TUGAS AKHIR – IF184802

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT ON REACHING K SUBSCRIBERS* DENGAN PENDEKATAN *VIRTUAL RECORDS*

BAGUS DHARMA ISWARA
NRP 05111540000028

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

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – IF184802

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT ON REACHING K SUBSCRIBERS* DENGAN PENDEKATAN *VIRTUAL RECORDS*

BAGUS DHARMA ISWARA
NRP 05111540000028

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

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – IF184802

DESIGN AND IMPLEMENTATION APPLICATION TO ANSWER WHY-NOT ON REACHING K SUBSCRIBERS PROBLEM WITH VIRTUAL RECORDS APPROACHES

BAGUS DHARMA ISWARA
NRP 05111540000028

Supervisors
Bagus Jati Santoso, S.Kom., Ph.D.

DEPARTMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT ON REACHING K SUBSCRIBERS* DENGAN PENDEKATAN *VIRTUAL RECORDS*

TUGAS AKHIR

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

Oleh
BAGUS DHARMA ISWARA
NRP: 05111540000028

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Bagus Jati Santoso, S.Kom., Ph.D. .....
   (NIP: 198611252018031001) (Pembimbing 1)

SURABAYA
JANUARI 2020

[Halaman ini sengaja dikosongkan]

DESAIN DAN IMPLEMENTASI APLIKASI UNTUK MENJAWAB PERMASALAHAN *WHY-NOT ON REACHING K SUBSCRIBERS* DENGAN PENDEKATAN *VIRTUAL RECORDS*

Nama Mahasiswa : Bagus Dharma Iswara
NRP : 05111540000028
Jurusan : Departemen Teknik Informatika ELECTICS-ITS
Dosen Pembimbing 1 : Bagus Jati Santoso, S.Kom., Ph.D.

ABSTRAK

Di zaman sekarang teknologi sudah berkembang semakin pesat, banyak perusahaan menggunakan data sebagai dasar untuk mengambil keputusan. Dalam hal ini perusahaan manufaktur yang ingin produknya menjangkau pelanggan sebanyak-banyaknya namun dengan biaya yang rendah, karena ketika jumlah pelanggan tidak sesuai dengan ekspektasi perusahaan maka langkah yang dilakukan perusahaan harus memperbaiki produknya, namun tetap mempertimbangkan biaya perubahan yang seminimal mungkin. Permasalahan ini selanjutnya disebut dengan istilah "*Why-Not on Reaching k Subscribers* dengan pendekatan *Virtual Records"*.

Penelitian ini dilakukan untuk menjawab tantangan tersebut dengan ilmu rekayasa data serta sebagai solusi baru untuk mengembangkan solusi permasalahan yang telah ditemukan sebelumnya dengan mencari data produk lain yang belum ada namun memiliki dampak yang lebih baik dibandingkan produk yang sudah ada sebelumnya. Penelitian ini juga menawarkan solusi untuk memperbaiki nilai pada setiap atribut produk agar mencapat jumlah data yang diharapkan pengguna secara optimal dibandingkan metode yang sudah ada.

Hasil uji coba menunjukkan bahwa algoritma yang ditawarkan dapat memberikan banyak hasil solusi terbaik kepada pengguna dengan waktu eksekusi yang baik dimana waktu eksekusi didapatkan secara signifikan. Solusi untuk menghitung biaya perubahan pada semua solusi 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: Virtual Records, Why-Not, Clustering, Pre-Indexing, Rekayasa Data

DESIGN AND IMPLEMENTATION APPLICATION TO ANSWER WHY-NOT ON REACHING K SUBSCRIBERS PROBLEM WITH VIRTUAL RECORDS APPROACH

Student's Name	**:**	**Bagus Dharma Iswara**
Student's ID	**:**	**05111540000028**
Department	**:**	**Department of Informatics Engineering, Faculty of Intelligent Electrical and Informatics Technology - ITS**
First Advisor	**:**	**Bagus Jati Santoso, S.Kom., Ph.D.**

ABSTRACT

In this day and age, technology has developed more rapidly. Many companies use data as a basis for making decisions. In this case the manufacturing company want it's products to reach as many customers as possible. But at a low cost, because when the number of customers does not match company expectations then the steps taken by the company must improve it's products, but still consider the minimum change costs. This problem is called "Why-Not on Reaching k Subscribers with Virtual Records approach".

This research was conducted to answer these challenges with data engineering, as well as new solutions to develop solutions to problems that have been found before by searching for other product data that does not yet exist, but has a better impact than existing products. This research also offers a solution to improve the value of each product attribute in order to optimally reach the amount of data the user expects from existing methods.

The trial results show that the algoritm offered can provide many of the best solution results to users with a good

execution time where the execution time is obtained significantly. The solution for calculating the cost of change in all solutions can also be used for datasets with different value ranges, so that the algoritm offered can be implemented for various types of datasets.

Keywords: Virtual Records, Why-Not, Clustering, Pre-Indexing, Data Engineering

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* DENGAN PENDEKATAN *VIRTUAL RECORDS*".

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 penulis I Made Wijaya Kusuma serta Ni Ketut Kartikasari, terimakasih atas doa dan bantuan moral maupun material selama penulis menempuh pendidikan di Departemen Informatika ITS.
3. Bapak Bagus Jati Santoso, S.Kom, Ph.D. selaku pembimbing I yang selama ini telah membantu dan membimbing penulis selama pengerjaan tugas akhir

4. Bapak Dr.Eng Darlis Herumurti, S.Kom.,M.Kom. selaku Kepala Departemen Informatika ITS.
5. Bapak Dr. Radityo Anggoro, S.Kom.,M.Sc. selaku koordinator TA, dan segenap dosen Teknik Informatika yang telah banyak memberikan ilmu kepada penulis.
6. Bapak dan Ibu Sutantra beserta keluarga atas kesempatan tempat tinggal yang telah diberikan kepada penulis selama di Surabaya..
7. Aditya Pramana yang telah membantu penulis dalam menjalani kegiatan kuliah.
8. Wahyu dan Satria yang telah menemani penulis dari awal masa mahasiswa baru sampai saat ini.
9. Seluruh rekan TC 2015 yang saya cintai dan saya banggakan.
10. Rekan-rekan kos J2 Arka, Soma, Bayu, Made, Fandy dan Suparta atas segala dukungan dan waktunya dalam menemani penulis selama di Surabaya.

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, Januari 2020

Bagus Dharma Iswara

DAFTAR ISI

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 mengenai sistem basis data telah membuat kemajuan luar biasa selama beberapa tahun terakhir dan mendapat perhatian lebih terkait masalah pengembangan performa, kualitas serta penggunaan [1]. Algoritma *query* adalah salah satu sistem basis data yang berkembang saat ini digunakan untuk membantu mencari data dengan nilai atribut terbaik. Beberapa jenis algoritma *query* seperti *Top-K Query, Skyline Query, dan Top-K Dominating Query* dikembangkan. Perkembangan ini memunculkan permasalahan baru, salah satunya adalah ketika pengguna merasa hasil dari sebuah *query* tidak sesuai dengan ekspektasi dan mulai bertanya mengenai alasan data tertentu tidak muncul pada hasil *query*. Permasalahan ini dikenal dengan "*Why-Not Question*".

Berawal dari permasalahan "*Why-Not Question*", terdapat masalah lain yang dapat diangkat dan dicari solusinya. Diketahui suatu nilai tertentu dari sebuah produk yang dimiliki oleh seorang pengguna dan terdapat sebuah dataset yang berisi data preferemsi untuk masing-masing konsumen. Untuk mengetahui berapa banyak data konsumen yang sesuai dengan kriteria produk tersebut dilakukan sebuah *query* oleh pengguna.

Permasalahan akan muncul ketika jumlah konsumen hasil *query* lebih rendah dibandingkan ekspektasi pengguna. Permasalahan ini selanjutnya disebut dengan istilah "*Why-Not on Reaching k Subscribers*". Perbedaan mendasar permasalahan "*Why-Not Question*" dan "*Why-Not on Reaching k Subscribers*" terdapat pada masukan permasalahan. Pada "*Why-Not Question*" yang menjadi masukan adalah data mana yang hilang dari hasil *query*, sedangkan "*Why-Not on Reaching k Subscribers*" adalah

jumlah data yang diharapkan dari hasil *query.* Permasalahan yang sama telah dipecahkan oleh Wira [2] menggunakan pendekatan *existing product.* Pendekatan ini dilakukan dengan mencari data-data produk referensi yang sudah ada sebelumnya kemudian pengguna melakukan *query* melalui data preferensi dari data produk yang telah dipilih sehingga hasil dari *query* yang dimasukkan akan menghasilkan atribut produk yang dapat mencapai jumlah data konsumen yang diharapkan secara optimal dengan nilai penalti yang rendah.

Secara garis besar penyelesaian masalah dilakukan melalui 2 tahap, yaitu *Preprocessing* dan Algoritma berbasis *Close Dominance Graph* (CDG). *Preprocessing* dilakukan untuk melakukan *indexing* dengan membentuk struktur data graf CDG. Proses *indexing* dilakukan dengan membuat lapisan yang disebut *LAYER*. *Close Dominance Graph* (CDG) adalah sebuah kerangka kerja yang memodelkan himpunan berdasarkan *close dominance relation* antar data. Pada saat CDG dan *LAYER* telah terbentuk, dilakukan proses menggunakan algoritma CDG dengan cara menginputkan sebuah *query* preferensi asli q_0, preferensi perubahan nilai atribut pada *query wp*, dan ekspektasi *subscribers k.* Pencarian dilakukan dengan cara menelusuri graf yang telah dibentuk menggunakan algoritma DFS hingga menemukan data yang sesuai dengan ekspektasi pengguna *k.*

Kekurangan pada permasalahan yang telah dipecahkan sebelumnya adalah solusi yang digunakan harus terdapat produknya terlebih dahulu sebagai referensi, kemudian dilakukan pencarian data produk yang sudah ada dan memilih produk mana yang akan dipakai. Melihat terdapat kekurangan pada solusi sebelumnya, maka dapat ditawarkan solusi baru dengan mencari produk-produk lain yang belum ada, tetapi memiliki dampak yang lebih baik dibandingkan produk yang sudah ada sebelumnya.

Berdasarkan uraian di atas maka dapat dilihat bahwa dibutuhkan sebuah algoritma untuk menjawab permasalahan "*Why-Not on Reaching k Subscribers*" dengan menggunakan pendekatan *Virtual Records*. Algoritma berbasis *Virtual Records*

sangat penting digunakan dalam memperbaiki kekurangan dalam permasalahan *Why-Not On Reaching k Subscribers* karena pada algoritma berbasis *virtual records* tidak perlu mencari referensi produk yang sudah ada sebelumnya. Produk-produk ini akan didapatkan melalui pencarian data produk referensi yang belum ada sebelumnya namun tetap dapat memiliki dampak yang lebih besar serta dapat mencapai jumlah target *subscribers* yang diinginkan pengguna.

Pada permasalahan *Why-Not* telah dijelaskan sebelumnya bahwa masukan permasalahan adalah sebuah data yang hilang dalam sebuah *query*, kemudian berdasarkan pada permasalahan ini terdapat permasalahan lain yang dapat dicari solusinya yaitu permasalahan *Why-Not on Reaching k Subscribers* yang dimana masukan pada permasalahan ini adalah jumlah target *subscribers* yang diinginkan pengguna pada suatu produk yang dimiliki.

Virtual Records pada permasalahan ini digunakan sebagai bentuk solusi baru untuk memecahkan permasalahan *Why-Not on Reaching k Subscribers*, yaitu dengan mencari titik-titik *virtual* yang ada pada suatu himpunan data produk yang sangat bervariasi. Titik *virtual* ini nantinya akan digunakan sebagai referensi hasil perbaikan nilai untuk sebuah atribut produk. Algoritma yang akan dirancang nantinya diharapkan mampu memperbaiki nilai pada setiap atribut dari produk agar hasil *query* dapat mencapai jumlah target *subscribers* yang diharapkan pengguna secara optimal serta dapat memperbaiki data suatu produk.

Nilai penalti juga akan dipertimbangkan sebagai biaya dari suatu perubahan. Diharapkan juga hasil perbaikan nilai atribut juga memiliki nilai penalti serendah mungkin serta dapat menghasilkan solusi yang lebih banyak kepada pengguna.

1.2. Rumusan Masalah

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

1. Bagaimana algoritma yang tepat untuk memperbaiki nilai atribut produk untuk mencapai ekspektasi pengguna terhadap jumlah konsumen berbasis *virtual records*?
2. Bagaimana mengindeks seluruh data *subscribers* untuk mendukung solusi atas permasalahan *Why-Not on Reaching k Subscribers* dengan pendekatan *virtual records*?
3. Bagaimana perbandingan performa antara metode baru dengan metode lama ?

1.3. Batasan Masalah

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

1. Himpunan data yang digunakan adalah data *real-life* dan sintetis.
2. Algoritma ini hanya memproses nilai atribut yang bertipe numerik.
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. Merancang algoritma yang tepat menggunakan pendekatan *virtual records* untuk memperbaiki nilai atribut produk agar mencapai ekspektasi pengguna terhadap jumlah konsumen.
2. Menemukan formulasi yang tepat dalam mengindeks seluruh data konsumen untuk mendukung solusi atas permasalahan *Why-Not on Reaching k Subscribers* dengan pendekatan *virtual records.*
3. Mengetahui perbaikan performa berbasis *virtual records*.

1.5. Manfaat

Manfaat dari pembuatan tugas akhir ini adalah:

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

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 PyCharm.

5. Pengujian dan evaluasi

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

   1. 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* dengan pendekatan *Virtual Records*.

   2. Pengujian Penggunaan Memori

      Pengujian ini akan berfokus pada pengukuran besarnya memori yang digunakan saat 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* dengan pendekatan *Virtual Records* yang diajukan untuk tugas akhir ini. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat.

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*
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
$score(v)$	jumlah data yang didominasi oleh *virtual records*
k	jumlah ekspektasi *subscribers*
s	jumlah solusi yang akan dihasilkan
n_cluster	jumlah *cluster* yang akan dibuat
qo	*query* awal sebelum dilakukan *query refinement*
q'	hasil *query refinement*
$CAND$	himpunan kandidat q'

Simbol	Keterangan
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

Permasalahan hasil kueri seperti *Why* dan *Why-Not* dapat dijawab menggunakan suatu pendekatan untuk menghasilkan kueri baru yang selanjutnya akan didefinisi sebagai *query refinement*. Hasil riset terakhir pada [3] dijelaskan ada 2 kriteria untuk pendekatan *query refinement* yang baik. Pertama *query* tersebut bersifat *similar* berarti memiliki sedikit perubahan jika dibandingkan dengan *query* aslinya. Kriteria kedua adalah *refined query* yang baik bersifat *precise* berarti memiliki sedikit data tambahan pada hasil *query*.

