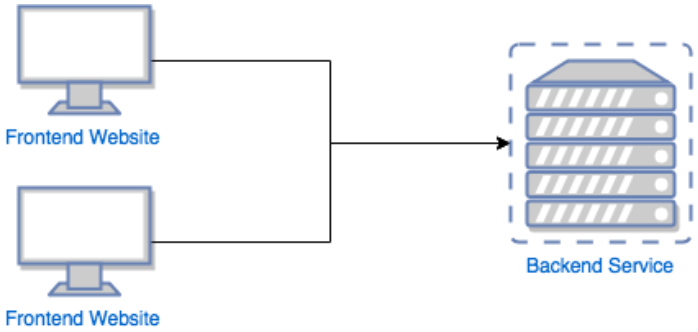
Dengan persamaan (3.2) dapat ditentukan solusi terbaik dari $CAND$ untuk dijadikan sebagai *query refinement* q' dengan membandingkan nilai penalti dari masing-masing $CAND$ dan pilih data yang memiliki nilai penalti terkecil.



Gambar 3.6 Ilustrasi Alur Algoritma Berbasis CDG

3.1.4. Desain Arsitektur Aplikasi

Algoritma yang telah dijabarkan pada sub-bab sebelumnya akan diaplikasikan sebagai perangkat lunak berbasis *web*. Dalam pengaplikasian algoritma, penelitian ini menggunakan arsitektur *client-server* yang dapat dilihat pada Gambar 3.7.



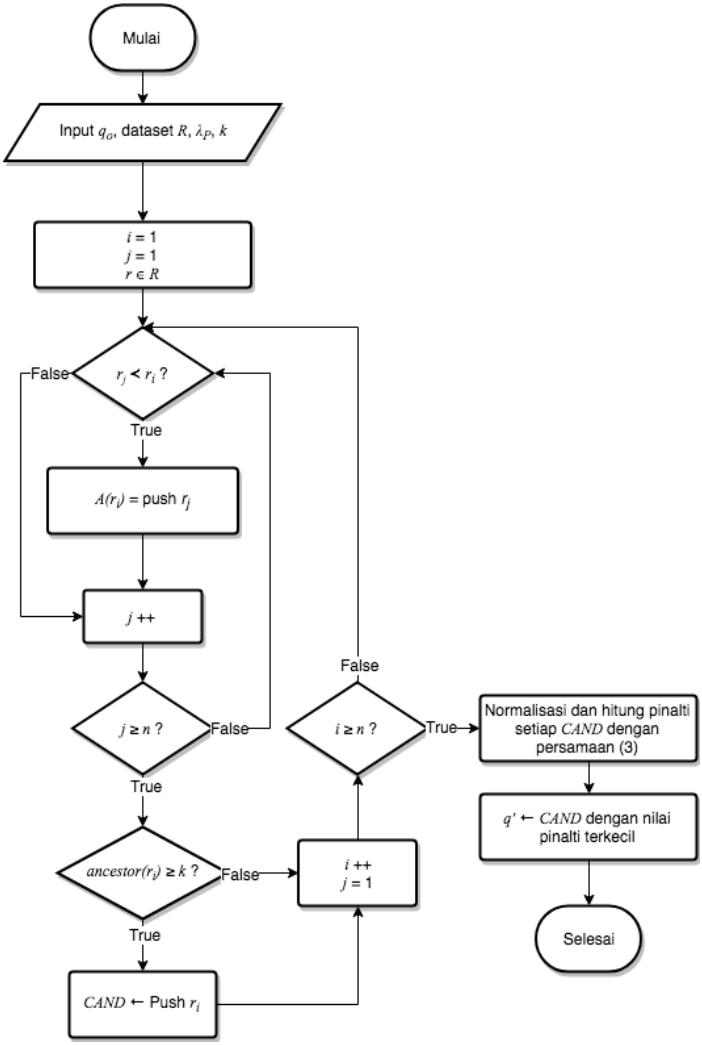
Gambar 3.7 Desain Arsitektur Aplikasi

Frontend Website berperan sebagai *client* untuk antarmuka pengguna. Pengguna melakukan masukan data pada *Frontend Website*, kemudian data masukan tersebut dikirim ke *Backend Service* sebagai *server* untuk kemudian diterima oleh Apache HTTP *Web Server*, diteruskan oleh PHP, kemudian barulah diolah oleh Python sesuai dengan algoritma yang telah dijabarkan pada sub-bab sebelumnya. Jika proses algoritma telah selesai, maka solusi akan dikembalikan oleh *Backend Service* ke Frontend *Website* untuk kemudian ditampilkan ke layar pengguna beserta visualisasi graf dari himpunan data dengan menggunakan pustaka Vis.JS.

3.2. Algoritma Pembanding (*Brute Force*)

Dalam penelitian ini dihasilkan juga algoritma *Brute Force* sebagai pembanding dari algoritma utama. Algoritma ini dirancang sesederhana mungkin yaitu dengan membandingkan seluruh data pada himpunan data R untuk mendapatkan kandidat solusi $CAND$. Untuk menyeleksi $CAND$ dan menemukan solusi terbaik sebagai

q' akan menggunakan metode yang sama dengan algoritma utama yaitu dengan mencari kandidat dengan skor penalti terkecil.



Gambar 3.8 Ilustrasi Alur Algoritma *Brute Force*

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan sebelumnya. Pada bagian implementasi ini juga akan dijelaskan mengenai fungsi-fungsi yang digunakan dalam program tugas akhir ini dan disertai dengan pseudocode masing-masing fungsi utama.

4.1. Implementasi *Preprocessing*

Algoritma *Preprocessing* diimplementasikan sesuai dengan rancangan yang telah dijelaskan pada bab perancangan. *Preprocessing* secara garis besar bertujuan untuk mengolah himpunan data R dengan jumlah data sebanyak n dan ukuran dimensi d menjadi CDG dengan tambahan *indexing* berupa $LAYER$. Terdapat dua tahap utama pada *Preprocessing*, yaitu tahap pembentukan CDG pada fungsi *BuildGraph* dan tahap *layer indexing* pada fungsi *LayerIndexing*.

4.1.1. Fungsi *BuildGraph*

Pada bagian ini akan dijelaskan pseudocode dari metode dalam membangun CDG. Fungsi ini menggunakan algoritma rekursif dan DFS dalam membentuk CDG. Fungsi ini akan mengolah variabel global himpunan data *VISITED* yang membantu untuk mencatat data mana saja yang telah ditelusuri dalam pembangunan CDG agar tidak ditelusuri berulang kali. Masukan pada fungsi ini adalah berupa data $node \in R$ sebagai penanda data yang sedang aktif dalam pembangunan CDG, sebuah set yang berisi himpunan data $S \subseteq R$ sebagai area pembangunan CDG, dan variabel *is_initial* sebagai penanda awal iterasi pembangunan CDG. Fungsi ini tidak memiliki keluaran karena mengolah himpunan data R yang didefinisikan secara global pada algoritma utama *Preprocessing* yang dapat dilihat pada gambar. Pseudocode untuk fungsi *BuildGraph* dapat dilihat pada Gambar 4.1.

Fungsi BuildGraph(*node, S, is_initial*)	
Variabel Global	*VISITED*
Masukan	*node, S, is_initial*
Keluaran	-

```
1.  if is_initial = TRUE then
2.      for all next nodes next ∈ sky(S) do
3.          BuildGraph(next, D(next), FALSE)
4.  else
5.      if node ∈ VISITED then return
6.      else
7.          VISITED ← push node
8.          C(node) ← sky(S)
9.          build relation node → c, ∀c ∈ C(node)
10.         for all next nodes next ∈ sky(D(node)) do
11.             BuildGraph(next, D(next), FALSE)
12. return
```

Gambar 4.1 Pseudocode Fungsi *BuildGraph*

4.1.2. Fungsi *LayerIndexing*

Pada bagian ini akan dijelaskan pseudocode dari metode dalam membuat lapisan $LAYER_l$. Fungsi ini menggunakan algoritma rekursif dan DFS dalam menelusuri CDG. Fungsi ini akan mengolah variabel global himpunan data *VISITED* yang membantu untuk mencatat *vertex* mana saja yang telah ditelusuri dalam pembuatan lapisan *LAYER* agar tidak ditelusuri berulang kali. Masukan pada fungsi ini adalah berupa himpunan data $V\_ROOT \subseteq R$ atau *virtual root* sebagai titik mulai penelusuran CDG, data $node \in R$ sebagai penanda data yang sedang aktif dalam penelusuran CDG, batas *clean cut layer l*, dan variabel *is_initial* sebagai penanda awal iterasi penelusuran CDG. Fungsi ini menghasilkan keluaran berupa himpunan data $LAYER_l$. Pseudocode untuk fungsi *BuildGraph* dapat dilihat pada Gambar 4.2.

Fungsi LayerIndexing(*V_ROOT*, *node*,*l*, *is_initial*)	
Variabel Global	*VISITED*
Masukan	*V_ROOT*, *node*, *l*, *is_initial*
Keluaran	$LAYER_l$

```
1.  RESULTS
2.  if is_initial = TRUE then
3.          NEXT ← V_ROOT
4.  else
5.          if node ∈ VISITED then return RESULTS
6.          else
7.                  VISITED ← push node
8.                  if ancscore(node) ≥ l then
9.                          RESULTS ← push node
10.                         return RESULTS
11.                 NEXT ← C(node)
12. for all next nodes next ∈ NEXT do
13.         RESULTS ← merge LayerIndexing(V_ROOT,
            next,l, FALSE)
14. return RESULTS
```

Gambar 4.2 Pseudocode Fungsi *LayerIndexing*

Setelah mendefinisikan dua fungsi utama, *BuildGraph* dan *LayerIndexing*, maka dapat dilanjutkan dengan implementasi algoritma *Preprocessing* secara utuh. *Preprocessing* menerima input berupa himpunan data *R* yang akan digunakan pada fungsi *BuildGraph* dan nilai jarak antar lapisan *LAYER layerdist* yang akan digunakan pada fungsi *LayerIndexing*. Pseudocode *Preprocessing* dapat dilihat pada Gambar 4.3.