Diperlukan suatu metrik (alat ukur) untuk membandingkan kualitas dari sebuah *refined query* karena jumlah kemungkinan terhadap sebuah *query* sangat banyak. Metrik tersebut akan bermanfaat agar algoritma yang memproduksi *refined query* hanya mengembalikan sebuah *query* terbaik sebagai solusinya. Berdasarkan pada 2 kriteria *refined query* ini, maka kriteria tersebut dapat dijadikan sebagai metrik atas kualitas *refined query*.

Pertama adalah metrik ketidakmiripan. *Refined query* yang baik seharusnya semirip mungkin dari kueri aslinya. Diketahui sebuah kueri asli *Q* dan *refined query Q'*, dilakukan pembandingan tingkat kemiripan antara *Q* dan *Q'* dengan mengukur jarak/selisih

dari perubahan yang paling minimal. Kueri dikatakan lebih mirip (atau lebih tidak mirip) terhadap satu sama lain jika jarak perubahannya kecil (begitu pula sebaliknya). *Q* dan *Q'* bersifat *union-compatible* atau seluruh atribut pada klausa select *Q* dan *Q'* adalah sama. Klausa *from* dan klausa *where* dipertimbangkan sebagai operator perubahan *Q* menjadi *Q'* untuk diubah. Terdapat empat kunci pada operasi perubahan *query*, yang pertama adalah (O_1) pada klausa *where* beberapa nilai konstan pada predikat *select* diubah, kedua (O_2) pada klausa *where* ditambahkan sebuah predikat *select*, ketiga (O_3) pada klausa *where* ditambahkan atau dihilangkan predikat *join*. Keempat (O_4) pada klausa *from* ditambahkan atau dihilangkan *relation*. Memodelkan O_1 seperti yang telah diketahui bahwa tidak ada perubahan operator secara eksplisit ketika menghilangkan predikat *select.* Mengubah jarak dari nilai *select* untuk menutupi domain seluruh atribut dapat dilakukan dengan proses predikat *select* dihilangkan secara efisien. Pada klausa *from* untuk menghilangkan relasi R_i menggunakan O_4 seluruh predikat *select* dan *join* yang berasosiasi R_i juga dihilangkan sebagai bagian dari operasi perubahan.

Biaya untuk operasi perubahan O_i dinyatakan sebagai w_i $i \in [1,4]$. Selanjutnya mengasumsikan bahwa $w_1 < w_2 < w_3 < w_4$[2] Menyatakan n_i sebagai total operasi O_i yang digunakan untuk merubah *Q* menjadi *Q'*, kemudian dapat dihitung secara efisien untuk *Q* dan *Q'*.

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

Dari seluruh hasil *refined query* yang mungkin akan menjadi solusi, *Skyline refined query* adalah solusi terbaik. *Skyline refined query* memiliki nilai terbaik (paling rendah) pada metrik ketidakmiripan maupun pada metrik ketidaktepatan [3].

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

2.2. 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* [4].

2.3. *Query* Berbasis Preferensi

Pada basis data, representasi dari preferensi memiliki dua kategori utama yaitu preferensi kuantitatif dan preferensi kualitatif. Preferensi kuantitatif diekspresikan dengan menetapkan skor pada suatu data atau pada elemen kueri 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*. Sedangkan preferensi kualitatif dispesifikasikan menggunakan predikat biner yang dapat membandingkan antara suatu data dengan data lainnya secara relatif [5].

2.4. Min-Max Algorithm

Salah satu metode yang digunakan dalam melakukan normalisasi data adalah Algoritma Min-Max. 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 [6]:

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

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

2.5. Clustering Algoritm

Clustering adalah teknik umum untuk analisis data statistik, yang digunakan di banyak bidang, termasuk pembelajaran mesin, penambangan data, pengenalan pola, analisis gambar dan bioinformatika. *Clustering* adalah proses pengelompokan objek yang mirip ke dalam kelompok yang berbeda, atau lebih tepatnya, pemartisian data yang diatur ke dalam himpunan bagian, sehingga data di setiap subset sesuai dengan beberapa ukuran jarak yang ditentukan.

Clustering dapat dianggap yang paling penting pada masalah *unsupervised learning* jadi, seperti setiap masalah lain dari jenis ini, ini berkaitan dengan menemukan struktur dalam kumpulan data yang tidak berlabel. *Cluster* adalah kumpulan objek yang "mirip" di antara mereka dan "berbeda" dengan objek milik cluster lain.

Proses *Clustering* menggunakan algoritma *K-means.* Algoritma ini memberikan setiap titik ke Cluster pusatnya yang disebut dengan *centroid.*Titik pusat adalah rata-rata dari semua titik dalam cluster tersebut [7].

2.6. *Virtual Records*

Virtual Records merupakan *record* yang tidak mengandung nilai data, *record* berisi pointer *logic* ke *record* fisik tertentu. Langkah ini bukan mereplikasi, melainkan menyimpan satu salinan *record* fisik, dan di tempat lain menyimpan *record virtual* yang berisi pointer ke *record* fisik tersebut [8]. Penelitian ini menggunakan konsep *Virtual Records* sebagai struktur data untuk menunjang performa algoritma.

2.7. Python

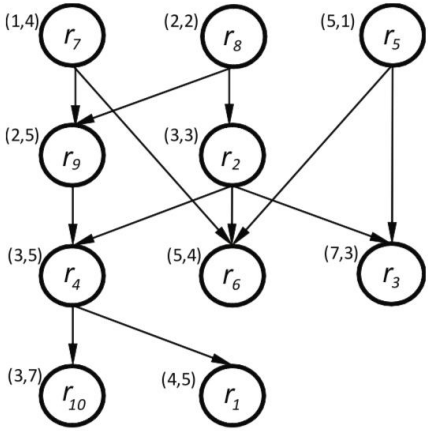
Python adalah Bahasa pemrograman yang ditafsirkan, interaktif, dan berorientasi objek. Ini menggabungkan modul, pengecualian, pengetikan dinamis, tipe data dinamis tingkat sangat tinggi, dan kelas. Python menggabungkan kekuatan luar biasa dengan sintaks yang sangat jelas. Ini memiliki *interfaces* ke banyak panggilan sistem dan *library*. Dapat digunakan sebagai Bahasa ekstensi untuk aplikasi yang mebutuhkan *interfaces* yang dapat diprogram dan berjalan pada banyak varian Unix, Mac, dan Windows 2000 serta yang lebih baru [9].

Pada penelitian ini Python digunakan sebagai bahasa pemrograman untuk mengimplementasikan algoritma.

2.8. Metode *Close Dominance Graph* (CDG)

Close Dominance Graph (CDG) adalah sebuah kerangka kerja yang memodelkan himpunan berdasarkan close dominance relation antar data. Sebagai contoh sebuah set R berisi himpunan data dalam *multidimensional space*. $A(x)$ mewakili himpunan data pada R yang mendominasi data $x \in R$, data $x \in R$ disebut *closely dominated* pada data lain $s \in A(r)$. Hubungan relasi ini dilambang 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

Pada Gambar 2.1 terdapat 10 data yang memiliki 2 dimensi atribut, data r_4 didominasi oleh $A(r_4) = \{r_{2,}, r_7, r_8, r_9\}$. Pada gambar, r_2 dan r_9 tidak mendominasi data lain pada $A(r_4)$ maka akan terbentuk $r_2 \rightarrow r_7$ dan $r_9 \rightarrow r_4$ pada CDG [10].

Pada penelitian ini menggunakan metode *Close Dominance Graph* (CDG) sebagai landasan dasar dalam membangun algoritma pembanding.

[Halaman ini sengaja dikosongkan]

BAB III
DESAIN DAN PERANCANGAN

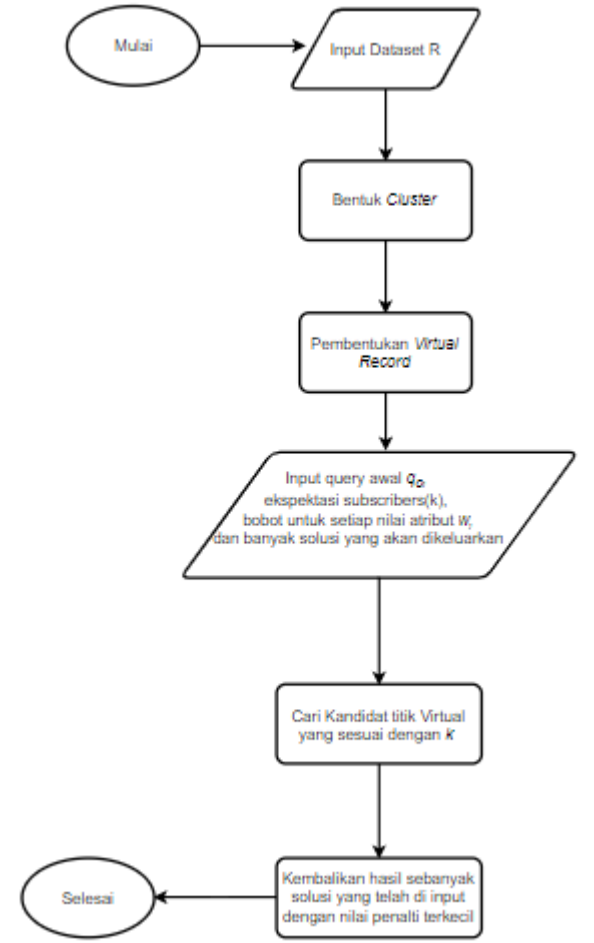
Pada bab ini akan dijelaskan mengenai perancangan sistem program yang akan dibuat. Secara garis besar alur program akan melakukan proses clustering kemudian dilanjutkan dengan pencarian berdasarkan algoritma berbasis *virtual records*

1. *Clustering* digunakan untuk mengumpulkan data berdasarkan jarak terdekatnya agar memudahkan dalam proses pembuatan *virtual records* pada himpunan data.
2. Penerapan algoritma untuk menjawab permasalahan *Why-Not on Reaching k Subscribers dengan Pendekatan Virtual Records* dari struktur data *Virtual Records* yang telah dibentuk pada proses *Clustering*.

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

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

Alur program secara garis besar dilakukan menggunakan metode *clustering* dan dilanjutkan dengan algoritma berbasis *virtual records*. Alur program dimulai dari memasukkan himpunan data *R* yang selanjutnya akan dibentuk beberapa *cluster* dari himpunan data yang ada. Ketika *cluster* pada himpunan data sudah

terbentuk maka dilanjutkan dengan mencari titik *virtual* pada setiap *cluster* yang telah dihasilkan sebelumnya. Pembentukan atau pencarian titik-titik *virtual* dilakukan dengan mencari nilai minimal pada setiap atribut pada setiap *cluster* yang dihasilkan. Kemudian dilanjutkan dengan memasukkan *query* awal qo, ekspektasi jumlah target *subscribers k*, bobot koefisiensi perubahan atau toleransi pengguna terhadap perubahan atribut produk *w*, serta banyak solusi yang akan dihasilkan sebagai hasil dari q'. Dilanjutkan dengan mencari nilai *score(v)* pada setiap titik *virtual* pada setiap *cluster*.

Nilai *score(v)* didapatkan melalui perhitungan total data yang memiliki nilai atribut lebih buruk atau lebih besar dibandingkan nilai atribut pada setiap titik *virtual*. *Score(v)* yang telah didapatkan akan diperiksa dengan jumlah target ekspektasi *subscribers k*, ketika *score(v)* memiliki nilai yang lebih tinggi dibandingkan jumlah target ekspektasi *subscribers* atau minimal memiliki nilai yang sama dengan jumlah target ekspektasi *subscribers k* maka titik *virtual* akan dimasukkan sebagai titik kandidat q'.

Diakhiri dengan mengeluarkan hasil *query* q' yang memiliki nilai penalti terkecil sebanyak solusi yang dimasukkan pengguna. Tahapan *cluster* Pada tahap algoritma ini 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 *Clustering* terlebih dahulu sehingga membentuk beberapa jumlah *cluster* yang nantinya digunakan untuk mencari titik data *virtual* dari tiap *cluster* dan membentuk sebuah struktur data *Virtual Records*. Setelah *cluster* dan titik data *virtual* terbentuk, maka barulah dilanjutkan ke tahap Algoritma Berbasis *Virtual Records*.

Pada Algoritma Berbasis *Virtual Records*, pengguna melakukan masukan sebuah *query* preferensi asli q_0, preferensi perubahan nilai atribut pada *query* λ_P, ekspektasi *subscribers k*, dan jumlah solusi keluaran yang akan dihasilkan.

Program akan mencari titik-titik data *virtual* yang sesuai dengan ekspektasi pengguna *k*. Solusi yang dihasilkan adalah solusi dengan data nilai penalti yang paling rendah serta mengeluarkan hasil sebanyak masukan solusi yang diinginkan oleh pengguna.

3.1.1 *Clustering*

Pengerjaan program diawali dengan mempersiapkan terlebih dahulu himpunan data yang akan digunakan dalam proses pengerjaan. *Clustering* merupakan suatu proses penganalisaan data, yang sering dimasukkan sebagai salah satu metode *Data Mining*.

Clustering digunakan untuk mengumpulkan atau mengelompokkan himpunan data berdasarkan karakteristik yang sama atau melalui data yang memiliki jarak terdekat antar data. Pada tahap *clustering* ini menggunakan metode *k-means*, yaitu salah satu metode yang banyak digunakan dalam melakukan proses *clustering*.

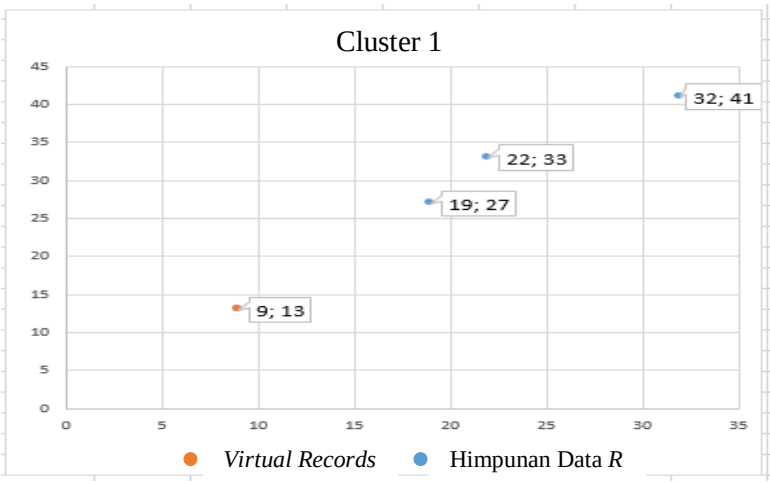
Secara umum metode *k-means* melakukan proses pengelompokan data dengan cara menentukan jumlah cluster yang akan dihasilkan pada suatu himpunan data, mengalokasikan data secara *random* ke *cluster* yang tersedia, menghitung rata-rata setiap *cluster* dari data yang tergabung di dalamnya, dilanjutkan dengan mengalokasikan kembali semua data ke *cluster* terdekat.

Selanjutnya hasil dari proses *clustering* yang telah dihasilkan akan dapat digunakan sebagai penunjang algoritma pada Algoritma Berbasis *Virtual Records*. Ilustrasi proses pembentukan *cluster* pada tahap *Clustering* dapat dilihat pada Gambar 3.2, Gambar 3.3 dan Gambar 3.4.

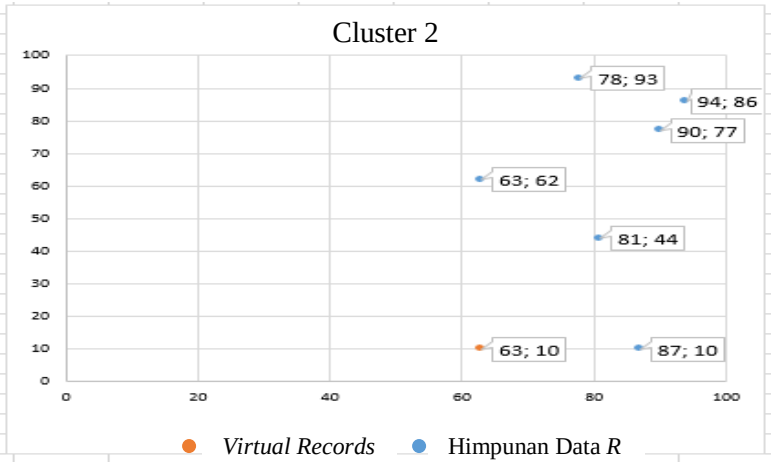
Tabel 3.1 Contoh himpunan data R dengan 15 data

R	a_1	a_2
r_1	35	94
r_2	32	41
r_3	19	27
r_4	78	93

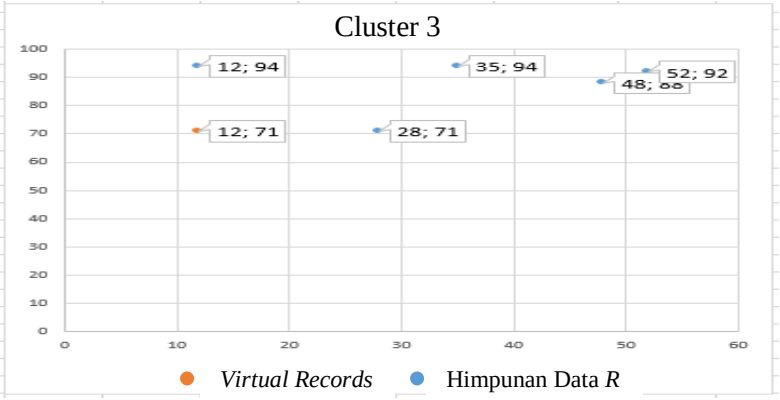
R	a_1	a_2
r_5	28	71
r_6	90	77
r_7	48	88
r_8	94	86
r_9	52	92
r_{10}	22	33
r_{11}	81	44
r_{12}	63	62
r_{13}	9	13
r_{14}	87	10
r_{15}	12	94



Gambar 3.2 Ilustrasi *Clustering* (Bagian 1)



Gambar 3.3 Ilustrasi *Clustering* (Bagian 2)

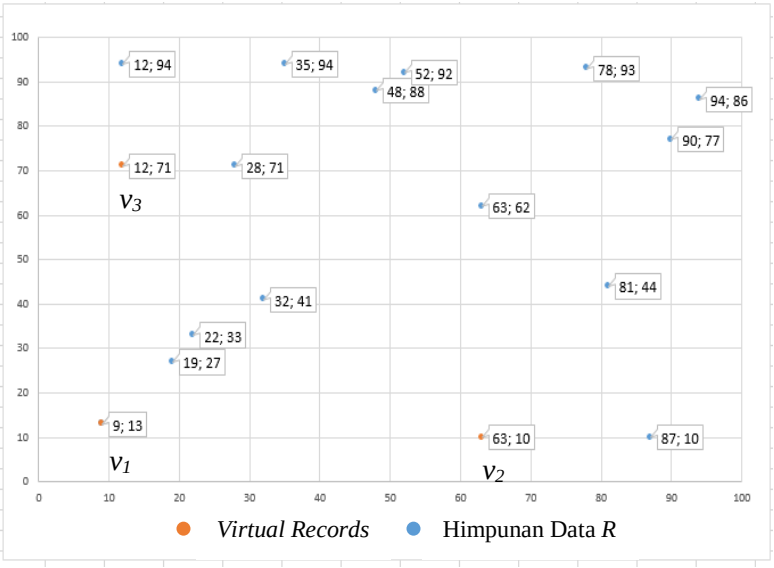


Gambar 3.4 Ilustrasi *Clustering* (Bagian 3)

Algoritma *pre-indexing* pada penelitian ini menggunakan struktur data *Virtual Records*. Diawali dengan memasukkan himpunan data *R* dengan ukuran dua dimensi, himpunan data *R* selanjutnya akan dibentuk tahapan *cluster* sebanyak 5 tahap, pada tahap pertama himpunan data *R* dibentuk sebanyak 100 *cluster*, dilanjutkan dengan pembentukan *cluster* tahap kedua sebanyak 75 *cluster*, tahap ketiga sebanyak 50 c*luster*, tahap keempat sebanyak 25 *cluster* dan terakhir pada tahap kelima akan dibentuk sebanyak 10 *cluster*. Setelah semua *cluster* terbentuk maka dilanjutkan dengan mencari titik *virtual* dari setiap *cluster* dengan cara menghitung nilai atribut minimal dari setiap dimensi dalam setiap *cluster*.

Score(v) didapatkan dari jumlah total dominasi data oleh *virtual records* terhadap himpunan data *R* dengan mencari seluruh himpunan data *R* yang terdapat pada rentang yang dimulai dari data titik *virtual* setiap *cluster* sampai rentang setiap dimensi maksimal. Kemudian menghitung data yang memiliki nilai atribut lebih besar dari nilai data atribut titik *virtual*. Contoh himpunan data *R* dapat dilihat pada Tabel 3.1. Hasil dari *Virtual Records (R)* dapat dilihat pada Gambar 3.5 dan untuk *score(v)* dapat dilihat pada Tabel 3.2. Sebagai contoh gambaran pembentukan *cluster* dapat dilihat pada Gambar 3.2, Gambar 3.3 dan Gambar 3.4 selanjutnya hasil dari himpunan data *R* dilakukan *clustering* sebanyak 3 *cluster* untuk mendapat titik *virtual* dari setiap *cluster*.

Pada Gambar 3.5 sebuah data dilambangkan dengan titik pada grafik berisi dengan angka disamping kanan sebagai nilai atribut a_1 dan a_2 secara berurutan dan sudah memiliki *Virtual Records* yang selanjutnya dapat dilakukan perhitungan untuk mencari titik-titik mana saja yang didominasi oleh titik *virtual* yang telah didapatkan.



Gambar 3.5 Contoh hasil *Virtual Records* pada himpunan data R

Tabel 3.2 Contoh hasil nilai *score(v)* dari 3 titik *virtual*

Titik Virtual	*score(v)*
v_1	15
v_2	6
v_3	14

3.1.2 Metode Pengukuran Biaya Perubahan

Perubahan nilai atribut pada *query* asli q_0 dilakukan untuk mengukur toleransi terhadap pengguna. Pada penelitian ini akan mendefinisikan metode mengenai 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'. Biaya perubahan nilai atribut ΔP dapat didefinisikan secara sederhana, yaitu selisih antara nilai atribut baru $p' \in P$ dengan atribut pada *query* awal $p_o \in P$, atau dalam notasi matematika

dinyatakan dengan $\Delta P = p' - p_o$. Selanjutnya permasalahan yang muncul adalah pengguna pasti memiliki preferensi atas atribut mana yang sebaiknya diubah maka pengguna membutuhkan pembobotan w_i untuk setiap atributnya. Maka dapat dirumuskan persamaan dasar pada 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) merupakan 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 *Virtual Records*

Pada bagian ini dijelaskan desain metode dari algoritma untuk melakukan query refinement terhadap permasalahan *Why-Not on Reaching k Subscribers* dengan pendekatan *Virtual Records*. Algoritma berbasis *Virtual Records* secara garis besar dilakukan dengan cara mengolah data serta melakukan *pre-indexing* terhadap data yang dimiliki. Data dikelompokkan menjadi beberapa kelompok atau *cluster*, setelah *cluster* terbentuk maka dilanjutkan dengan mencari titik *virtual* dari setiap *cluster* dengan mencari nilai atribut minimal dari setiap dimensi. Masukan pada algoritma berbasis *Virtual Records* adalah himpunan data yang telah dipersiapkan pada tahap *Clustering Virtual Records*, *query* awal q_o, ekspektasi jumlah subscribers k, koefisien bobot

untuk perubahan setiap nilai atribut $\lambda_P = \{\lambda_{p_1}, \lambda_{p_2}, \ldots, \lambda_{p_d}\}$ dan banyak solusi *s* yang diinginkan pengguna. Algoritma dimulai dengan mencari nilai atribut titik *virtual* pada seluruh *cluster* hingga menemukan sebuah set kandidat solusi $CAND = \{cand_1, cand_2, \ldots\}$ dengan $score(cand_i) \geq k$.

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 mengeluarkan solusi hasil penalti sebanyak *s* dengan nilai yang minimal. Pada Gambar 3.6 dapat dilihat ilustrasi alur algoritma berbasis *Virtual Records.*

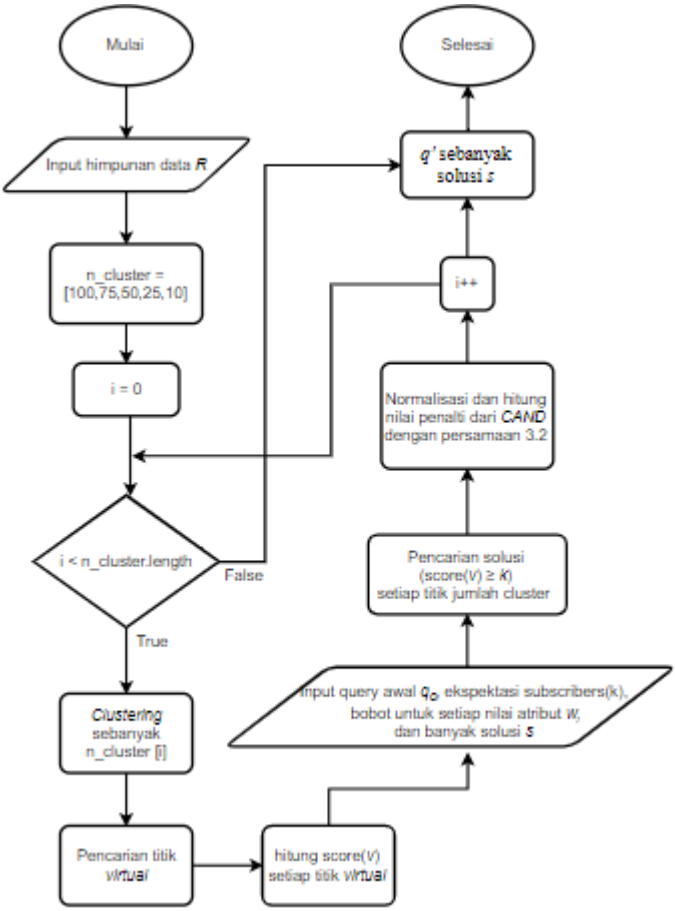
Algoritma berbasis *virtual records* dimulai dari memasukkan himpunan data *R*, yang selanjutnya himpunan data *R* dilakukan proses pembuatan *cluster* sebanyak *n_cluster*, banyaknya *n_cluster* ini adalah sebanyak 5 tahapan yaitu tahapan pertama sebanyak 100 *cluster*, tahapan kedua sebanyak 75 *cluster*, tahapan ketiga sebanyak 50 *cluster*, tahapan keempat sebanyak 25 *cluster*, tahap kelima atau tahap pembentukan *cluster* terakhir adalah sebanyak 10 *cluster*. Dilakukan inisialisasi $i = 0$. Kemudian akan dilakukan pemeriksaan kondisi masih tersedia atau tidak tahapan pada panjang *n_cluster* dengan panjang *n_cluster* sebanyak 5. Ketika masih terdapat tahapan *cluster* yang akan diproses untuk suatu himpunan data, maka akan dilakukan proses *clustering* sebanyak tahapan pada *n_cluster*. *Cluster* yang telah dihasilkan akan dilakukan pencarian titik *virtual* dengan mencari nilai titik minimal dari setiap atribut pada setiap cluster. Titik-titik *virtual* yang telah dihasilkan pada proses *clustering* maka akan dilakukan perhitungan nilai *score(v)* setiap titik *virtual* pada setiap *n_cluster*.

Score(v) merupakan total titik asli yang didominasi oleh titik *virtual* pada himpunan data yang memiliki nilai atribut lebih buruk atau lebih besar dibandingkan dengan nilai atribut pada titik *virtual*. *Score(v)* dapat dihitung dengan mencari seluruh data titik asli himpunan data pada setiap rentang maksimal setiap atribut titik *virtual*. Dilanjutkan dengan memasukkan query awal qo, jumlah

target ekspektasi *subscribers k*, bobot perubahan atau koefisien toleransi perubahan setiap atribut *w*, serta banyak solusi *s* yang akan dikeluarkan sebagai hasil solusi perbaikan *query* q'.

Query yang telah dimasukkan sebelumnya diproses dan mencari data titik *virtual* yang dapat memenuhi jumlah target ekspektasi *subscribers k*. Pencarian solusi dilakukan pada setiap titik *virtual* yang ada pada setiap *n_cluster*. Titik-titik *virtual* yang memiliki nilai *score(v)* lebih tinggi atau minimal sama dengan jumlah target ekspektasi *subscribers k* yang telah dimasukkan sebelumnya akan menjadi *CAND*. *CAND* merupakan kandidat titik *virtual* yang memenuhi jumlah target ekspektasi *subscribers k*. *CAND* kemudian dilakukan proses normalisasi data, serta melakukan penghitungan nilai penalti dari setiap *CAND* yang didapatkan.