Algoritma *Preprocessing*	
Masukan	*R, layerdist*
Keluaran	*CDG, LAYER*

1. $ROOT \leftarrow \overline{sky}(S)$
2. BuildGraph(*NONE*, *R*, *TRUE*)
3. $l \leftarrow layerdist$
4. $first\_itteration \leftarrow TRUE$
5. **while** $l < n$ **do**
6. **if** $first\_itteration = TRUE$ **then** $V\_ROOT \leftarrow ROOT$
7. $LAYER_l \leftarrow$ LayerIndexing(*V_ROOT*, *NONE*,*l*, *TRUE*)
8. V_ROOT $\leftarrow LAYER_l$
9. $l \leftarrow$ l + *layerdist*

Gambar 4.3 Pseudocode Algoritma *Preprocessing*

4.2. Implementasi Algoritma Berbasis CDG

Algoritma berbasis CDG diimplementasikan sesuai dengan rancangan yang telah dijelaskan pada bab desain dan perancangan. Algoritma berbasis CDG secara garis besar adalah bertujuan untuk menghasilkan q' agar dapat menjawab permasalahan *Why-Not on Reaching k Subscribers*. Untuk menunjang algoritma berbasis CDG, maka dibutuhkan sebuah fungsi rekursif untuk menelusuri CDG dengan algoritma DFS yang dinamakan *FindCandidate*.

Pada bagian ini akan dijelaskan pseudocode dari metode dalam mencari kandidat *refined query CAND*. Fungsi ini menggunakan algoritma rekursif dan DFS dalam menelusuri CDG. Fungsi ini akan mengolah variabel global himpunan data *VISITED* yang membantu untuk mencatat *vertex* mana saja yang telah ditelusuri dalam pencarian *CAND* agar tidak ditelusuri berulang kali. Masukan pada fungsi ini adalah berupa himpunan data $V\_ROOT \subseteq LAYER$ atau *virtual root* sebagai titik mulai penelusuran CDG, data $node \in R$ sebagai penanda data yang sedang aktif dalam penelusuran CDG, ekspektasi jumlah

subscribers k, dan variabel *is_initial* sebagai penanda awal iterasi penelusuran CDG. Fungsi ini menghasilkan keluaran berupa himpunan data *CAND*. Pseudocode untuk fungsi *FindCandidate* dapat dilihat pada Gambar 4.4.

Fungsi FindCandidate(*V_ROOT*, *node*,*k*, *is_initial*)	
Variabel Global	*VISITED*
Masukan	*V_ROOT*, *node*, *k*, *is_initial*
Keluaran	$LAYER_l$

```
1.  RESULTS
2.  if is_initial = TRUE then
3.          NEXT ← V_ROOT
4.  else
5.          if node ∈ VISITED then return RESULTS
6.          else
7.                  VISITED ← push node
8.                  if ancscore(node) ≥ k then
9.                          RESULTS ← push node
10.                         return RESULTS
11.                 NEXT ← C(node)
12. for all next nodes next ∈ NEXT do
13.         RESULTS ← merge LayerIndexing(V_ROOT,
            next,k, FALSE)
14. return RESULTS
```

Gambar 4.4 Pseudocode Fungsi *FindCandidate*

Setelah mendefinisikan fungsi *FindCandidate*, maka dapat dilanjutkan dengan implementasi algoritma berbasis CDG secara utuh. Algoritma berbasis CDG menerima masukan berupa himpunan data *R* yang telah dibuat dalam bentuk CDG dan *LAYER* yang akan digunakan pada fungsi *FindCandidate*, jumlah ekspektasi *subscribers k*, nilai koefisien bobot untuk toleransi

perubahan tiap atribut *w*, dan kueri awal q_o. Pseudocode algoritma berbasis CDG dapat dilihat pada Gambar 4.5.

Algoritma Algoritma Berbasis CDG	
Masukan	*R, CDG, LAYER, k, w,* q_o
Keluaran	q'

```
1.  l_closest ← cari nilai l terdekat dari k, l ≤ k
2.  V_ROOT ← LAYER_l_closest
3.  CAND ← FindCandidate(V_ROOT, NONE,k, TRUE)
4.  best_penalty ← ∞
5.  best_cand ← NONE
6.  for all candidate c ∈ CAND do
7.          if Penalty(c) < best_penalty then
8.                  best_penalty ← Penalty(q_o, c)
9.                  best_cand ← best_cand
10. return q' = best_cand
```

Gambar 4.5 Pseudocode Algoritma Algoritma Berbasis CDG

4.3. 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 berbasis CDG, namun dengan metode yang lebih sederhana dan tanpa *Preprocessing* atau *indexing*. Untuk masukan pada algoritma *Brute Force* adalah berupa himpunan data *R*, *query* awal q_o, jumlah ekspektasi subscribers k dan nilai koefisien bobot untuk toleransi perubahan tiap atribut *w*. Algoritma ini menghasilkan keluaran yang sama dengan algoritma berbasis CDG, yaitu *refined query* q'. Pseudocode algoritma berbasis CDG dapat dilihat pada Gambar 4.6.

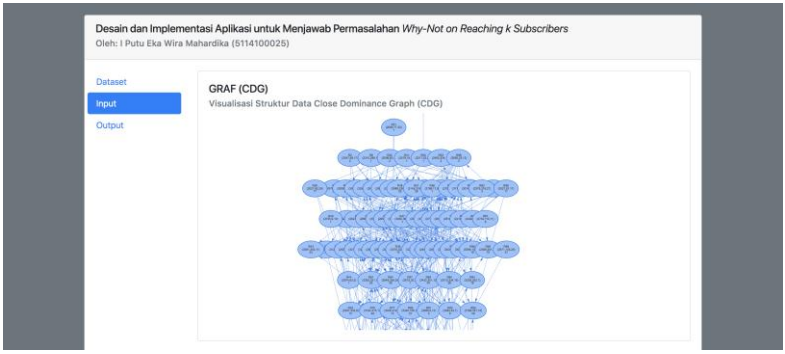
Algoritma *Brute Force*	
Masukan	R, q_o, k, w
Keluaran	q'

1. *best_penalty* ← ∞
2. *best_cand* ← *NONE*
3. **for all** candidate $c \in R$ **do**
4. **for all** data $r \in R$ **do**
5. **if** $r \prec c$ **then** $A(c)$ ← push r
6. **if** *Penalty*(c) < best_penalty ∧ *ancscore*(c) ≥ k **then**
7. *best_penalty* ← *Penalty*(q_o, c)
8. *best_cand* ← *best_cand*
9. **return** q' = *best_cand*

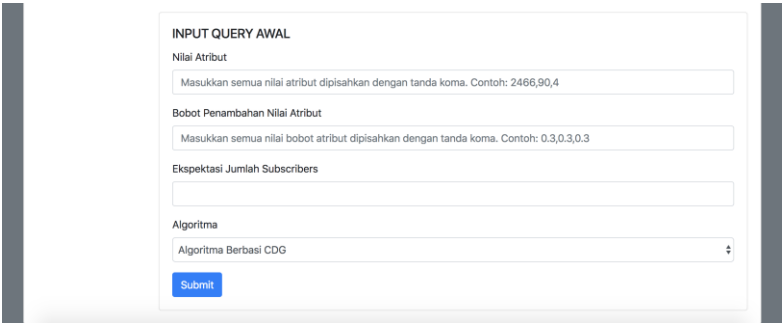
Gambar 4.6 Pseudocode Algoritma *Brute Force*

4.4. Implementasi Antarmuka Pengguna

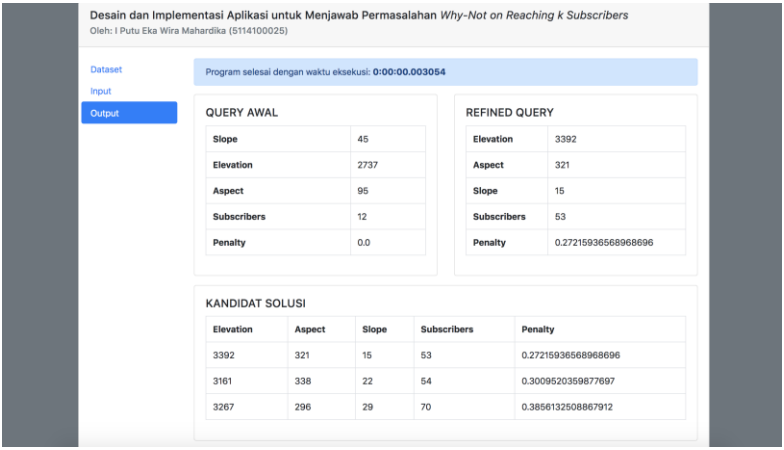
Pada bagian ini, ditampilkan beberapa cuplikan layar dari hasil implementasi visualisasi antarmuka pengguna.



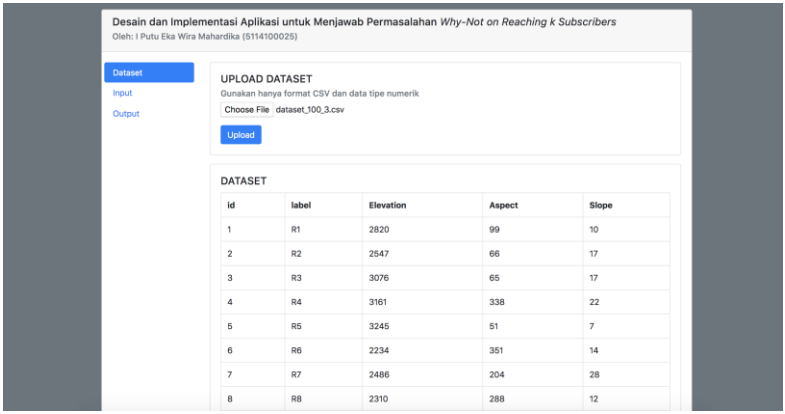
Gambar 4.7 Implementasi Visualisaasi CDG



Gambar 4.8 Implementasi *Form* Masukan *Query* Awal



Gambar 4.9 Implementasi Hasil *Refined Query*



Gambar 4.10 Implementasi Pratinjau dan *Form Upload* Himpunan Data

[Halaman ini sengaja dikosongkan]

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 lingkungan uji coba, data uji coba, dan skenario uji coba yang meliputi uji fungsionalitas dan uji performa serta analisa setiap pengujian.