Dilanjutkan dengan memeriksa kembali tahapan *n_cluster*, ketika tahapan pada *n_cluster* sudah tidak tersedia atau kosong maka dilakukan pengeluaran hasil *query* q' sebanyak solusi yang telah dimasukkan oleh pengguna.



Gambar 3.6 Ilustrasi Alur Algoritma Berbasis *Virtual Records*

3.2 Algoritma Pembanding *Close Dominance Graph* (CDG)

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 $\boldsymbol{qo}$, preferensi perubahan nilai atribut pada *query* λp, 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.

3.2.1 *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 3.7 dan Gambar 3.8 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

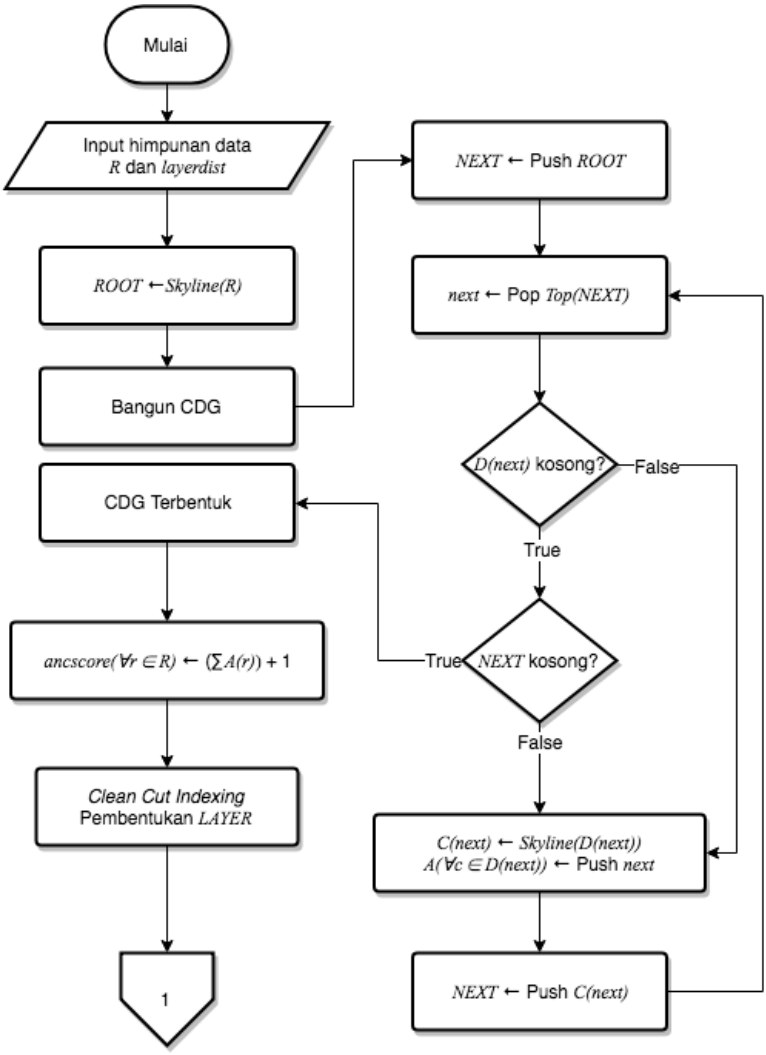
vertex akan menjadi *edge*. Setiap data *r* pada himpunan data *R* memiliki skor *ancscore(r)* yang merupakan jumlah *vertex* lain yang mendominasi *r*. Atau dapat dikatakan skor *ancscore(r)* adalah total jumlah data yang mendominasi *vertex* tersebut ditambah 1. Nilai *ancscore(r)* ini akan menjadi patokan untuk mengetahui berapa banyak data yang yang mendominasi *r*. 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)* >= l dan *r* tidak didominasi data *r'* dimana $\boldsymbol{r} \in \boldsymbol{R}$ dan $\boldsymbol{r'} \in \boldsymbol{LAYER}(\boldsymbol{l})$. Pada algoritma berbasis CDG, salah satu *LAYER* akan menjadi *root* bayangan untuk penelusuran DFS.

3.2.2 Algoritma Berbasis CDG

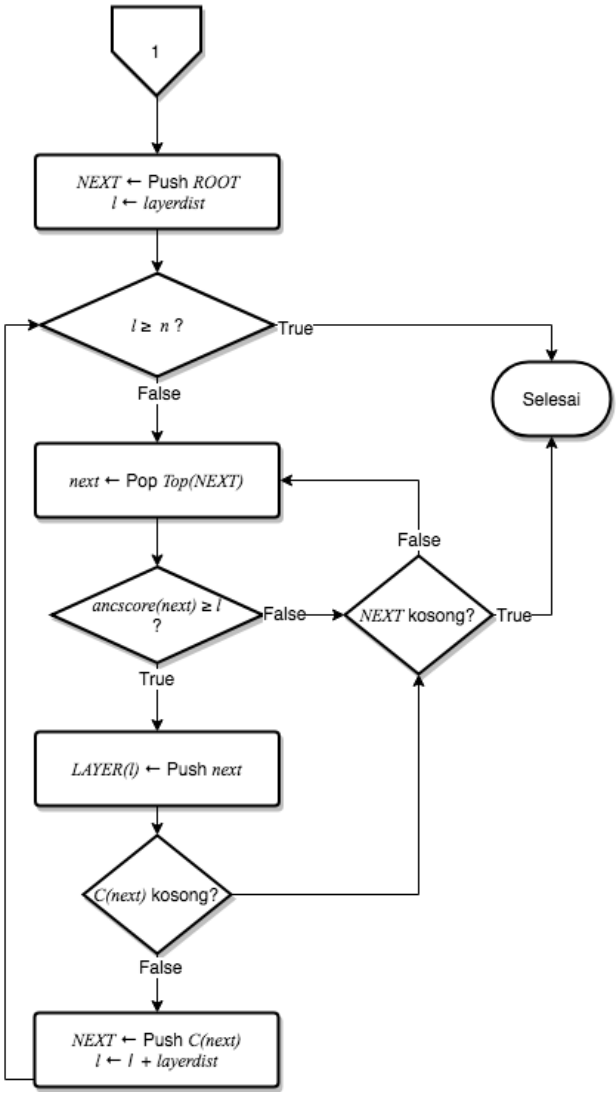
Dalam penelitian ini menggunakan algoritma *Close Dominance Graph* sebagai pembanding dari algoritma utama. Algoritma CDG telah digunakan pada permasalahan *Why Not on Reaching K Subscribers* sebelumnya. Secara garis besar algoritma CDG melakukan pencarian melalui jarak terdekat pada setiap data pada himpunan *R*. Alur program diawali dengan memasukkan $\boldsymbol{q_0}$, himpunan data *R*, bobot nilai preferensi setiap atribut, dan nilai ekspektasi *subscribers k.*

Dilanjutkan dengan melakukan pemeriksaan nilai atribut pada himpunan data untuk mencari data yang akan didominasi. Data mendominasi data lainnya ketika nilai atribut $\boldsymbol{r_j} < \boldsymbol{r_i}$. Kemudian dilanjutkan dengan mencari nilai skor yang memenuhi nilai ekspektasi *subscribers k* untuk dijadikan *CAND*. Ketika *CAND* telah dikumpulkan maka dilakukan perhitungan nilai penalti pada setiap *CAND.*

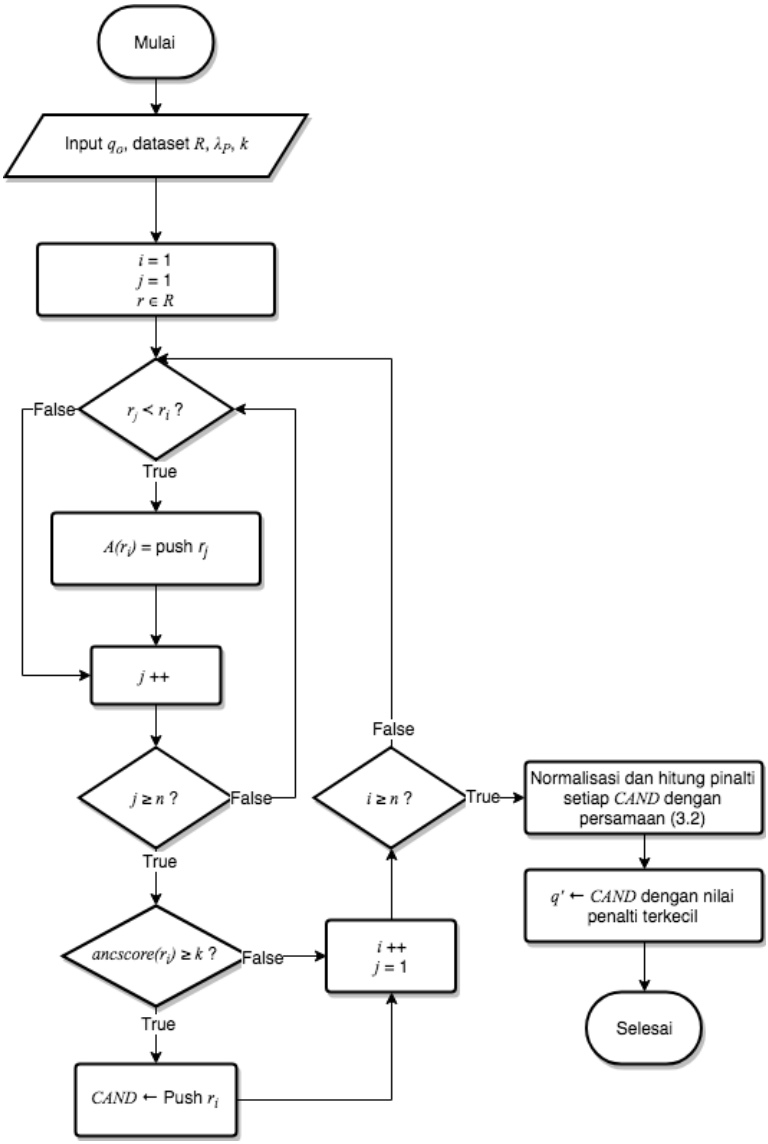
CAND yang memiliki nilai penalti terkecil akan dikeluarkan sebagai hasil $\boldsymbol{q'}$. Pada Gambar 3.9 dapat dilihat ilustrasi alur algoritma berbasis CDG.



Gambar 3. 7 Ilustrasi Alur *Preprocessing* (Bagian 1)



Gambar 3. 8 Ilustrasi Alur *Preprocessing* (Bagian 2)



Gambar 3.9 Ilustrasi Alur Algoritma Berbasis CDG

[Halaman ini sengaja dikosongkan]

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 potongan kode masing-masing fungsi utama.

4.1 Implementasi *Clustering*

Algoritma *Clustering* diimplementasikan sesuai dengan rancangan yang telah dijelaskan pada bab perancangan. *Clustering* secara garis besar bertujuan untuk membentuk beberapa *cluster* pada himpunan data R dengan jumlah data sebanyak n dan ukuran dimensi d menjadi *Virtual Records*. Terdapat 3 tahap utama pada *Clustering*, yaitu tahap pengambilan himpunan data R pada fungsi *load_initial_data*, tahap pembentukan *Virtual Records* pada fungsi *create_virtual_attrs_minimums* dan tahap pencarian jumlah titik data pada himpunan data R yang didominasi oleh *Virtual Records* pada fungsi *get_children*.

4.1.1 Fungsi Load Inital Data

Fungsi *load_initial_data* secara umum memiliki keluaran berupa himpunan data, karena mengambil himpunan data R yang dimasukkan. Pada Gambar 4.1 terlihat pengambilan himpunan data R dilakukan dengan memisahkan *header* dan isi baris pada himpunan data. Potongan kode untuk fungsi *load_initial_data* dapat dilihat pada Gambar 4.1.

```
1.  FUNCTION load_initial_data(filename):
2.    data <- {}
3.    with open(filename) as csv_file:
4.      read_csv <- csv.reader(csv_file, delimiter=”,”)
        first_read <- True
5.          header <- []
6.          FOR row in read_csv:
7.              IF first_read:
8.                  header <- row
9.                  FOR key in header:
10.                     data[key] <- []
11.                 ENDFOR
12.                 first_read <- False
13.             ELSE:
14.                FOR i, label in enumerate(header):
15.                     data[label].append(row[i])
16.             ENDIF
17.         ENDFOR
18.                 ENDFOR
19.    RETURN data
20. ENDFUNCTION
```

Gambar 4.1 *Pseudocode* Fungsi Load Initial Data

Pseudocode fungsi *load initial* data digunakan untuk mengambil data dari sebuah himpunan data *R*. Pada fungsi *load initial* data memiliki parameter berupa nama file yang akan diambil. Baris ke-2 menyimpan data kedalam bentuk *dictionary* Baris ke-3 sampai baris ke-4 menjelaskan mengenai membuka sebuah file berisi himpunan data serta menyimpan setiap judul pada file kedalam list, baris ke-6 sampai baris ke-18 menjelaskan mengenai pembacaan data dan memisahkan judul kedalam list *header* dan menyimpan isi kedalam *dictionary* data. Jadi fungsi *load initial* data secara umum adalah untuk membaca file himpunan data agar dapat diproses pada aplikasi.

4.1.2 Fungsi Create Virtual Attrs Minimums

Pada fungsi *create_virtual_attrs_minimums* pada Gambar 4.2. menjelaskan mengenai tahapan pembuatan *virtual records* melaui proses *clustering*. Proses dimulai dari pencarian nilai *k-means* pada himpunan data *R*. *Clustering* dilakukan menggunakan *library open source* pada python. Ketika *cluster* sudah terbentuk, maka dilanjutkan dengan mencari *virtual records* pada setiap *cluster*. Pencarian titik data *virtual* dilakukan dengan mencari nilai minimal pada setiap dimensi yang ada pada himpunan data *R*.

Dapat dilihat fungsi *create virtual attrs minimums* digunakan untuk mencari nilai titik virtual dari setiap cluster yang akan dibuat yang memiliki parameter berupa *n_cluster* atau masukan tahap-tahap untuk melakukan proses *cluster*, serta data dari himpunan data *R*, dan atribut yang dimiliki. Pada baris ke-2 sampai baris ke-5 dilakukan proses *cluster* menggunakan *library k-means* untuk membentuk *cluster* sebanyak tahap proses *clustering* yang telah dijelaskan sebelumnya. Baris ke-6 sampai baris ke-15 dilakukan proses memasukkan data titik kedalam *cluster* yang telah dibuat secara otomatis. Selanjutnya di baris ke-16 sampai baris ke-28 melakukan proses pencarian titik-titik *virtual* pada setiap *cluster* dengan mencari nilai atribut minimal dari setiap atribut.

Jadi pada fungsi ini secara umum menjelaskan mengenai proses himpunan data untuk dilakukan pembuatan *cluster* serta mendapatkan nilai titik *virtual* pada setiap *cluster* yang telah dihasilkan. Pembuatan *cluster* dilakukan pada tahap awal dengan membuat 100 *cluster* dan selanjutnya dilakukan pencarian titik minimal dari setiap atribut untuk membentuk titik-titik *virtual*. 100 titik *virtual* yang didapatkan selanjutnya digunakan sebagai himpunan data pada tahap *clustering* selanjutnya sampai *n_cluster* kosong.

```
1.  FUNCTION create_virtual_attrs_minimums(n_cluster,
    data, attrs):
2.      df <- DataFrame(data, columns=attrs)
3.      k_means <- KMeans(n_clusters=n_cluster).fit(df)
4.      result <- k_means.predict(df)
5.      clusters <- [SET DEFAULT [SET DEFAULT WITH
                     LENGTH max_range n_cluster]]
6.      FOR i in range(len(result)):
7.          node <- {
8.            id <- data[id][i],
9.            label <- data[label][i]
10.         }
11.         FOR attr_ in attrs:
12.             node[attr_] <- float(data[attr_][i])
13.         ENDFOR
14.         clusters[int(result[i])].append(node)
15.     ENDFOR
16.     virtual_attrs_minimums <- {}
17.     FOR attr_ in attrs:
18.         virtual_attrs_minimums[attr_] <-
19.   [SET DEFAULT ∞ WITH LENGTH max_range n_cluster]
20.     ENDFOR
21.
22.     FOR i, cluster in enumerate(clusters):
23.        FOR node in cluster:
24.            FOR attr_ in attrs:
25.               virtual_attrs_minimums[attr_][i] <-
    min(virtual_attrs_minimums[attr_][i], node[attr_])
26.     ENDFOR
27.         ENDFOR
28.             ENDFOR
29.     RETURN virtual_attrs_minimums
30. ENDFUNCTION
```

Gambar 4.2 *Pseudocode* Fungsi Create Virtual Attrs Minimums

4.1.3 Fungsi Get Children

Pada Gambar 4.3 menjelaskan mengenai proses pencarian jumlah titik-titik data yang didominasi oleh *virtual records.* Secara garis besar fungsi ini memasukkan semua data pada variabel

children, kemudian ketika titik data memiliki nilai atribut yang lebih kecil daripada nilai atribut pada titik *virtual* maka titik data ini tidak akan dihitung sebagai titik yang didominasi oleh titik *virtual*.

```
1.  FUNCTION get_children(n_cluster, data, attrs,
2.                        virtual_attrs_minimums):
3.      children <- [SET DEFAULT [SET DEFAULT WITH
    LENGTH max_range] WITH LENGTH max_range n_cluster]
4.
5.      FOR i in range(n_cluster):
6.          FOR j in range(len(data[id])):
7.              node <- {
8.                 id <- data[id][j],
9.                 label <- data[label][j]
10.             }
11.            FOR attr_ in attrs:
12.                node[attr_] <- float(data[attr_][j])
13.             ENDFOR
14.             data_passed <- True
15.             FOR attr_ in attrs:
16.                 IF data_passed AND node[attr_] <
    virtual_attrs_minimums[attr_][i]:
17.                     data_passed <- False
18.                 ELSE:
19.                    break
20.                 ENDIF
21.             ENDFOR
22.             IF data_passed:
23.                 children[i].append(node)
24.             ENDIF
25.     ENDFOR
26.         ENDFOR
27.     RETURN children
28. ENDFUNCTION
```

Gambar 4.3 *Pseudocode* Fungsi Get Children

Pada fungsi *get children* dilakukan proses menghitung jumlah titik asli yang akan didominasi oleh *virtual records*. Fungsi ini mempunyai parameter *n_cluster* atau banyaknya tahap cluster yang akan dibuat, data dan atribut dari himpunan data, serta titik *virtual* yang telah didapatkan. Pada baris ke-2 dijelaskan untuk menyimpan data *children*. Data ini diambil dari rentang maksimal dari setiap atribut titik *virtual* pada setiap *cluster*. Selanjutnya pada baris ke-5 sampai dengan baris ke-26 dilakukan pemeriksaan atribut pada titik asli yang memiliki nilai atribut lebih besar atau lebih buruk dibandingkan dengan titik *virtual*. Ketika nilai atribut pada titik asli lebih besar maka titik asli ini akan langsung masuk kedalam *children* dari titik *virtual*. Dapat dikatakan titik *virtual* telah mendominasi titik asli.

Setelah mendefinisikan tiga fungsi utama, *Load Data*, *Create Virtual*, dan *Get Children* maka dapat dilanjutkan dengan implementasi algoritma *Clustering*. *Clustering* menerima input berupa himpunan data *R* diambil menggunakan fungsi *Load Data*, yang akan digunakan pada fungsi *Create Virtual* dan titik *virtual* yang telah dihasilkan akan digunakan pada fungsi *Get Children*.

4.2 Implementasi Algoritma Berbasis *Virtual Records*

Rancangan algoritma berbasis *Virtual Records* diimplementasikan sesuai yang telah dijelaskan pada bab desain dan perancangan. Algoritma berbasis *Virtual Records* secara garis besar adalah bertujuan untuk menghasilkan q' agar dapat menjawab permasalahan *Why-Not on Reaching k Subscribers* dengan pendekatan *Virtual Records*. Untuk menunjang algoritma berbasis *Virtual Records* dalam mencari kandidat *refined query* maka dilakukan pembuatan solusi pada setiap *cluster*. Masukan pada fungsi ini adalah berupa nilai data titik *virtual* pada setiap *cluster* dan ekspektasi jumlah *subscribers k.*

4.2.1 Fungsi Generate Target

Subbab ini menjelaskan mengenai input data atribut yang akan dilakukan oleh pengguna dipisahkan dengan separator. Pseudocode untuk fungsi Generate Target dapat dilihat pada Gambar 4.4.

```
1.  FUNCTION generate_target(attr_input, attrs):
2.      attr_value <- input().split(“,”)
3.      FOR i in range(len(attrs)):
4.          target[attrs[i]] <- attr_value[i]
5.      ENDFOR
6.      RETURN target
7.  ENDFUNCTION
```

Gambar 4.4 *Pseudocode* Fungsi Generate Target

Pseudocode fungsi generate target memiliki parameter atribut input dan atribut dari himpunan data. Dapat dilihat pada baris ke-2 sampai baris ke-5 menjelaskan mengenai masukan preferensi produk dari seorang pengguna. Jadi pada fungsi ini menjelaskan mengenai pembuatan target dari preferensi sebuah produk seorang pengguna.

4.2.2 Fungsi Generate Weight

Subbab ini menjelaskan mengenai input bobot atau preferensi yang akan dilakukan oleh pengguna dipisahkan dengan separator. Pseudocode untuk fungsi Generate Weight dapat dilihat pada Gambar 4.5.

```
1.  FUNCTION generate_weight(weight_inp, attrs):
2.          weight <- dict()
3.          weight_list <- weight_inp.split(“,”)
4.          FOR i in range(len(attrs)):
5.            weight[attrs[i]] <- float(weight_list[i])
6.          ENDFOR
7.      RETURN weight
8.  ENDFUNCTION
```

Gambar 4.5 *Pseudocode* Fungsi Generate Weight

Pseudocode pada fungsi *generate weight* memiliki parameter *weight input* dan atribut dari himpunan data. Baris ke-2 sampai baris ke-6 membuat masukan bobot koefisien perubahan pada setiap atribut. Jadi pada fungsi ini menjelaskan proses pembuatan bobot perubahan atau toleransi perubahan pada atribut sebuah produk.

4.2.3 Fungsi Get Cluster Penalties

Pada subbab ini menjelaskan tentang mencari nilai penalti untuk solusi setiap cluster. Pseudocode fungsi Get Cluster Penalties dapat dilihat pada Gambar 4.6. Secara garis besar fungsi ini mengambil nilai pada input data target dan input bobot preferensi dari pengguna, yang selanjutnya akan dilakukan pemeriksaan pada setiap titik virtual yang berada didalam cluster. Atribut data target yang telah diinput dan atribut pada titik virtual dilakukan normalisasi menggunakan algoritma *min-max.* Penalti dapat dicari dengan menghitung total jumlah nilai *absolut* dari jarak perpindahan atribut target ke jarak atribut titik virtual, dan dikalikan dengan nilai bobot yang telah dimasukkan.

```
1.  FUNCTION get_cluster_penalties(virtual_attrs_minimu
    ms, attrs, properties):
2.      target <- generate_target()
3.      weight <- generate_weight()
4.      penalties <- []
5.      FOR i in range all_virtualdata_cluster:
6.         cluster_penalty <- 0.0
7.         FOR j, attr_ in enumerate(attrs):
8.           attr_normalized_before <- normalize target
9.           attr_normalized_after <- normalize virtual
                                      records
10.             abs_attr <- abs(attr_normalized_after -
                                attr_normalized_before)
11.             cluster_penalty += (abs_attr *
12.                                 (1.0 - weight[j]))
13.         ENDFOR
14.         penalties.append(cluster_penalty)
15.     ENDFOR
16.     RETURN penalties
17. ENDFUNCTION
```

Gambar 4.6 *Pseudocode* Fungsi Get Cluster Penalties

Fungsi *get cluster penalties* memiliki parameter titik *virtual*, atribut dari himpunan data serta *properties*, yang memiliki nilai atribut maksimal ($Pmax$) dan nilai atribut minimal ($Pmin$) pada setiap atribut. Pada baris ke-2 dan baris ke-3 memanggil fungsi dari *generate target* dan fungsi dari *generate weight*. Dilanjutkan pada baris ke-4 sampai baris ke-15 dilakukan proses normalisasi data serta mencari nilai penalti pada setiap titik *virtual* di setiap *cluster*. Jadi pada fungsi *get cluster penalties* melakukan proses pembuatan nilai penalti dari setiap titik *virtual* yang ada pada setiap *cluster*.

4.2.4 Fungsi Create N Cluster Solutions

Subbab ini menjelaskan mengenai pencarian solusi yang akan dihasilkan, dengan nilai ekspektasi subscribers yang diinginkan oleh pengguna. Fungsi ini secara garis besar mengambil nilai subscribers *k* dari input pengguna. Kemudian dilanjutkan dengan memeriksa seluruh titik *virtual* pada setiap *cluster* yang memenuhi jumlah subscribers *k*. Ketika score(*v*) >= *subscribers k* selanjutnya titik virtual pada *cluster* akan diambil menjadi kandidat solusi. Pseudocode fungsi Create N Cluster Solutions dapat dilihat pada Gambar 4.7.

```
1.  FUNCTION create_n_cluster_solutions(cluster_childre
    n, penalties, attrs, virtual_attrs_minimums):
2.   subscribers <- k
3.   solutions <- []
4.   FOR i in range LENGTH cluster_children:
5.     solution <- {
6.          subscribers <-LENGTH cluster_children[i],
7.          penalty <- penalties[i],
8.          }
9.          FOR attr_ in attrs:
10.           solution[attr_] <- virtual_attrs_minimums
11.                              [attr_][i]
12.         ENDFOR
13.         IF solution[subscribers] >= subscribers:
14.             solutions.append(solution)
15.         ENDIF
16.    ENDFOR
17.    RETURN solutions
18. ENDFUNCTION
```

Gambar 4.7 *Pseudocode* fungsi Create N Cluster Solutions

Pseudocode fungsi *create n cluster solutions* memiliki parameter *cluster children*, yaitu *children* pada setiap titik *virtual* pada setiap *cluster*, kemudian memiliki parameter nilai penalti, atribut pada himpunan data dan titik *virtual*. Dapat dilihat pada baris ke-2 merupakan jumlah ekspektasi *subscribers k* yang

diinginkan oleh seorang pengguna, kemudian pada baris ke-3 sampai baris ke-16 merupakan proses dalam mencari titik *virtual* yang dapat memenuhi jumlah target ekspektasi *subscribers k* sebagai keluaran hasil *query* q'. Jadi pada fungsi ini menjelaskan mengenai proses pencarian titik *virtual* pada setiap *cluster* yang memiliki *score(v)* yang dapat memenuhi jumlah target ekspektasi *subscribers* dari seorang pengguna.

4.2.5 Fungsi Create Final Solutions

Subbab ini menjelaskan mengenai banyaknya solusi yang akan dikeluarkan berdasarkan jumlah solusi yang diinginkan oleh pengguna. Solusi akan diambil dari titik virtual yang memiliki nilai penalti paling minimal. Pseudocode fungsi Create Final Solutions dapat dilihat pada Gambar 4.8. Pada baris ke-2 menjelaskan mengenai jumlah solusi yang akan dikeluarkan sebagai hasil *query* q' berdasarkan masukan jumlah solusi *s.*

```
1.  FUNCTION create_final_solutions(solutions):
2.      user_defined <- int(sys.argv[4])
3.      n_solutions <- min(LENGTH(solutions),
4.                         user_defined)
5.      final_solutions <- []
6.      counter <- 0
7.      FOR solution in SORTED penalties:
8.          IF counter = n_solutions:
9.              break
10.         ENDIF
11.         final_solutions.append(solution)
12.         counter += 1
13.     ENDFOR
14.     RETURN final_solutions
15. ENDFUNCTION
```

Gambar 4.8 *Pseudocode* fungsi Create Final Solution

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan menjelaskan hasil uji coba yang telah dikerjakan pada sistem serta uji coba dan analisis 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* dengan pendekatan *virtual records.* Pada penelitian ini, 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) Core(TM) i5-8250U CPU @ 1.60GHz (8CPUs)
Memori	8 GB
Arsitektur Prosesor	64-bit

Tabel 5.2 Lingkungan Pengujian Perangkat Lunak

Perangkat Lunak	
Sistem Operasi	Windows 10 Pro
Perangkat Pengembang	Python 2.7

5.2 Data Uji Coba

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

5.2.1 Data Independent (IND)

Data independen (IND) pada penelitian ini merupakan himpunan data sintetis. Persebaran nilai atribut data independen memiliki 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 – 10000.

5.2.2 Data *Anti Correlated* (ANT)

Data *anti correlated* (ANT) pada penelitian ini merupakan himpunan data sintetis. Persebaran nilai atribut memiliki persebaran nilai yang saling bertolak belakang antara satu atribut 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 – 10000.

5.2.3 Data *Forest Covertype* (FC)

Himpunan data *Forest Covertype* (FC) berisi data hasil prediksi tipe tutupan hutan hanya dari variable kartografi (tidak ada data penginderaan jauh). Jenis tutupan hutan yang sebenarnya untuk pengamatan yang diberika (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) [11].