5.1. Lingkungan Pengujian

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi metode *query refinement* untuk menjawab *Why-Not On Reaching k Subscribers* pada penelitian ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan pada Tabel 5.1 dan Tabel 5.2.

Tabel 5.1 Lingkungan Pengujian Perangkat Keras

Perangkat Keras	
Prosesor	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
Memori	64 GB
Arsitektur Prosesor	64-bit

Tabel 5.2 Lingkungan Pengujian Perangkat Lunak

Perangkat Lunak	
Sistem Operasi	Ubuntu 16.04.4 LTS
Perangkat Pengembang	Python 2.7

5.2. Data Uji Coba

Tahap uji coba pada penelitian ini menggunakan 3 jenis data, yaitu data *Forest Covertype* (FC), data independen (IND), dan data anti *corelated* (ANT). Masing-masing jenis data memiliki variasi pada jumlah data *n* dan jumlah dimensi *d*.

5.2.1. 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) [12].

Himpunan data ini terdiri dari 581.012 data dan memiliki 54 dimensi atribut. Tidak ditemukan nilai yang hilang atau anomali pada himpunan data ini. Seluruh atribut bertipe data numerik dengan rincian dapat dilihat pada Tabel 5.3.

Tabel 5.3 Rincian Atribut pada Himpunan Data *Forest Covertype*

Nama Kolom	**Jenis Data**	**Jumlah Kolom**
Elevation	kuantitatif	1
Aspect	kuantitatif	1
Slope	kuantitatif	1
Horizontal_Distance_To_Hydrology	kuantitatif	1
Vertical_Distance_To_Hydrology	kuantitatif	1
Horizontal_Distance_To_Roadways	kuantitatif	1
Hillshade_9am	index 0-255	1
Hillshade_Noon	index 0-255	1
Hillshade_3pm	index 0-255	1
Horizontal_Distance_To_Fire_Points	kuantitatif	1
Wilderness_Area	kualitatif	4
Soil_Type	kualitatif	40
Cover_Type	kualitatif	1

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.

5.2.2. 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 – 1000.

5.2.3. Data *Anti Correlated* (ANT)

Data *anti correlated* (ANT) adalah himpunan data sintetis yang dibuat pada penelitian ini. Data *anti correlated* memiliki persebaran nilai atribut yang saling bertolak belakang antara satu antribut dengan atribut lainnya, yang artinya sebuah data memiliki nilai yang sangat baik pada salah satu atributnya namun sangat buruk pada atribut lainnya. Penggunaan data ini bertujuan untuk menguji performa algoritma jika berhadapan dengan data yang nilai atributnya saling bertolak belakang dan memiliki relasi dominasi antar data paling rendah dibandingkan dengan jenis data lainnya. Hal ini dikarenakan seluruh data akan menjadi titik *skyline* pada himpunan data ini. Rentang nilai yang digunakan untuk setiap atribut adalah 1 – 1000.

5.3. Skenario dan Evaluasi Pengujian

Uji coba ini dilakukan untuk menguji apakah program yang telah diimplementasikan dapat berjalan dengan sebagaimana mestinya. Uji coba akan dibagi menjadi dua jenis, yaitu uji coba fungsionalitas dan uji coba performa.

5.3.1. Uji Coba Fungsionalitas

Uji coba fungsionalitas berfokus pada pengujian apakah aplikasi dapat digunakan sesuai dengan kebutuhan fungsional. Kebutuhan fungsional didefinisikan pada Tabel 5.4.

Tabel 5.4 Daftar Kebutuhan Fungsional

Nomor	Deskripsi Kebutuhan
F01	Pengguna dapat mengunggah himpunan data untuk dilakukan *Preprocessing* oleh aplikasi
F02	Aplikasi dapat menampilkan struktur data CDG dalam visualisasi graf
F03	Pengguna dapat memasukkan *query* awal, ekspektasi subscribers, dan bobot atribut untuk dilakukan proses *query refinement* oleh aplikasi
F04	Pengguna dapat memilih antara algoritma berbasis CDG atau algoritma *brute force* dalam proses *query refinement*.
F05	Aplikasi dapat menampilkan hasil *query refinement* terbaik
F06	Aplikasi dapat menampilkan kandidat hasil *query refinement*
F07	Aplikasi dapat menampilkan waktu eksekusi dalam proses *query refinement*

5.3.2. Uji Coba Performa

Skenario uji coba performa berfokus pada pengujian seberapa baik program dapat memberikan solusi atas masukan yang diberikan. Secara garis besar kualitas performa akan dibandingkan dengan durasi waktu eksekusi dan jumlah penggunaan memori. Sebagai perbandingan, algoritma berbasis CDG akan dibandingkan dengan algoritma *brute force*. Skenario uji coba diberlakukan ke semua jenis data (*Forest Covertype*, independen, dan *anti correlated*). Berikut adalah beberapa skenario pengujian:

1. Perbandingan jumlah data *n* dengan waktu eksekusi dan jumlah penggunaan memori. Detail skenario dapat dilihat pada Tabel 5.5.

Tabel 5.5 Skenario Perbandingan Jumlah Data dengan Waktu Eksekusi dan Penggunaan Memori

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Ekspektasi *Subscribers* (*k*)
A01	10.000	3	5.000
A02	30.000	3	5.000
A03	50.000	3	5.000
A04	70.000	3	5.000
A05	100.000	3	5.000

2. Perbandingan jumlah dimensi *d* dengan waktu eksekusi dan jumlah penggunaan memori. Detail skenario dapat dilihat pada Tabel 5.6.

Tabel 5.6 Skenario Perbandingan Jumlah Dimensi Terhadap Waktu Eksekusi dan Penggunaan Memori

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Ekspektasi *Subscribers* (*k*)
B01	30.000	2	5.000
B02	30.000	3	5.000
B03	30.000	5	5.000
B04	30.000	7	5.000
B05	30.000	10	5.000

3. Perbandingan ekspektasi jumlah *subscribers k* dengan waktu eksekusi dan jumlah penggunaan memori. Detail skenario dapat dilihat pada Tabel 5.7.

Tabel 5.7 Skenario Perbandingan Ekspektasi Jumlah *Subscribers* Terhadap Waktu Eksekusi dan Penggunaan Memori

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Ekspektasi *Subscribers* (*k*)
C01	30.000	3	1.000
C02	30.000	3	2.000
C03	30.000	3	5.000
C04	30.000	3	10.000
C05	30.000	3	15.000

5.4. Analisis Hasil Uji Coba

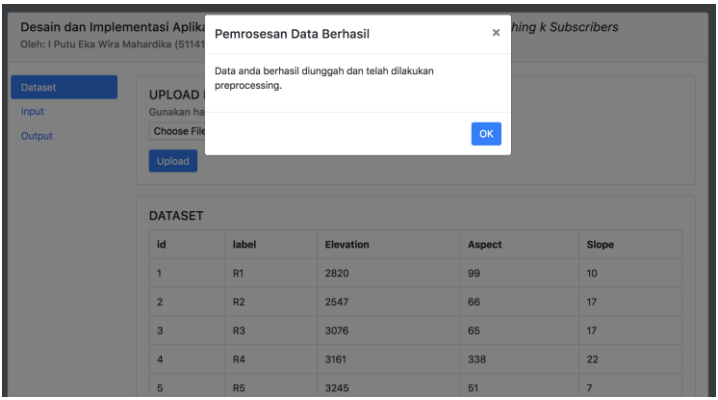
Bagian ini menjelaskan hasil uji coba beserta analisanya. Uji coba yang dilakukan sesuai dengan skenario uji coba yang telah didefinisikan sebelumnya. Hasil uji coba dibagi menjadi dua bagian, yaitu hasil uji coba fungsionalitas dan hasil uji coba performa.

5.4.1. Hasil Uji Coba Fungsionalitas

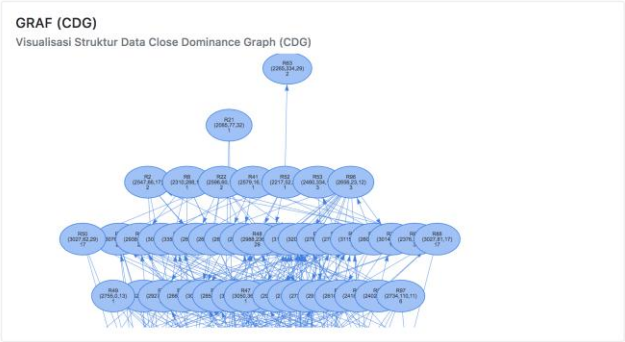
Uji coba dilakukan sesuai dengan skenario daftar kebutuhan fungsional pada Tabel 5.4. Hasil uji coba fungsionalitas dapat dilihat pada Tabel 5.8. Dari hasil uji coba, aplikasi dapat memenuhi semua kebutuhan fungsional.