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.

5.3 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 *Clustering* oleh aplikasi
F02	Pengguna dapat memasukkan *query* awal, ekspektasi subscribers, banyak solusi dan bobot atribut untuk dilakukan proses *query refinement*
F03	Aplikasi dapat menampilkan seluruh kandidat hasil *query refinement*
F04	Aplikasi dapat menampilkan hasil *query refinement* dengan nilai penalti minimal sebanyak solusi yang diinginkan pengguna
F05	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. Skenario uji coba diberlakukan ke semua jenis data (independen, *anti correlated,* dan *Forest Covertype*). Berikut adalah beberapa skenario pengujian[2]:

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

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Ekspektasi *Subscribers* (*k*)
C03	30.000	3	5.0000
C04	30.000	3	10.000
C05	30.000	3	15.000

4. Perbandingan jumlah *virtual records* dengan waktu eksekusi dan jumlah memori. Detail skenario dapat dilihat pada Tabel 5.8.

Tabel 5. 8 Skenario Perbandingan Jumlah *Virtual Records* Terhadap Waktu Eksekusi dan Penggunaan Memori

Nomor	Jumlah Data (*n*)	Dimensi (*d*)	Ekspektasi *Subscribers* (*k*)	*Virtual Records*
D01	30.000	3	5.000	100
D02	30.000	3	5.000	75
D03	30.000	3	5.000	50
D04	30.000	3	5.000	25
D05	30.000	3	5.000	10

5.4 Analisis Hasil Uji Coba

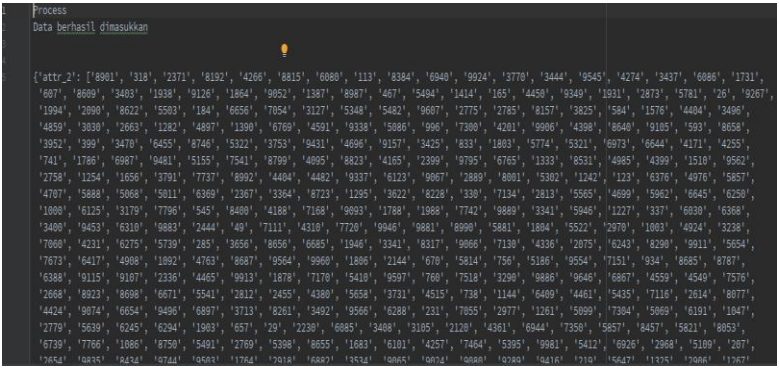
Bagian ini menjelaskan hasil uji coba beserta analisanya. Uji coba yang dilakukan sesuai dengan skenario uji coba yang telah didefinisikan sebelumnya. Hasil uji coba dibagi menjadi dua bagian, yaitu hasil uji coba fungsionalitas dan hasil uji coba performa.

5.4.1 Hasil Uji Coba Fungsionalitas

Uji coba dilakukan sesuai dengan skenario daftar kebutuhan fungsional pada Tabel 5.8. Hasil uji coba fungsionalitas dapat dilihat pada Gambar 5.1, Gambar 5.2, Gambar 5.3, Gambar 5.4 dan Gambar 5.5. Dari hasil uji coba, aplikasi dapat memenuhi semua kebutuhan fungsional.

Tabel 5.9 Hasil Uji Coba Fungsionalitas

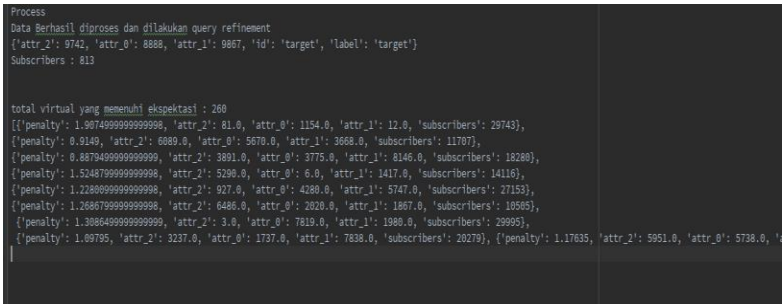
Nomor	Keterangan
F01	Berhasil
F02	Berhasil
F03	Berhasil
F04	Berhasil
F05	Berhasil



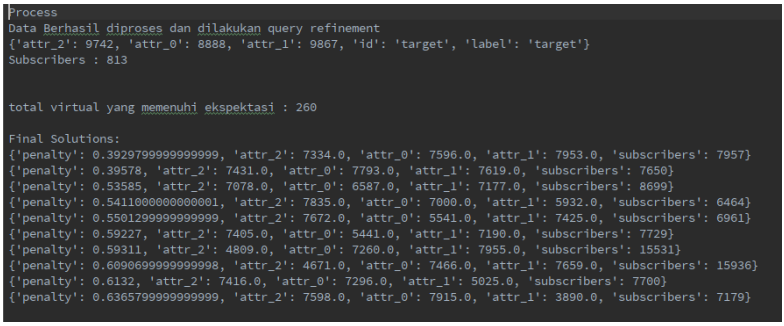
Gambar 5.1 Hasil Uji Coba F01

```
python virtual_records_final.py 9742,8888,9867 5000 0.3,0.3,0.3 10
Process
Input Data Berhasil diproses dan dilakukan query refinement
```

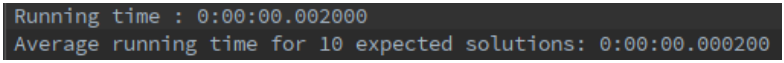
Gambar 5.2 Hasil Uji Coba F02



Gambar 5.3 Hasil Uji Coba F03



Gambar 5.4 Hasil Uji Coba F04



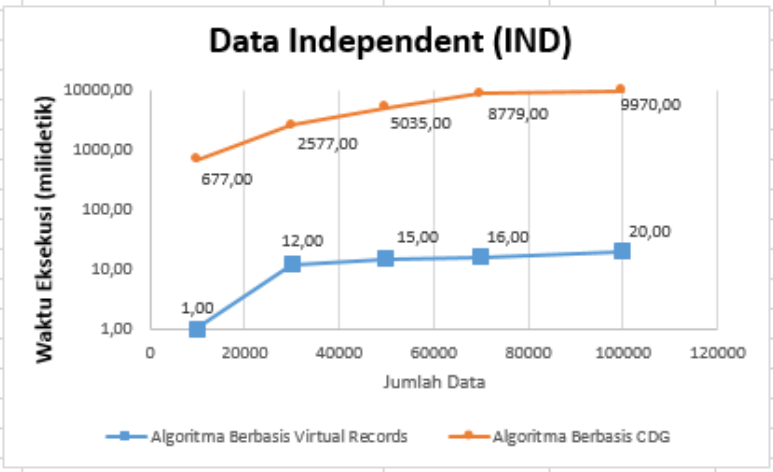
Gambar 5.5 Hasil Uji Coba F05

5.4.2 Hasil Uji Coba Performa

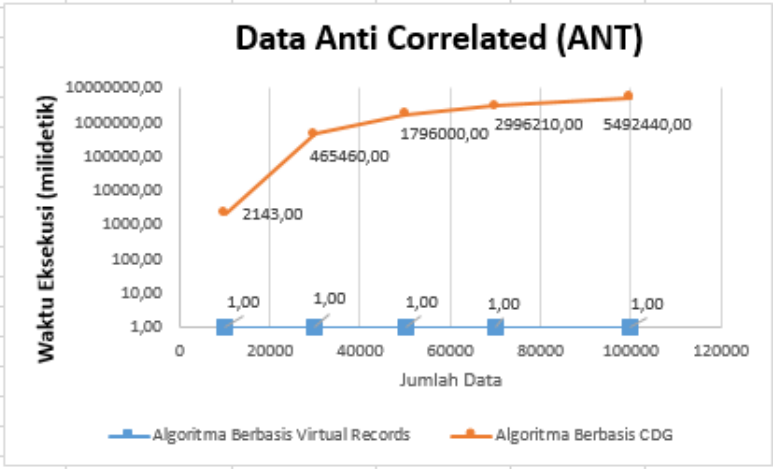
Pada Gambar 5.6, Gambar 5.7, dan Gambar 5.8 ditampilkan grafik hasil uji coba perbandingan waktu eksekusi dengan jumlah data pada semua jenis data untuk mengukur performa masing-masing algoritma. Uji coba dilakukan sesuai dengan skenario pada Tabel 5.5, dimana setiap kasus uji coba menggunakan himpunan data dengan jumlah data yang bervariasi, sementara dimensi dan ekspektasi *subscribers* bersifat tetap (d = 3, k = 5000).

Grafik pada Gambar 5.6, Gambar 5.7 dan Gambar 5.8 menjelaskan bahwa algoritma berbasis *Virtual Records* memiliki waktu eksekusi lebih baik dengan selisih yang signifikan di setiap kasus uji coba. Hal ini membuktikan struktur data *Virtual Records* dan metode *clustering* sangat membantu algoritma berbasis *Virtual Records* dalam hal menangani jumlah data yang bervariasi pada himpunan data IND, ANT dan FC.

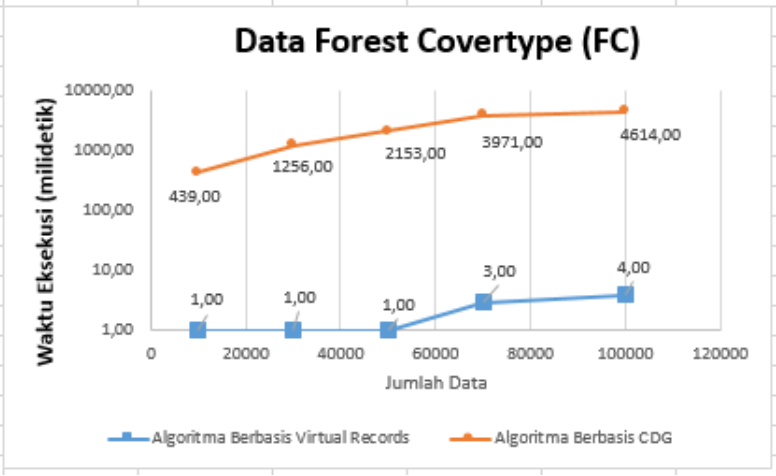
Hasil uji coba pada jumlah data terlihat bahwa algoritma berbasis *Virtual Records* memiliki waktu eksekusi lebih baik, dan signifikan.



Gambar 5.6 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Data Pada IND



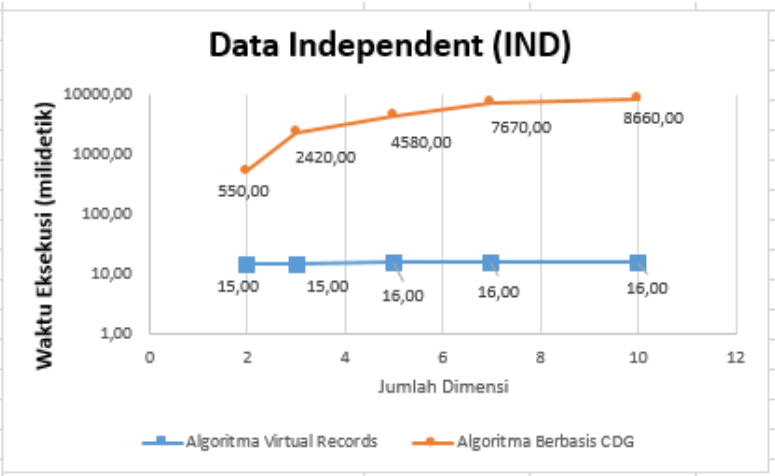
Gambar 5.7 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Data Pada ANT



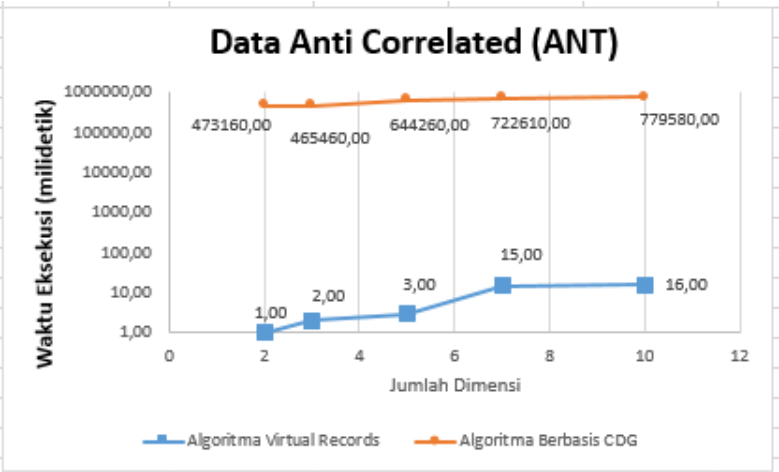
Gambar 5.8 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Data Pada FC

Pada Gambar 5.9, Gambar 5.10, dan Gambar 5.11 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).

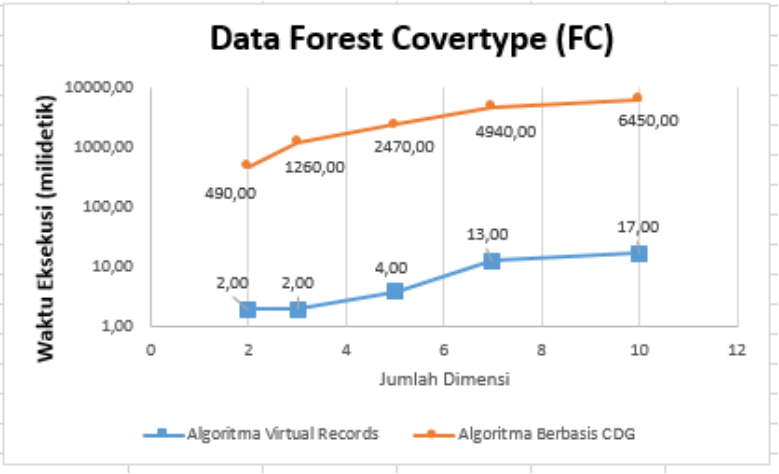
Secara keseluruhan, pada Gambar 5.9 algoritma *Virtual Records* memiliki waktu eksekusi yang cenderung konstan. Pada Gambar 5.9, Gambar 5.10 dan Gambar 5.11 membuktikan bahwa algoritma berbasis *Virtual Records* masih lebih unggul dengan selisih waktu eksekusi yang signifikan dibandingkan dengan algoritma berbasis CDG. Hal ini membuktikan struktur data *Virtual Records* dan metode *clustering* sangat membantu algoritma berbasis *Virtual Records* dalam hal menangani jumlah dimensi yang bervariasi pada himpunan data IND, ANT dan FC. Sama seperti kasus uji coba sebelumnya, dapat dilihat algoritma berbasis *Virtual Records* masih memiliki waktu eksekusi lebih baik, dan signifikan.