Tabel 5.8 Hasil Uji Coba Fungsionalitas

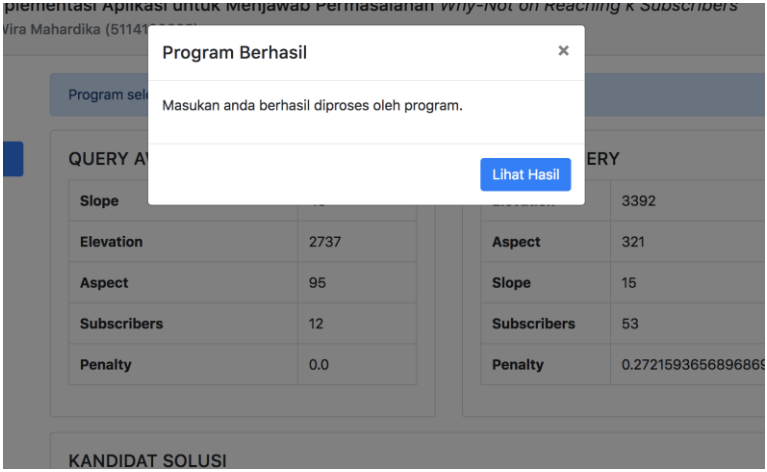
Nomor	Keterangan
F01	Berhasil
F02	Berhasil
F03	Berhasil
F04	Berhasil
F05	Berhasil
F06	Berhasil
F07	Berhasil



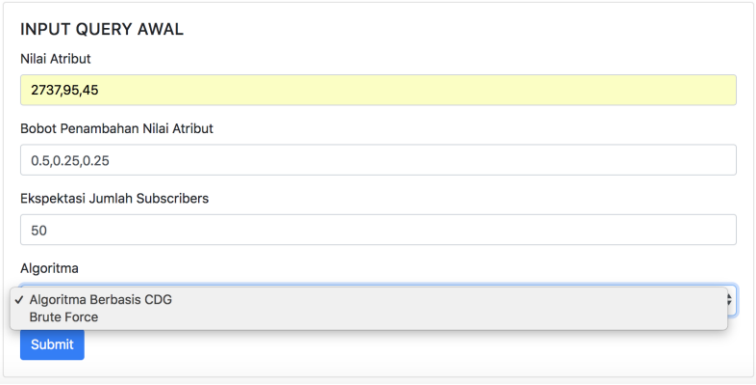
Gambar 5.1 Hasil Uji Coba F01



Gambar 5.2 Hasil Uji Coba F02



Gambar 5.3 Hasil Uji Coba F03



Gambar 5.4 Hasil Uji Coba F04

QUERY AWAL

Slope	45
Elevation	2737
Aspect	95
Subscribers	12
Penalty	0.0

REFINED QUERY

Elevation	3392
Aspect	321
Slope	15
Subscribers	53
Penalty	0.27215936568968696

Gambar 5.5 Hasil Uji Coba F05

KANDIDAT SOLUSI

Elevation	Aspect	Slope	Subscribers	Penalty
3392	321	15	53	0.27215936568968696
3161	338	22	54	0.3009520359877697
3267	296	29	70	0.3856132508867912

Gambar 5.6 Hasil Uji Coba F06

Program selesai dengan waktu eksekusi: **0:00:00.003054**

Gambar 5.7 Hasil Uji Coba F07

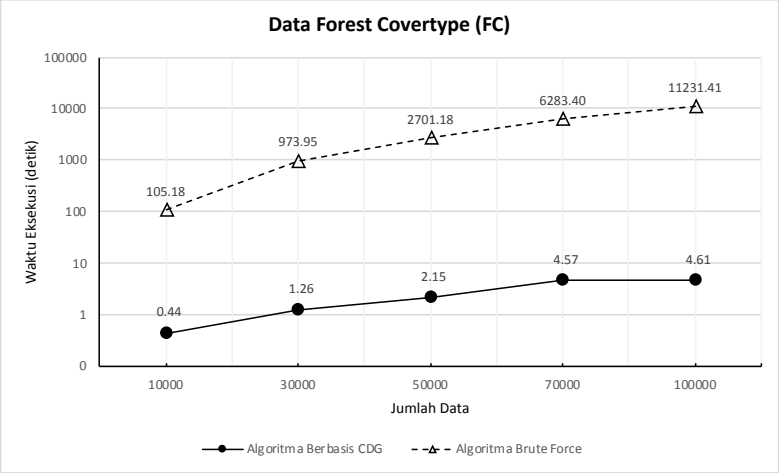
5.4.2. Hasil Uji Coba Performa

Pada Gambar 5.8, Gambar 5.9, dan Gambar 5.10 ditampilkan grafik hasil uji coba perbandingan waktu eksekusi dengan jumlah data pada semua jenis data untuk mengukur performa masing-masing algoritma. Uji coba dilakukan sesuai dengan skenario pada Tabel 5.5, dimana setiap kasus uji coba menggunakan himpunan data dengan jumlah data yang bervariasi, sementara dimensi dan ekspektasi *subscribers* bersifat tetap ($d = 3$, $k = 5000$).

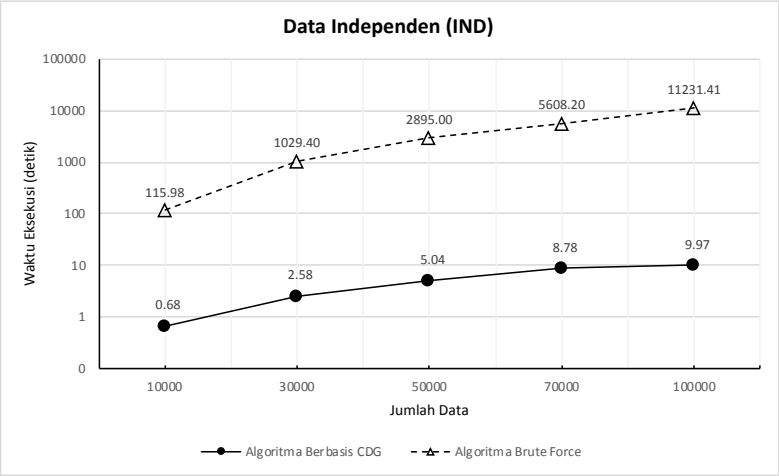
Grafik pada Gambar 5.8 dan Gambar 5.9 menjelaskan bahwa algoritma berbasis CDG memiliki waktu eksekusi lebih

baik dengan selisih yang signifikan di setiap kasus uji coba. Hal ini membuktikan struktur data CDG dan *layer indexing* sangat membantu algoritma berbasis CDG dalam hal menangani jumlah data yang bervariasi pada himpunan data FC dan IND.

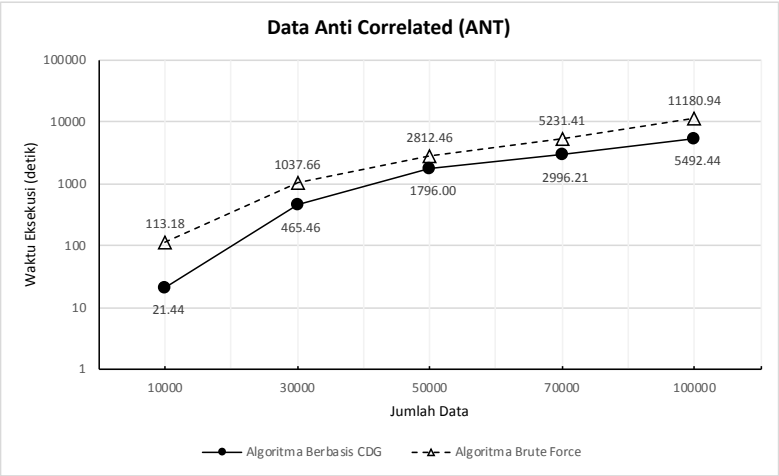
Pada Gambar 5.10 dapat dilihat algoritma berbasis CDG masih memiliki waktu eksekusi lebih baik, namun tidak signifikan. Hal ini dikarenakan pada himpunan data ANT, *close dominance relation* antar data jauh berkurang sehingga performa waktu eksekusi algoritma berbasis CDG mendekati algoritma *brute force*.



Gambar 5.8 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Data Pada FC



Gambar 5.9 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Data Pada IND



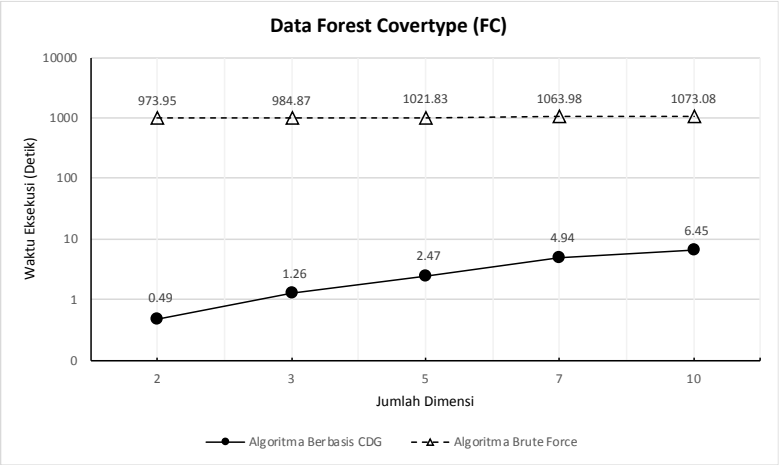
Gambar 5.10 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Data Pada ANT

Pada Gambar 5.11, Gambar 5.12, dan Gambar 5.13 ditampilkan grafik hasil uji coba perbandingan waktu eksekusi dengan jumlah dimensi pada semua jenis data untuk mengukur performa masing-masing algoritma. Uji coba dilakukan sesuai dengan skenario pada Tabel 5.6, dimana setiap kasus uji coba menggunakan himpunan data dengan jumlah dimensi yang bervariasi, sementara jumlah data dan ekspektasi *subscribers* bersifat tetap (n = 30000, k = 5000).

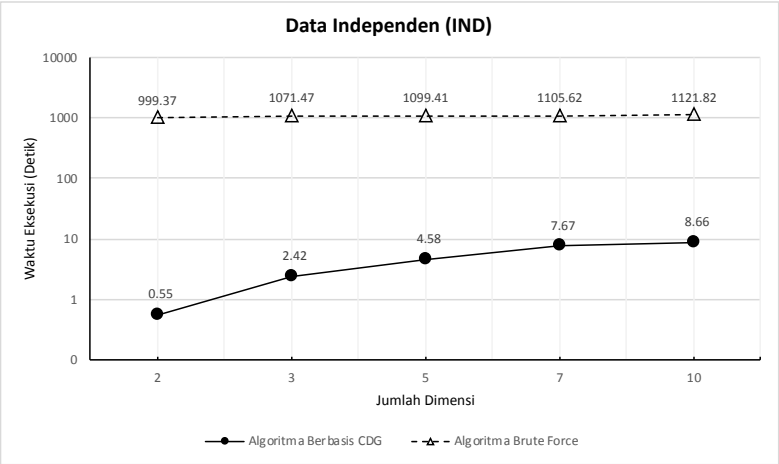
Secara keseluruhan, algoritma *brute force* memiliki waktu eksekusi yang cenderung konstan. Hal tersebut disebabkan oleh komparasi satu per satu yang tidak menimbulkan perubahan yang signifikan terhadap jumlah dimensi, selama jumlah data tetap.

Pada Gambar 5.11 dan Gambar 5.12 membuktikan bahwa algoritma berbasis CDG masih lebih unggul dengan selisih waktu eksekusi yang signifikan dibandingkan dengan algoritma *brute force*. Hal ini membuktikan struktur data CDG dan *layer indexing* sangat membantu algoritma berbasis CDG dalam hal menangani jumlah dimensi yang bervariasi pada himpunan data FC dan IND.

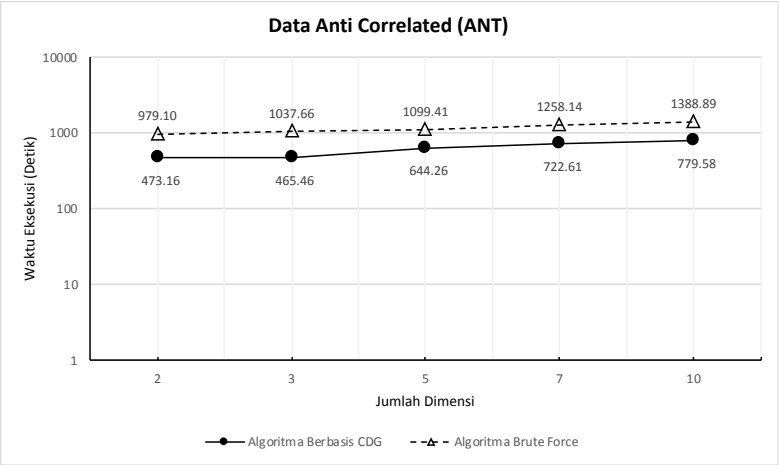
Sama seperti kasus uji coba sebelumnya, pada Gambar 5.13 dapat dilihat algoritma berbasis CDG masih memiliki waktu eksekusi lebih baik, namun tidak signifikan. Hal ini dikarenakan pada himpunan data ANT, *close dominance relation* antar data jauh berkurang sehingga performa waktu eksekusi algoritma berbasis CDG mendekati algoritma *brute force*.



Gambar 5.11 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Dimensi Pada FC



Gambar 5.12 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Dimensi Pada IND



Gambar 5.13 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Dimensi Pada ANT

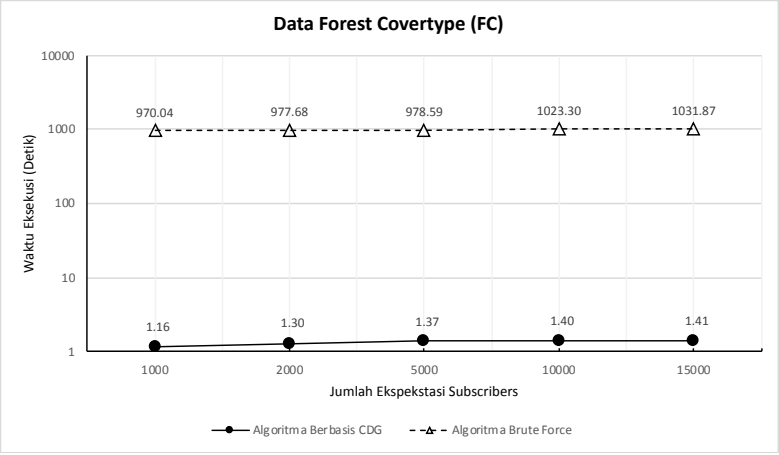
Pada Gambar 5.14, Gambar 5.15, dan Gambar 5.16 ditampilkan grafik hasil uji coba perbandingan waktu eksekusi dengan jumlah ekspektasi *subscribers* pada semua jenis data untuk mengukur performa masing-masing algoritma. Uji coba dilakukan sesuai dengan skenario pada Tabel 5.7, dimana setiap kasus uji coba menggunakan himpunan data dengan jumlah ekspektasi *subscribers* yang bervariasi, sementara jumlah data dan dimensi bersifat tetap (n = 30000, d = 3).

Secara keseluruhan baik algoritma berbasis CDG maupun algoritma *brute force* cenderung memiliki waktu eksekusi yang konstan. Hal ini membuktikan bahwa jumlah ekspektasi *subscribers* tidak memiliki pengaruh yang signifikan terhadap waktu eksekusi.

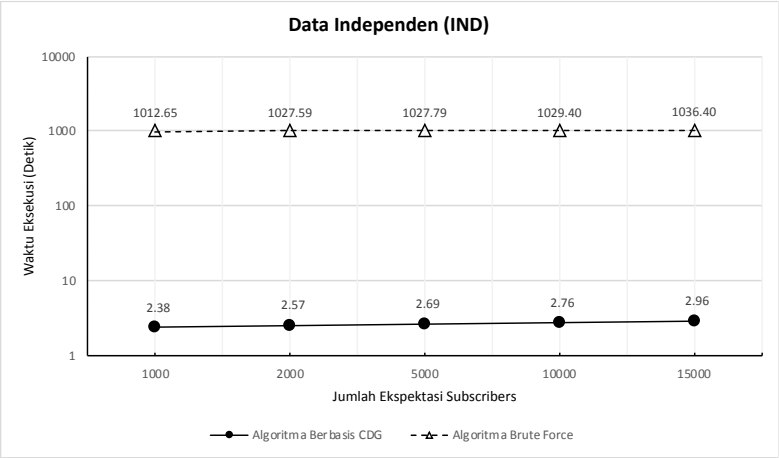
Pada Gambar 5.14 dan Gambar 5.15 membuktikan bahwa algoritma berbasis CDG masih lebih unggul dengan selisih waktu eksekusi yang signifikan dibandingkan dengan algoritma *brute force*. Hal ini membuktikan struktur data CDG dan *layer indexing*

sangat membantu algoritma berbasis CDG dalam hal menangani ekspektasi *subscribers* yang bervariasi pada himpunan data FC dan IND.

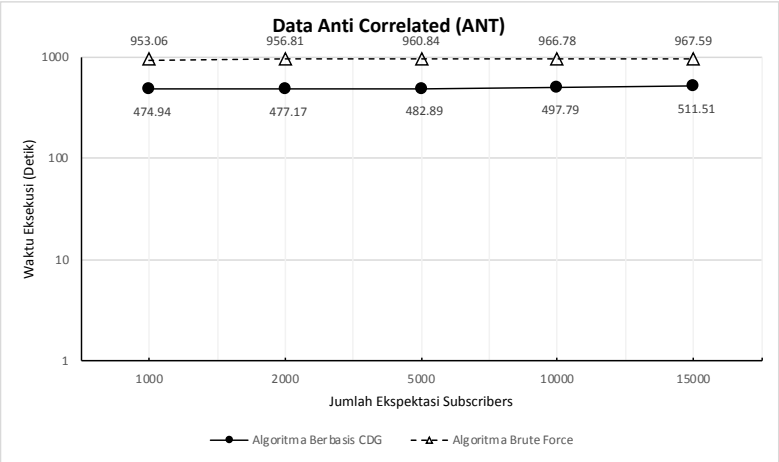
Sama seperti kasus uji coba sebelumnya, pada Gambar 5.16 dapat dilihat algoritma berbasis CDG masih memiliki waktu eksekusi lebih baik, namun tidak signifikan. Hal ini dikarenakan pada himpunan data ANT, *close dominance relation* antar data jauh berkurang sehingga performa waktu eksekusi algoritma berbasis CDG mendekati algoritma *brute force*.



Gambar 5.14 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Ekspektasi *Subscribers* Pada FC



Gambar 5.15 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Ekspektasi *Subscribers* Pada IND



Gambar 5.16 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Ekspektasi *Subscribers* Pada ANT

Hasil uji coba penggunaan memori dapat dilihat pada Tabel 5.9, Tabel 5.10, dan Tabel 5.11. Hasil untuk algoritma berbasis CDG dilambangkan dengan CDG dan algoritma *brute force* dilambangkan dengan BF. Skenario yang digunakan sama dengan pengujian waktu eksekusi.

Secara keseluruhan algoritma berbasis CDG memiliki konsumsi memori yang cenderung lebih tinggi daripada algoritma *brute force*. Hal ini dikarenakan himpunan data yang telah dilakukan *Preprocessing* akan mengandung informasi yang lebih banyak daripada himpunan data tanpa modifikasi yang digunakan di algoritma brute force. Konsumsi memori untuk algoritma berbasis CDG memiliki karakteristik hasil pengujian yang berbeda pada setiap skenario.

Pada skenario jumlah data, dapat dilihat pada Tabel 5.9 bahwa konsumsi memori terus meningkat seiring meningkatnya jumlah data. Hal ini membuktikan bahwa jumlah data mempengaruhi konsumsi memori.

Pada skenario jumlah dimensi, dapat dilihat pada tabel bahwa konsumsi memori pada jenis data FC dan IND cenderung meningkat hingga pada jumlah dimensi 5. Pada jumlah dimensi 7 dan 10 konsumsi memori mulai menurun. Hal ini disebabkan karena jumlah dimensi yang semakin banyak akan mengurangi *close dominance relation* antar data, sehingga dengan jumlah data sebanyak 3000, *close dominance relation* mulai menurun pada jumlah dimensi 7. Dengan berkurangnya *close dominance relation*, maka informasi yang disimpan pada struktur data CDG juga berkurang sehingga penggunaan memori menurun. Sedangkan pada jenis data ANT penggunaan memori cenderung terus meningkat, namun dengan konsumsi yang jauh lebih rendah disbanding jenis data lainnya. Hal ini dikarenakan jumlah close dominance relation pada jenis data ANT memang sudah sangat sedikit, sehingga penggunaan memori jauh lebih kecil sejak awal dan peningkatannya hanya disebabkan oleh penambahan informai dimensi.