Gambar 5.9 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Dimensi Pada IND



Gambar 5.10 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Dimensi Pada ANT



Gambar 5.11 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Dimensi Pada FC

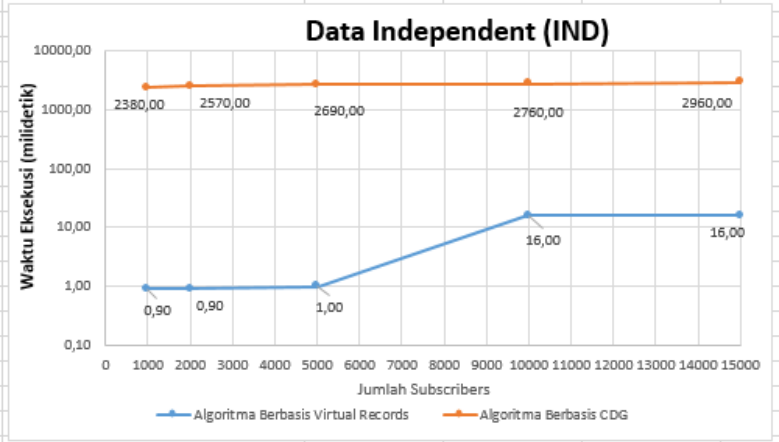
Pada Gambar 5.12, Gambar 5.13, dan Gambar 5.14 ditampilkan grafik hasil uji coba perbandingan waktu eksekusi dengan jumlah ekspektasi *subscribers* pada semua jenis data untuk mengukur performa masing-masing algoritma. Uji coba dilakukan sesuai dengan skenario pada Tabel 5.7, dimana setiap kasus uji coba menggunakan himpunan data dengan jumlah ekspektasi *subscribers* yang bervariasi, sementara jumlah data dan dimensi bersifat tetap (n = 30000, d = 3).

Pada Gambar 5.12 terlihat peningkatan waktu eksekusi pada algoritma *Virtual Records*. Perubahan waktu eksekusi terjadi karena algoritma *Virtual Records* masih sedikit mempertimbangkan banyaknya jumlah *subscribers* yang akan dieksekusi oleh algoritma.

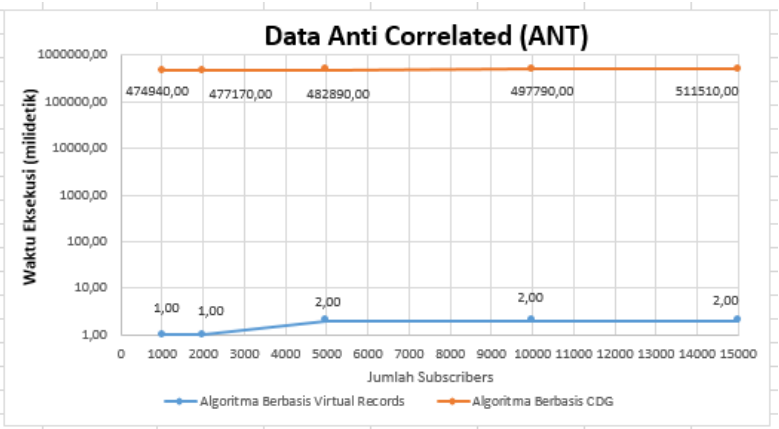
Secara keseluruhan pada Gambar 5.13 dan Gambar 5.14 baik algoritma berbasis *Virtual Records* maupun algoritma CDG cenderung memiliki waktu eksekusi yang konstan. Hal ini membuktikan bahwa jumlah ekspektasi *subscribers* tidak memiliki pengaruh yang signifikan terhadap waktu eksekusi.

Algoritma *Virtual Records* membuktikan bahwa algoritma berbasis *Virtual Records* masih lebih unggul dengan selisih waktu eksekusi yang signifikan dibandingkan dengan algoritma CDG. Hal ini membuktikan struktur data *Virtual Records* dan metode *clustering* sangat membantu algoritma berbasis *Virtual Records* dalam hal menangani ekspektasi *subscribers* yang bervariasi pada himpunan data IND, ANT dan FC.

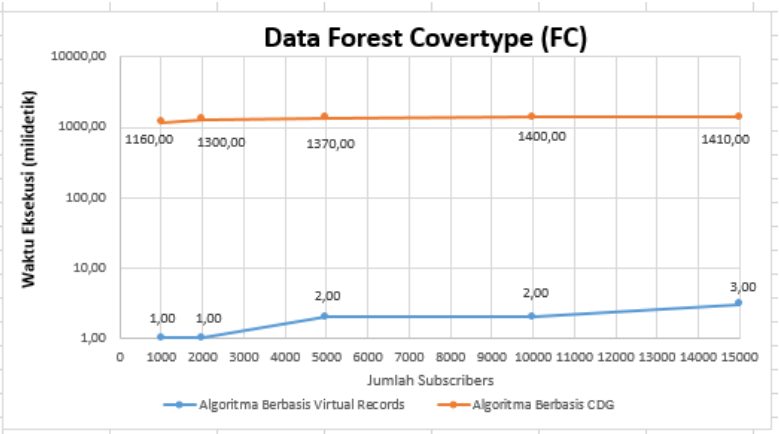
Sama seperti kasus uji coba sebelumnya, dapat dilihat algoritma berbasis *Virtual Records* masih memiliki waktu eksekusi lebih baik, dan signifikan.



Gambar 5.12 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Ekspektasi *Subscribers* Pada IND



Gambar 5.13 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Ekspektasi *Subscribers* Pada ANT



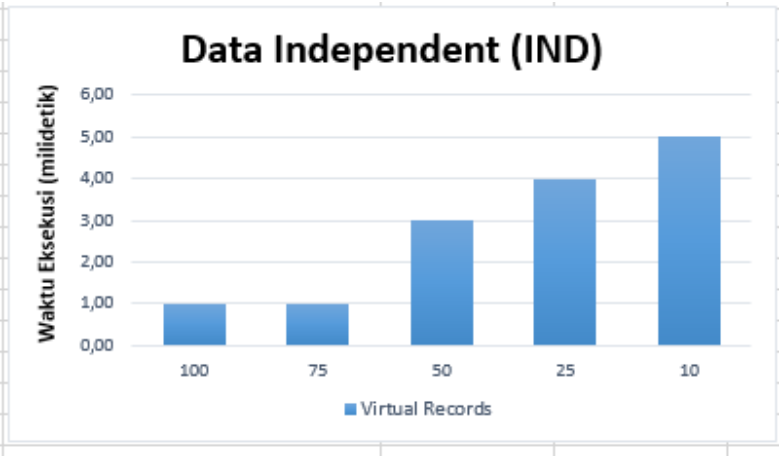
Gambar 5.14 Grafik Perbandingan Waktu Eksekusi dengan Jumlah Ekspektasi *Subscribers* Pada FC

Pada Gambar 5.15, Gambar 5.16, dan Gambar 5.17 ditampilkan grafik hasil uji coba perbandingan waktu eksekusi dengan jumlah *virtual records* pada semua jenis data untuk mengukur performa algoritma. Uji coba dilakukan sesuai dengan skenario pada Tabel 5.8, dimana setiap kasus uji coba menggunakan himpunan data dengan jumlah *virtual records subscribers* yang bervariasi, sementara jumlah data, dimensi serta jumlah *subscribers* bersifat tetap (n = 30000, d = 3, k = 5000).

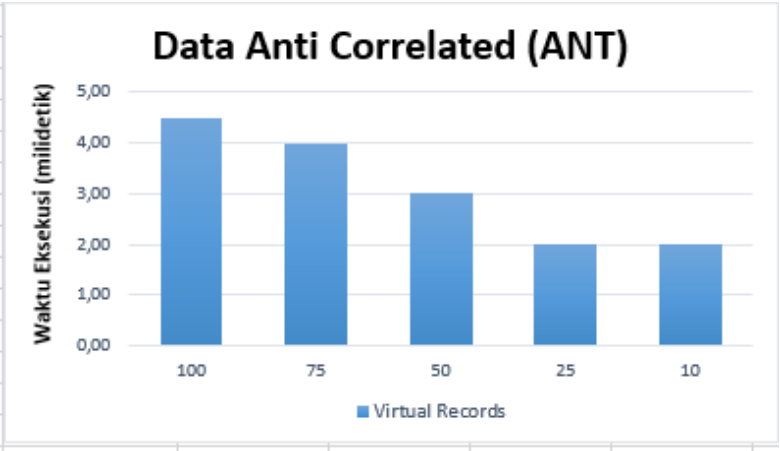
Secara keseluruhan pada Gambar 5.15 dan Gambar 5.17 terlihat peningkatan waktu eksekusi pada algoritma *Virtual Records*. Perubahan waktu eksekusi terjadi karena algoritma *Virtual Records* masih mempertimbangkan banyaknya jumlah tahapan atau banyaknya *virtual records* yang akan dieksekusi oleh algoritma.

Secara keseluruhan pada Gambar 5.16 terlihat penurunan waktu eksekusi pada himpunan data ANT. Penurunan disebabkan karena pada himpunan data ANT tidak mempertimbangkan proses tahapan banyaknya pembentukan *cluster* ataupun banyaknya *virtual records*

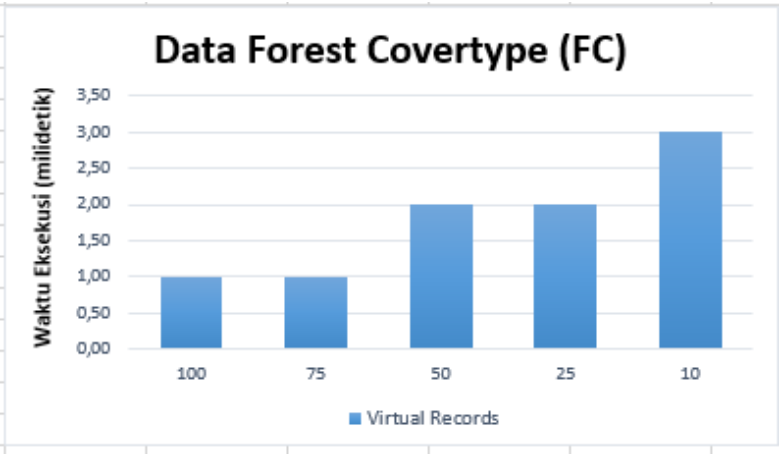
Jadi pada uji coba pada himpunan data IND dan himpunan data FC banyaknya tahapan proses pembentukan *cluster* sangat mempengaruhi waktu eksekusi pada algoritma berbasis *virtual records.* Disamping itu pada himpunan data ANT pembentukan *cluster* atau banyaknya titik *virtual* tidak mempengaruhi waktu eksekusi pada algoritma berbasis *virtual records.*



Gambar 5. 15 Grafik Perbandingan Waktu Eksekusi dengan Jumlah *Virtual Records* Pada IND



Gambar 5. 16 Grafik Perbandingan Waktu Eksekusi dengan Jumlah *Virtual Records* Pada ANT



Gambar 5. 17 Grafik Perbandingan Waktu Eksekusi dengan Jumlah *Virtual Records* Pada FC

Hasil uji coba penggunaan memori dapat dilihat pada Tabel 5.10, Tabel 5.11, Tabel 5.12. Hasil untuk algoritma berbasis *Virtual Records* dilambangkan dengan VR dan algoritma CDG dilambangkan dengan CDG. Skenario yang digunakan sama dengan pengujian waktu eksekusi.

Secara keseluruhan algoritma berbasis *Virtual Records* memiliki konsumsi memori yang cenderung lebih rendah daripada algoritma CDG. Pada skenario jumlah data, dapat dilihat pada Tabel 5.10 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 5.11 bahwa konsumsi memori pada jenis data IND, ANT dan FC cenderung semakin meningkat. Hal ini disebabkan karena jumlah dimensi yang semakin banyak, maka informasi yang disimpan pada struktur data *Virtual Records* juga bertambah sehingga penggunaan memori bertambah.

Pada skenario jumlah ekspektasi *subscribers* dapat dilihat pada Tabel 5.12 sama seperti pengujian waktu eksekusi, penggunaan memori cenderung konstan. Hal ini membuktikan bahwa jumlah ekspektasi *subscribers* tidak mempengaruhi performa algoritma dalam hal penggunaan memori.

Pada skenario jumlah *virtual records* dapat dilihat pada Tabel 5.13 secara garis besar proses pembentukan setiap tahapan *cluster* membutuhkan memori yang lebih besar atau dapat dikatakan setiap pembentukan tahapan *cluster* penggunaan memori selalu mengalami kenaikan Hal ini membuktikan bahwa jumlah *virtual records* mempengaruhi performa algoritma dalam hal penggunaan memori

Tabel 5.10 Perbandingan Penggunaan Memori dengan Jumlah Data

n	***d* = 3, *k* = 5000**					
	IND		**ANT**		**FC**	
	VR	**CDG**	**VR**	**CDG**	**VR**	**CDG**
10K	148.94 MB	165.52 MB	134.77 MB	38.58 MB	176.70 MB	108.99 MB
30K	249.06 MB	555.72 MB	225.80 MB	110.18 MB	328.17 MB	305.04 MB
50K	361.38 MB	963.18 MB	308.84 MB	199.40 MB	496.16 MB	488.54 MB
70K	451.01 MB	1386.16 MB	396.51 MB	291.24 MB	659.10 MB	663.69 MB
100K	596.04 MB	2167.53 MB	534.72 MB	437.35 MB	888.47 MB	917.81 MB

Tabel 5.11 Perbandingan Penggunaan Memori dengan Jumlah Dimensi

d	***n* = 30000, *k* = 5000**					
	IND		**ANT**		**FC**	
	VR	**CDG**	**VR**	**CDG**	**VR**	**CDG**
2	186.24 MB	155.33 MB	183.08 MB	106.57 MB	190.08 MB	139.49 MB
3	246.65 MB	555.72 MB	226.23 MB	110.18 MB	330.14 MB	305.04 MB
5	440.97 MB	1916.18 MB	341.15 MB	135.61 MB	520.53 MB	1340.35 MB
7	601.20 MB	1854.05 MB	418.80 MB	150.96 MB	938.13 MB	1176.21 MB
10	978.62 MB	585.81 MB	552.57 MB	173.40 MB	1165.50 MB	544.63 MB

Tabel 5.12 Perbandingan Penggunaan Memori dengan Jumlah Ekspektasi *Subscribers*

k	***n* = 30000, *d* = 3**					
	IND		**ANT**		**FC**	
	VR	**CDG**	**VR**	**CDG**	**VR**	**CDG**
1K	241.02 MB	555.61 MB	223.16 MB	110.23 MB	326.34 MB	110.23 MB
2K	249.45 MB	555.56 MB	225.35 MB	110.17 MB	328.34 MB	110.17 MB
5K	244.85 MB	555.83 MB	227.55 MB	110.21 MB	328.71 MB	110.21 MB
10K	250.40 MB	555.63 MB	226.93 MB	110.18 MB	329.50 MB	110.18 MB
15K	247.28 MB	555.61 MB	228.32 MB	110.22 MB	330.17 MB	110.22 MB

Tabel 5. 13 Perbandingan Penggunaan Memori dengan Jumlah *Virtual Records*

VR	***n* = 30000, *d* = 3, *k* = 5000**		
	IND	**ANT**	**FC**
100	242.88 MB	221.18 MB	326.32 MB
75	247.83 MB	218.48 MB	332.04 MB
50	254.11 MB	226.32 MB	337.52 MB
25	251.87 MB	229.18 MB	345.23 MB
10	247.73 MB	228.50 MB	330.01 MB

Dari hasil uji coba waktu eksekusi dan penggunaan memori dapat ditarik beberapa analisa dan kesimpulan. Pada algoritma berbasis *Virtual Records*, semakin besar penggunaan memori maka waktu eksekusi menjadi lebih singkat. Hal ini disebabkan oleh tahap *clustering*, 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. Hal ini juga berlaku sebaliknya, semakin sedikit penggunaan memori maka waktu eksekusi menjadi lebih lama.