Pada skenario jumlah ekspektasi subscribers sama seperti pengujian waktu eksekusi, penggunaan memori cenderung konstan. Hal ini membuktikan bahwa jumlah ekspektasi *subscribers* tidak mempengaruhi performa algoritma dalam hal penggunaan memori.

Tabel 5.9 Perbandingan Penggunaan Memori dengan Jumlah Data

n	***d* = 3, *k* = 5000**					
	FC		**IND**		**ANT**	
	CDG	**BF**	**CDG**	**BF**	**CDG**	**BF**
10K	108.99 MB	25.70 MB	165.52 MB	25.94 MB	38.58 MB	26.14 MB
30K	305.04 MB	53.58 MB	555.72 MB	53.79 MB	110.18 MB	54.75 MB
50K	488.54 MB	78.99 MB	963.18 MB	80.55 MB	199.40 MB	81.35 MB
70K	663.69 MB	104.73 MB	1386.16 MB	106.31 MB	291.24 MB	102.90 MB
100K	917.81 MB	146.12 MB	2167.53 MB	124.39 MB	437.35 MB	150.40 MB

Tabel 5.10 Perbandingan Penggunaan Memori dengan Jumlah Dimensi

d	***n* = 30000, *k* = 5000**					
	FC		**IND**		**ANT**	
	CDG	**BF**	**CDG**	**BF**	**CDG**	**BF**
2	139.49 MB	53.60 MB	155.33 MB	53.79 MB	106.57 MB	54.07 MB
3	305.04 MB	53.58 MB	555.72 MB	54.51 MB	110.18 MB	54.75 MB
5	1340.35 MB	53.92 MB	1916.18 MB	55.41 MB	135.61 MB	56.20 MB
7	1176.21 MB	54.75 MB	1854.05 MB	56.68 MB	150.96 MB	57.72 MB
10	544.63 MB	55.50 MB	585.81 MB	57.99 MB	173.40 MB	59.94 MB

Tabel 5.11 Perbandingan Penggunaan Memori dengan Jumlah Ekspektasi *Subscribers*

k	***n* = 30000, *d* = 3**					
	FC		**IND**		**ANT**	
	CDG	**BF**	**CDG**	**BF**	**CDG**	**BF**
1K	110.23 MB	53.61 MB	555.61 MB	54.43 MB	110.23 MB	54.68 MB
2K	110.17 MB	53.69 MB	555.56 MB	54.43 MB	110.17 MB	54.75 MB
5K	110.21 MB	53.59 MB	555.83 MB	54.41 MB	110.21 MB	54.89 MB
10K	110.18 MB	53.54 MB	555.63 MB	54.44 MB	110.18 MB	54.75 MB
15K	110.22 MB	53.56 MB	555.61 MB	54.52 MB	110.22 MB	54.79 MB

Dari hasil uji coba waktu eksekusi dan penggunaan memori dapat ditarik beberapa analisa dan kesimpulan. Pada algoritma berbasis CDG, semakin besar penggunaan memori maka waktu eksekusi menjadi lebih singkat. Hal ini disebabkan oleh tahap preprocessing, dimana struktur data yang dibuat mengandung semakin banyak informasi relasi antar data mengakibatkan penggunaan memori semakin meningkat. Semakin banyak informasi yang disimpan pada struktur data, maka waktu eksekusi menjadi semakin cepat karena informasi relasi antar data pada struktur data CDG akan memudahkan algoritma berbasis CDG dalam mencari kandidat solusi tanpa perlu menelusuri semua data. Hal ini juga berlaku sebaliknya, semakin sedikit penggunaan memori maka waktu eksekusi menjadi lebih lama.

[Halaman ini sengaja dikosongkan]

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. Proses *indexing* dilakukan dengan struktur data graf (CDG) untuk membantu proses *query refinement* sehingga tidak dibutuhkan untuk melakukan iterasi ke seluruh himpunan data. Proses pencarian kandidat *refined query* cukup dengan menelusuri CDG dan berhenti pada *vertex* tertentu karena *vertex* selanjutnya pasti juga merupakan kandidat solusi namun dengan nilai penalti perubahan lebih tinggi. Untuk memaksimalkan proses *indexing*, selanjutnya dilakukan pembuatan lapisan *clean cut* untuk membantu proses *query refinement* mengakibatkan proses penelusuran CDG lebih efisien. Dengan memotong titik mulai yang seharusnya pada bagian *root* CDG, dengan menggunakan lapisan *clean cut* titik mulai penelusuran dimulai dari lapisan *clean cut* terdekat dengan *vertex* kandidat solusi.
2. Algoritma berbasis CDG melakukan perbaikan query agar mencapai ekspektasi pengguna (*query refinement*) dengan menelusuri struktur data CDG hingga menemukan kandidat solusi yang memiliki nilai *ancscore* lebih dari atau sama dengan jumlah ekspektasi *subscribers*. Selanjutnya kandidat solusi diseleksi kembali dengan diurutkan sesuai dengan nilai penalti masing-masing kandidat dari yang terkecil

hingga terbesar. Algoritma mengembalikan kandidat dengan nilai penalti terkecil sebagai hasil *query refinement* terbaik.

3. Persamaan 3.2 dapat digunakan untuk menentukan nilai penalti dari *refined query*. Semakin kecil nilai penalti suatu *refined query*, maka *refined query* tersebut memiliki kualitas yang lebih baik dan berlaku juga sebaliknya. Persamaan tersebut tetap berlaku walaupun rentang nilai atribut pada data berbeda-beda karena persamaan tersebut sudah termasuk proses menormalisasi rentang nilai atribut agar seimbang.

6.2. Saran

Saran yang diberikan untuk pengembangan penelitian ini adalah:

1. Menggabungkan dengan algoritma *skyline query* yang tepat agar proses *indexing* dapat diselesaikan lebih cepat.
2. Penelitian algoritma berbasis CDG dapat dikembangkan lagi dengan metode pencarian BFS untuk dapat dibandingkan dengan metode pada penelitian ini.
3. Penelitian dapat dikembangkan dengan kondisi himpunan data yang dinamis, mengalir, tidak pasti, atau memiliki nilai atribut yang hilang.

DAFTAR PUSTAKA

[1] Q. Tran dan C.-Y. Chan, "How to ConQueR Why-not Questions," *Proceedings of the ACM SIGMOD International Conference on Management of Data,* pp. 15-26, 2010.

[2] G. Koutrika, E. Pitoura dan K. Stefanidis, "Preference-Based Query Personalization," *Advanced Query Processing*, vol. I, no. 36, pp. 57-81, 2013.

[3] S. Borzsonyi, D. Kossmann dan K. Stocker, "The Skyline Operator," *ACM Trans. Database System.,* vol. 25 , no. 2, p. 129–178, 2000.

[4] R. J. Trudeau, Introduction to Graph Theory, New York: Dover Pub, 1993.

[5] Z. Mustaffa and Y. Yusof, "A Comparision of Normalization Techiques in Predicting Dengue Outbreak," *Conference on Business and Economics Research,* vol. 1, 2011.

[6] B. J. Santoso and G.-M. Chu, "Close Dominance Graph: An Efficient Framework for Answering Continuous Top-k Dominating Queries," *IEEE Transactions on Knowledge and Data Engineering,* vol. 26, no. 8, pp. 1853 - 1865, 2013.

[7] Python Software Foundation, "What is Good For?," General Python FAQ, [Online]. Available: https://docs.python.org/3/faq/general.html#what-is-python-good-for. [Accessed 25 Maret 2018].

[8] The Apache Software Foundation, "About the Apache HTTP *Server* Project - The Apache HTTP *Server* Project," The Apache Software Foundation, [Online]. Available: http://httpd.apache.org/ABOUT_APACHE.html. [Accessed 15 Maret 2018].

[9] The PHP Group, "PHP: What is PHP?," [Online]. Available: http://id1.php.net/manual/en/intro-whatis.php. [Accessed 30 Maret 2018].

[10] Almende B.V., "vis.js - A dynamic, browser based visualization library," Almende B.V., 2017. [Online]. Available: http://visjs.org/. [Accessed 12 Februari 2018].

[11] E. Fox and C. Guestrin, "Complexity of Brute Force Search," Coursera, [Online]. Available: https://www.coursera.org/lecture/ml-clustering-and-retrieval/complexity-of-brute-force-search-5R6q3. [Accessed 12 April 2018].

[12] 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].

LAMPIRAN 1

Kode sumber untuk algoritma *Preprocessing*

```
1.  import random
2.  import datetime
3.  import time
4.  import threading
5.  import json
6.  import sys
7.  import csv
8.  import copy
9.  import os, shutil
10. try:
11.     import psutil
12.     enable_psutil = True
13. except ImportError as e:
14.     enable_psutil = False
15. import resource
16.
17.
18. def format_graph(v_global, node, edges):
19.     root = find_skyline(node, v_global)
20.     for x in node:
21.         score = len(v_global[x]["ancestor"])
22.         v_global[x]["score"] = score+1
23.         del v_global[x]["ancestor"]
24.         if x in root:
25.             v_global[x]["is_root"] = True
26.
27. def format_nodes(v, node):
28.     v_local = dict(v)
29.     result = list()
30.     for x in range(len(node)):
31.         v_local[node[x]]["label"] += "\n("
32.         a_idx = 0
33.         for a in attribute_value:
34.             if a_idx == 0:
35.                 v_local[node[x]]["label"] += str(v_loc
    al[node[x]][a])
36.             else:
```

```
37.                 v_local[node[x]]["label"] += ","+str(v
    _local[node[x]][a])
38.             a_idx+=1
39.         v_local[node[x]]["label"] += ")\n"+str(v_local
    [node[x]]['score'])
40.         result.append(v_local[node[x]])
41.     return result
42.
43. def format_layer(v, node, start_point, edges, attribut
    e, cut_size = False):
44.     global node_visited
45.     v_local = dict(v)
46.     result = dict()
47.     num_of_nodes = len(node)
48.     if not cut_size:
49.         cut_size = num_of_nodes/2
50.     for layer in range(0,num_of_nodes+1,cut_size):
51.         result_temp = search_clean_cut(v_local, False,
    layer, edges, attribute, start_point, True)
52.         result_temp = sorted(result_temp, key=generate
    key)
53.         result_temp = find_skyline(result_temp, v_loca
    l)
54.         result[layer] = result_temp
55.         start_point = result[layer]
56.         node_visited = list()
57.         if len(result_temp) < 1:
58.             del result[layer]
59.             break
60.     return result
61.
62. def search_clean_cut(graph, node, layer_size, edges, a
    ttribute, root = [], is_initial = False):
63.     cut = list()
64.     clean_cut = list()
65.     if is_initial:
66.         child = root
67.     else:
68.         if node in node_visited:
69.             return list()
70.         else:
71.             node_visited.append(node)
72.             if graph[node]["score"] >= layer_size:
```

```
73.                 return [node]
74.             child = [d['to'] for d in edges if d['from
    '] == node]
75.     for next in child:
76.         cut += search_clean_cut(graph, next, layer_siz
    e, edges, attribute)
77.     return cut
78.
79. def slice_vertex(v, max):
80.   v_thread = []
81.   for x in range(0,len(v),max):
82.     v_thread.append(v[x:x+max])
83.   return v_thread
84.
85. def remove_from_list(origin, to_remove):
86.   temp = set(to_remove)
87.   result = [value for value in origin if value not in
    to_remove]
88.   return result
89.
90. def merge(first_list, second_list):
91.   return first_list + list(set(second_list) - set(firs
    t_list))
92.
93. def is_dominating(subject, target):
94.     dominate = 0
95.     for attr in attribute_value:
96.         if subject[attr] > target[attr]:
97.             return False
98.         elif subject[attr] < target[attr]:
99.             dominate+=1
100.       if dominate < 1:
101.           return False
102.       else:
103.           return True
104.
105.  def is_skyline(target, set, v):
106.    skyline_cand = list(set)
107.    for candidate in skyline_cand:
108.      if target is candidate:
109.        continue
110.      elif is_dominating(v[candidate], v[target]):
111.        return False
```

```
112.     return True
113.
114.   def find_skyline(set, v):
115.     skyline_cand = list(set)
116.     result = []
117.     i = 1;
118.     for record in skyline_cand:
119.       if is_skyline(record, skyline_cand[0:i], v):
120.         result.append(record)
121.       i+=1
122.     return result
123.
124.   def find_dominating(r, set, v):
125.     result = []
126.     for s in set:
127.       if r is s:
128.         continue
129.       elif is_dominating(v[r], v[s]):
130.         result.append(s)
131.     return result
132.
133.   def progress_report(node):
134.     global node_visited
135.     global num_of_nodes
136.     if enable_psutil:
137.         node_visited = merge(node_visited, [node])
138.         progress = float(len(node_visited))/float(num
    _of_nodes)*float(100)
139.         process = psutil.Process(os.getpid())
140.         mem_usage = float(process.memory_info().rss)/
    1000000.0
141.         sys.stdout.write("\rNumber of visited nodes:
    " + str(len(node_visited)) + " of " + str(num_of_nodes
    ) + " nodes --
    - " + str(progress) + "% ("+str(mem_usage)+" MB)")
142.         sys.stdout.flush()
143.
144.   def generatekey(x):
145.       results = list()
146.       for attr in attribute_value:
147.           results.append(vertex[x][attr])
148.       return tuple(results)
149.
```

```
150.   def build_graph(node, thread_vertex, sub_vertex, is
    _initial = False):
151.       if not is_initial:
152.           if vertex[node]["visited"] or node not in t
    hread_vertex:
153.               return
154.           else:
155.               child_cand = list(find_dominating(node,
    sub_vertex, vertex))
156.               vertex[node]["visited"] = True
157.               for c in child_cand:
158.                   if node not in vertex[c]["ancestor"
    ]:
159.                       vertex[c]["ancestor"].append(no
    de)
160.       else:
161.           child_cand = thread_vertex
162.       child = find_skyline(child_cand, vertex)
163.       sub_vertex_cand = remove_from_list(list(child_c
    and), child)
164.       if not is_initial:
165.           for c in child:
166.               edge_temp = dict()
167.               edge_temp["to"] = c
168.               edge_temp["from"] = node
169.               edges.append(edge_temp)
170.           progress_report(node)
171.       for record in child:
172.           if is_initial:
173.               index_cut = sub_vertex.index(record)
174.               build_graph(record, thread_vertex, sub_
    vertex[index_cut:])
175.           else:
176.               build_graph(record, thread_vertex, sub_
    vertex_cand)
177.       return
178.
179.   class cdgThread(threading.Thread):
180.       def __init__(self, threadID, name, thread_verte
    x, sub_vertex):
181.           threading.Thread.__init__(self)
182.           self.threadID = threadID
183.           self.name = name
```

```
184.            self.thread_vertex = thread_vertex
185.            self.sub_vertex = sub_vertex
186.        def run(self):
187.            print "\nTHREAD "+self.name+" started"
188.            build_graph(None, self.thread_vertex, self.
    sub_vertex, True)
189.            t_end = datetime.datetime.now()
190.            self.runtime = t_end - time_start
191.            print "THREAD "+self.name+" finished with r
    untime " + str(self.runtime)
192.
193.
194.    ts = time.time()
195.    st_start = datetime.datetime.fromtimestamp(ts).strf
    time('%Y-%m-%d__%H:%M:%S')
196.    time_start = datetime.datetime.now()
197.    print "Program started"
198.    count = 0
199.    node_visited = []
200.    num_of_nodes = 0
201.    try:
202.        max_nodes_in_thread = int(sys.argv[1])
203.    except IndexError:
204.        max_nodes_in_thread = 10000
205.    try:
206.        cut_size = int(sys.argv[2])
207.    except IndexError:
208.        cut_size = False
209.    try:
210.        session = sys.argv[3].split(',')
211.        session_name = session[0]
212.        session_type = session[1]
213.        session_full_id = "_"+session_name+"_"+session_
    type
214.    except IndexError:
215.        session = False
216.        session_name = ""
217.        session_type = ""
218.        session_full_id = ""
219.    vertex = dict()
220.    attribute = list()
221.    edges = list()
222.    if session:
```

```
223.        dataset_filename = "datasets/"+session_type+"/d
    ataset_"+session_name+".csv"
224.    else:
225.        dataset_filename = "dataset.csv"
226.    with open(dataset_filename) as csvfile:
227.        readCSV = csv.reader(csvfile, delimiter=',')
228.        init = True
229.        for row in readCSV:
230.            if init:
231.                attribute = list(row)
232.            else:
233.                vertex_data = dict()
234.                for idx in range(len(row)):
235.                    if idx > 1:
236.                        vertex_data[attribute[idx]] = i
    nt(row[idx])
237.                    else:
238.                        vertex_data[attribute[idx]] = r
    ow[idx]
239.                vertex[row[0]] = vertex_data
240.                vertex[row[0]]["score"] = 0
241.                vertex[row[0]]["ancestor"] = list()
242.                vertex[row[0]]["visited"] = False
243.                vertex[row[0]]["is_root"] = False
244.                num_of_nodes+=1
245.            init = False
246.    attribute_value = list(attribute[2:])
247.    properties = dict()
248.    for attr in attribute_value:
249.        vertex_min = sorted(vertex, key=lambda x: (vert
    ex[x][attr]))
250.        properties[attr] = {
251.            "min": vertex[vertex_min[0]][attr],
252.            "max": vertex[vertex_min[-1]][attr]
253.        }
254.    vertex_sorted = sorted(vertex, key=generatekey)
255.    vertex_thread = slice_vertex(vertex_sorted, max_nod
    es_in_thread)
256.    threads = []
257.    for t in range(0,len(vertex_thread)):
258.        threads.append(cdgThread(t+1,"Graph Builder #"+
    str(t+1),vertex_thread[t],vertex_sorted))
259.    for t in threads:
```

```
260.        t.start()
261.    for t in threads:
262.        t.join()
263.    folder = 'session'+session_full_id
264.    for the_file in os.listdir(folder):
265.        file_path = os.path.join(folder, the_file)
266.        try:
267.            if os.path.isfile(file_path):
268.                os.unlink(file_path)
269.        except Exception as e:
270.            print(e)
271.    format_graph(vertex, vertex_sorted, edges)
272.    with open("session"+session_full_id+"/graph.json",
    'w') as fp:
273.        json.dump(vertex, fp)
274.    node_visited = list()
275.    first_layer_root = {k: v for k, v in vertex.iterite
    ms() if v["is_root"]}
276.    first_layer_root = first_layer_root.keys()
277.    clean_cut_layer = format_layer(vertex, vertex_sorte
    d, first_layer_root, edges, attribute_value, cut_size)

278.    with open("session"+session_full_id+"/clean_cut_lay
    er.json", 'w') as fp:
279.        json.dump(clean_cut_layer, fp)
280.    del clean_cut_layer
281.    nodes = format_nodes(vertex, vertex_sorted)
282.    with open("session"+session_full_id+"/nodes.json",
    'w') as fp:
283.        json.dump(nodes, fp)
284.    del nodes
285.    with open("session"+session_full_id+"/attribute.jso
    n", 'w') as fp:
286.        json.dump(attribute_value, fp)
287.    with open("session"+session_full_id+"/properties.js
    on", 'w') as fp:
288.        json.dump(properties, fp)
289.    del properties
290.    with open("session"+session_full_id+"/edges.json",
    'w') as fp:
291.        json.dump(edges, fp)
292.    del edges
293.
```

```
294.    print "\nRUNTIME RESULTS:"
295.    print "Number of nodes: "+str(num_of_nodes)
296.    time_end = datetime.datetime.now()
297.    full_runtime = time_end - time_start
298.    print "Program finished with runtime " + str(full_r
    untime)
299.    if enable_psutil:
300.        process = psutil.Process(os.getpid())
301.        mem_usage = float(process.memory_info().rss)/10
    00000.0
302.        print "Memory usage:",mem_usage
303.    else:
304.        mem_usage = 0.0
305.
306.    res = list()
307.    ts = time.time()
308.    st_end = datetime.datetime.fromtimestamp(ts).strfti
    me('%Y-%m-%d__%H:%M:%S')
309.    res_data = list()
310.    res_data.append(st_start)
311.    res_data.append(st_end)
312.    res_data.append("algo1")
313.    res_data.append(num_of_nodes)
314.    res_data.append(len(attribute_value))
315.    res_data.append(session_type)
316.    res_data.append(len(threads))
317.    res_data.append(full_runtime)
318.    res_data.append(mem_usage)
319.    res.append(res_data)
320.    if session:
321.        with open("session_log/graph_script.csv", "a")
    as output:
322.            writer = csv.writer(output, lineterminator=
    '\n')
323.            writer.writerows(res)
324.    else:
325.        with open("session_log/graph_manual.csv", "a")
    as output:
326.            writer = csv.writer(output, lineterminator=
    '\n')
327.            writer.writerows(res)
```

[Halaman ini sengaja dikosongkan]

LAMPIRAN 2

Kode sumber algoritma berbasis CDG

```
1.  import json
2.  import glob
3.  import sys
4.  import datetime
5.  import time
6.  import os, shutil
7.  import psutil
8.  import resource
9.  import csv
10.
11.
12. def is_dominating(subject, target, attribute):
13.     dominate = 0
14.     for attr in attribute:
15.         if subject[attr] > target[attr]:
16.             return False
17.         elif subject[attr] < target[attr]:
18.             dominate+=1
19.     if dominate < 1:
20.         return False
21.     else:
22.         return True
23.
24. def is_skyline(target, set, v, attribute):
25.     skyline_cand = list(set)
26.     for candidate in skyline_cand:
27.         if target is candidate:
28.             continue
29.         elif is_dominating(v[candidate], v[target], attr
    ibute):
30.             return False
31.     return True
32.
33. def find_skyline(set, v, attribute):
34.     skyline_cand = list(set)
35.     result = []
36.     i = 1
```

```
37.     for record in skyline_cand:
38.         if is_skyline(record, skyline_cand[0:i], v, attr
    ibute):
39.             result.append(record)
40.         i+=1
41.     return result
42. 
43. def normalize_attr(min, max, x):
44.     return (float(1-min)+float(x))/(float(1-
    min)+float(max))
45. 
46. def find_new_clean_cut(graph, node, min_subs, edges, att
    ribute, root = [], is_initial = False):
47.     cut = list()
48.     clean_cut = list()
49. 
50.     if is_initial:
51.         child = root
52.     else:
53.         if node in node_visited:
54.             return list()
55.         else:
56.             node_visited.append(node)
57.             if graph[node]["score"] >= min_subs:
58.                 return [node]
59.             child = [d['to'] for d in edges if d['from']
    == node]
60.     for next in child:
61.         cut += find_new_clean_cut(graph, next, min_subs,
    edges, attribute)
62.     return cut
63. 
64. ts = time.time()
65. st_start = datetime.datetime.fromtimestamp(ts).strftime(
    '%Y-%m-%d__%H:%M:%S')
66. time_start = datetime.datetime.now()
67. print "Program started"
68. expected_subs = int(sys.argv[1])
69. try:
70.     session = sys.argv[3].split(',')
71.     session_name = session[0]
72.     session_type = session[1]
```

```
73.     session_full_id = "_"+session_name+"_"+session_type

74. except IndexError:
75.     session = False
76.     session_name = ""
77.     session_type = ""
78.     session_full_id = ""
79. with open('session'+session_full_id+'/countsubs_results.
    json') as f:
80.     countsubs_results = dict(json.load(f))
81. with open('session'+session_full_id+'/edges.json') as f:

82.     edges = list(json.load(f))
83. with open('session'+session_full_id+'/attribute.json') a
    s f:
84.     attribute = list(json.load(f))
85. with open('session'+session_full_id+'/graph.json') as f:

86.     graph = dict(json.load(f))
87. with open('session'+session_full_id+'/clean_cut_layer.js
    on') as f:
88.     clean_cut_layer = dict(json.load(f))
89. with open('session'+session_full_id+'/properties.json')
    as f:
90.     properties = dict(json.load(f))
91. try:
92.     weight_inp = sys.argv[2]
93.     weight_list = weight_inp.split(',')
94.     weight = dict()
95.     for i in range(len(attribute)):
96.         weight[attribute[i]] = float(weight_list[i])
97. except (IndexError,ValueError) as e:
98.     weight = dict()
99.     weight_default = 1.0/float(len(attribute))
100.        for i in range(len(attribute)):
101.            weight[attribute[i]] = weight_default
102.    print weight
103.    layer_indices = clean_cut_layer.keys()
104.    layer_indices = sorted(map(int, layer_indices))
105.    layer = str([l for l in layer_indices if l <= expec
    ted_subs][-1])
106.    node_visited = list()
```

```
107.    new_clean_cut = find_new_clean_cut(graph, False, ex
    pected_subs, edges, attribute, clean_cut_layer[layer], T
    rue)
108.    new_clean_cut = find_skyline(new_clean_cut, graph,
    attribute)
109.    solution = None
110.    solution_penalty = 999.0
111.    normalized_before = dict()
112.    for attr in attribute:
113.        normalized_before[attr] = normalize_attr(proper
    ties[attr]["min"], properties[attr]["max"], countsubs_re
    sults["target"][attr])
114.    if len(new_clean_cut) <= 0:
115.        graph['00'] = dict()
116.        for attr in attribute:
117.            graph['00'][attr] = properties[attr]["max"]

118.        graph['00']['score'] = len(graph)-1
119.        graph['00']['label'] = "R00"
120.        new_clean_cut.append('00')
121.    for new in new_clean_cut:
122.        print new
123.        penalty = 0.0
124.        for attr in attribute:
125.            normalized_after = normalize_attr(propertie
    s[attr]["min"], properties[attr]["max"], graph[new][attr
    ])
126.            penalty += ((normalized_after - normalized_
    before[attr]) * (1.0 - weight[attr]))
127.            print attr,":",graph[new][attr],((normalize
    d_after - normalized_before[attr]) * weight[attr])
128.        if penalty < solution_penalty:
129.            solution = new
130.            solution_penalty = penalty
131.        print "subscribers :",graph[new]["score"]
132.        print "penalty :",penalty
133.        print "\n"
134.    print "Number of solution candidate",len(new_clean_
    cut)
135.    print "BEST SOLUTION:"
136.    print solution
137.    for attr in attribute:
138.        print attr,":",graph[solution][attr]
```

```
139.     print "subscribers :",graph[solution]["score"]
140.     print "penalty :",solution_penalty
141.     print "\n"
142.     print "\nRUNTIME RESULTS:"
143.     print "Number of nodes: "+str(len(graph))
144.     time_end = datetime.datetime.now()
145.     full_runtime = time_end - time_start
146.     print "Program finished with runtime " + str(full_r
    untime)
147.     process = psutil.Process(os.getpid())
148.     mem_usage = float(process.memory_info().rss)/100000
    0.0
149.     print "Memory usage:",mem_usage
150.
151.     res = list()
152.     ts = time.time()
153.     st_end = datetime.datetime.fromtimestamp(ts).strfti
    me('%Y-%m-%d__%H:%M:%S')
154.     res_data = list()
155.     res_data.append(st_start)
156.     res_data.append(st_end)
157.     res_data.append("algo1")
158.     res_data.append(len(graph))
159.     res_data.append(len(attribute))
160.     res_data.append(session_type)
161.     res_data.append(expected_subs)
162.     initial_value = list()
163.     recomended_value = list()
164.     weight_list = list()
165.     for attr in attribute:
166.         initial_value.append(countsubs_results["target"
    ][attr])
167.         recomended_value.append(graph[solution][attr])

168.         weight_list.append(weight[attr])
169.     res_data.append("|".join(list(map(str, weight_list)
    )))
170.     res_data.append("|".join(list(map(str, initial_valu
    e))))
171.     res_data.append("|".join(list(map(str, recomended_v
    alue))))
172.     res_data.append(graph[solution]["label"])
173.     res_data.append(solution_penalty)
```

```
174.     res_data.append(full_runtime)
175.     res_data.append(mem_usage)
176.     res.append(res_data)
177.     if session:
178.         print "Saved to session_log/solution_script.csv
    "
179.         with open("session_log/solution_script.csv", "a
    ") as output:
180.             writer = csv.writer(output, lineterminator=
    '\n')
181.             writer.writerows(res)
182.     else:
183.         print session,"Saved to session_log/solution_ma
    nual.csv"
184.         with open("session_log/solution_manual.csv", "a
    ") as output:
185.             writer = csv.writer(output, lineterminator=
    '\n')
186.             writer.writerows(res)
```

BIODATA PENULIS

I Putu Eka Wira Mahardika, akrab dipanggil Wira, lahir pada tanggal 10 Juni 1996 di Denpasar. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Departemen Informatika Institut Teknologi Sepuluh Nopember. Memiliki beberapa hobi antara lain membaca dan mendengarkan musik. Selama masa kuliah penulis mengambil bidang minat Komputasi Berbasis Jaringan (KBJ). Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain sebagai Staff Media Informasi Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ajaran 2015/2016. Penulis juga aktif dalam kepanitiaan Schematics, antara lain sebagai Staff Biro Desain Dokumentasi dan Dekorasi pada tahun ajaran 2015/2016. Penulis juga aktif dalam organisasi Tim Pembina Kerohanian Hindu sebagai Staff Departemen Komunikasi dan Informasi pada tahun ajaran 2015/2016 dan kemudian dilanjutkan sebagai Kepala Departemen Media Kreatif pada tahun ajaran 2016/2017. Selama menempuh Pendidikan di kampus, penulis juga aktif dalam kegiatan kepanitiaan acara ITS EXPO 2016 dengan berkontribusi sebagai Staff Ahli Biro WebApps.