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 *pre-indexing* dilakukan dengan struktur data *Virtual Records* untuk membantu proses *query refinement* dengan mengumpulkan semua himpunan data menjadi beberapa kelompok himpunan data atau *cluster.* Dilanjutkan dengan Proses pencarian nilai titik *virtual* pada setiap *cluster* adalah dengan menemukan nilai minimal dari setiap dimensi yang tersedia, dari semua nilai minimal di setiap dimensi pada setiap *cluster* maka akan terbentuk sebuah titik *virtual*.
2. Proses pencarian kandidat *refined query* dilakukan dengan mencari nilai atribut titik-titik *virtual* yang memiliki skor ekspektasi yang sama bahkan lebih tinggi. Selanjutnya mencari jarak perubahan yang dilakukan ketika ingin memperbaiki sebuah *query* agar mencapai angka ekspektasi yang diinginkan dengan menghitung nilai penalti untuk setiap *cluster*.
3. Algoritma berbasis *Virtual Records* melakukan perbaikan *query* agar mencapai ekspektasi pengguna (*query refinement*) mencari semua nilai atribut titik-titik virtual di setiap *cluster* yang memiliki *score* ekspektasi yang lebih dari sama dengan nilai 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 sebanyak jumlah solusi yang diinginkan pengguna.

4. 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 *pre-indexing* dapat diselesaikan lebih cepat.
2. Penelitian dapat dikembangkan dengan kondisi himpunan data yang dinamis, mengalir, tidak pasti, atau memiliki nilai atribut yang hilang.

DAFTAR PUSTAKA

1. H. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi dan C. Yu, "Making Database Systems Usable," *SIGMOD,* pp. 13-24, 200.
2. Wira. "Desain dan Implementasi Aplikasi untuk Menjawab Permasalahan *Why-Not on Reaching k Subscribers*" [skripsi]. Surabaya (ID): Institut Teknologi Sepuluh Nopember. 2018.
3. 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.
4. S. Borzsonyi, D. Kossmann dan K. Stocker, "The Skyline Operator," *ACM Trans. Database System.,* vol. 25 , no. 2, p. 129–178, 2000.
5. G. Koutrika, E. Pitoura dan K. Stefanidis, "Preference-Based Query Personalization," Advanced Query Processing, vol. I, no. 36, pp. 57-81, 2013
6. Z. Mustaffa and Y. Yusof, "A Comparision of Normalization Techiques in Predicting Dengue Outbreak," *Conference on Business and Economics Research,* vol. 1, 2011.
7. Madhulatha, T.S. 2012. *An Overview on Clustering Method***.** Warangal : IQSR.
8. Abraham, Henry, dan S. Sudarshan. 2011. *DATABASE SYSTEM CONCEPTS, SIXTH EDITION*. New York: McGraw-Hill.
9. 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 19 Desember 2019].

10. 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.
11. 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 20 Desember 2019].

LAMPIRAN 1

Kode sumber

```
1.  import csv
2.  import sys
3.  import datetime
4.  import threading
5.  import psutil
6.  import os
7.  from pandas import DataFrame
8.  from sklearn.cluster import KMeans
9.
10. def load_initial_data(filename):
11.     data = {}
12.     with open(filename) as csv_file:
13.         read_csv = csv.reader(csv_file, delimiter='
    ,')
14.         first_read = True
15.         header = []
16.
17.         for row in read_csv:
18.             if first_read:
19.                 header = row
20.                 for key in header:
21.                     data[key] = []
22.                 first_read = False
23.             else:
24.                 for i, label in enumerate(header):

25.                     data[label].append(row[i])
26.
27.     return data
28.
29. def create_virtual_attrs_minimums(n_cluster, data,
    attrs):
30.     df = DataFrame(data, columns=attrs)
31.     k_means = KMeans(n_clusters=n_cluster).fit(df)

32.     result = k_means.predict(df)
33.     clusters = [[] for _ in range(n_cluster)]
```

```
34.
35.     for i in range(len(result)):
36.         node = {
37.             "id": data["id"][i],
38.             "label": data["label"][i]
39.         }
40.         for attr_ in attrs:
41.             node[attr_] = float(data[attr_][i])
42.
43.         clusters[int(result[i])].append(node)
44.
45.     virtual_attrs_minimums = {}
46.     for attr_ in attrs:
47.         virtual_attrs_minimums[attr_] = [sys.maxint
                        for _ in range(n_cluster)]
48.
49.     for i, cluster in enumerate(clusters):
50.         for node in cluster:
51.             for attr_ in attrs:
52.                 virtual_attrs_minimums[attr_][i] =
    min(virtual_attrs_minimums[attr_][i], node[attr_])
53.
54.     return virtual_attrs_minimums
55.
56. def get_children(n_cluster, data, attrs, virtual_at
    trs_minimums):
57.     children = [[] for _ in range(n_cluster)]
58.
59.     for i in range(n_cluster):
60.         for j in range(len(data["id"])):
61.             node = {
62.                 "id": data["id"][j],
63.                 "label": data["label"][j]
64.             }
65.
66.             for attr_ in attrs:
67.                 node[attr_] = float(data[attr_][j])
68.
69.             data_passed = True
70.             for attr_ in attrs:
```

```
71.                 if data_passed and node[attr_] < vi
    rtual_attrs_minimums[attr_][i]:
72.                     data_passed = False
73.                 else:
74.                     break
75.
76.             if data_passed:
77.                 children[i].append(node)
78.
79.     return children
80.
81. def normalize_attr(min, max, x):
82.     return (float(1-min)+float(x))/(float(1-
    min)+float(max))
83.
84. def get_properties(attrs, data):
85.     properties = {}
86.     for attr_ in attrs:
87.         properties[attr_] = {
88.             "min": sys.maxint,
89.             "max": 0
90.         }
91.
92.     for attr_ in attrs:
93.         for value in data[attr_]:
94.             properties[attr_]["min"] = min(
95.     properties[attr_]["min"], int(float(value)))
96.     properties[attr_]["max"] = max(properties
97.     [attr_]["max"], int(float(value)))
98.
99.     return properties
100.
101.        def basic_target():
102.            t = dict()
103.            t["id"] = "target"
104.            t["label"] = "target"
105.            return t
106.
107.        def generate_target(attr_input, attrs):
108.            target = basic_target()
109.            attr_value = [int(value) for value in at
    tr_input.split(',')]
```

```
110.
111.             for i in range(len(attrs)):
112.                 target[attrs[i]] = attr_value[i]
113.
114.             return target
115.
116.         def score_target(attr_input, attrs, data):
117.             data_target = generate_target(attr_input
    , attrs)
118.             print data_target
119.
120.             counter = 0
121.             for i in range(len(data["id"])):
122.                 data_passed = True
123.                 for attr_ in attrs:
124.                     if data_passed and int(float(dat
    a[attr_][i])) < data_target[attr_]:
125.                         data_passed = False
126.                     else:
127.                         break
128.
129.                 if data_passed:
130.                     counter += 1
131.
132.             return counter
133.
134.         def generate_weight(weight_inp, attrs):
135.             try:
136.                 weight = dict()
137.                 weight_list = weight_inp.split(',')

138.                 for i in range(len(attrs)):
139.                     weight[attrs[i]] = float(weight_
    list[i])
140.             except (IndexError, ValueError) as e:
141.             weight = dict()
142.             weight_default = 1.0 / float(len(attrs))

143.                 for i in range(len(attrs)):
144.                    weight[attrs[i]] = weight_default

145.
146.             return weight
```

```
147.
148.       def get_cluster_penalties(virtual_attrs_mini
    mums, attrs, properties):
149.           keys = virtual_attrs_minimums.keys()
150.           len_data = len(virtual_attrs_minimums[ke
    ys[0]])
151.
152.           target = [int(val) for val in sys.argv[1
    ].split(",")]
153.           weight = [float(val) for val in sys.argv
    [3].split(",")]
154.
155.           penalties = []
156.           for i in range(len_data):
157.               cluster_penalty = 0.0
158.               for j, attr_ in enumerate(attrs):
159.                   attr_normalized_before =
160.       normalize_attr(min=properties[attr_]["min"],
          max=properties[attr_]["max"], x=target[j])
161.
162.                   attr_normalized_after =
163.       normalize_attr(min=properties[attr_]["min"],
          max=properties[attr_]["max"],x=
164.       virtual_attrs_minimums[attr_][i])
165.
166.                   abs_attr_ = abs(attr_normalized_
    after - attr_normalized_before)
167.
168.                   cluster_penalty += (abs_attr_ *
    (1.0 - weight[j]))
169.               penalties.append(cluster_penalty)
170.
171.           return penalties
172.
173.       def create_n_cluster_solutions(cluster_child
    ren, penalties, attrs, virtual_attrs_minimums):
174.           threshold = int(sys.argv[2])
175.
176.         solutions = []
177.         for i in range(len(cluster_children)):
178.           solution = {
```

```
179.           "subscribers": len(cluster_children[i]),

180.                   "penalty": penalties[i],
181.               }
182.               for attr_ in attrs:
183.                   solution[attr_] = virtual_attrs_
    minimums[attr_][i]
184.
185.               if solution["subscribers"] >= thresh
    old:
186.                   solutions.append(solution)
187.
188.           return solutions
189.
190.       def create_final_solutions(solutions):
191.           user_defined = int(sys.argv[4])
192.           n_solutions = min(len(solutions),
193.                             user_defined)
194.           final_solutions = []
195.           counter = 0
196.
197.           for solution in sorted(solutions,
198.                     key=lambda x: x["penalty"]):
199.               if counter == n_solutions:
200.                   break
201.
202.               final_solutions.append(solution)
203.               counter += 1
204.
205.           return final_solutions
206.
207.       def show_final_solutions(final_solutions):
208.           print "\nFinal Solutions:"
209.           for solution in final_solutions:
210.               print solution
211.
212.       def show_precompute_time(
213.            n_cluster, start_precompute_time):
214.           end_precompute_time =
215.                           datetime.datetime.now()
216.           precompute_time = (end_precompute_time-
217.                           start_precompute_time)
```

```
218.           print ">> Waktu proses Pre-
    Compute Time {} Cluster : ".format(n_cluster) + str
    (precompute_time)
219.
220.       def main():
221.          data = load_initial_data("datasets/
222.          independent/dataset_30000_3.csv")
223.          attrs = filter(lambda key: key.
224.          startswith('attr_'), list(data.keys()))
225.          target_subs_score = score_target(
226.           sys.argv[1], attrs, data)
227.           print "Subscribers : " +
228.                           str(target_subs_score)
229.
230.           properties = get_properties(attrs, data)

231.           n_clusters = [100, 75, 50, 25, 10]
232.
233.           dummy_time = datetime.datetime.now()
234.           mutable_obj = {
235.               'sum_running_time':
236.                (dummy_time - dummy_time),
237.               'solutions': []
238.           }
239.
240.           for n_cluster in n_clusters:
241.               def execute_clustering():
242.                   start_precompute_time =
243.                    datetime.datetime.now()
244.                   virtual_points =
245.                   create_virtual_attrs_minimums(
246.                   n_cluster, data, attrs)
247.                   cluster_children = get_children(
248.           n_cluster,data,attrs,
249.           virtual_attrs_minimums=virtual_points)
250.                   show_precompute_time(n_cluster,
                          start_precompute_time=
251.                       start_precompute_time)
252.
253.                   s_time = datetime.datetime.now()
```

```
254.                     cluster_penalties = get_cluster_
    penalties(virtual_points, attrs, properties)
255.                     n_cluster_solutions = create_n_c
    luster_solutions(cluster_children=cluster_children,
256.
                     penalties=cluster_penalties,
257.
                     attrs=attrs,
258.
                     virtual_attrs_minimums=
259.                                  virtual_points)
260.                     e_time = datetime.datetime.now()
261.                     r_time = e_time - s_time
262.
263.                     mutable_obj['sum_running_time']
264.                                         += r_time
265.                     mutable_obj['solutions'] += n_cl
    uster_solutions
266.
267.                 t = threading.Thread(target=execute_
    clustering)
268.                 t.start()
269.                 t.join()
270.
271.             solutions = mutable_obj['solutions']
272.             sum_running_time = mutable_obj['sum_runn
    ing_time']
273.
274.             print '\n'
275.             print 'total virtual yang memenuhi ekspe
    ktasi : ' + str(len(solutions))
276.
277.             start_running_time = datetime.datetime.n
    ow()
278.             final_solutions = create_final_solutions
    (solutions)
279.             end_running_time = datetime.datetime.now
    ()
280.             running_time = (end_running_time - start
    _running_time)
281.
```

```
282.            sum_running_time += running_time
283.
284.            show_final_solutions(final_solutions)
285.
286.            print '\nRunning time : {}'.format(str(s
    um_running_time))
287.            print 'Average running time for {} expec
    ted solutions: {}'.format(sys.argv[4], str(sum_runn
    ing_time/ int(sys.argv[4])))
288.
289.            process = psutil.Process(os.getpid())
290.            mem_usage = float(process.memory_info().
    rss) / 1000000.0
291.            print "Memory usage: " + str(mem_usage)
    + " MB "
292.
293.        if __name__ == '__main__':
294.            main()
```

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Bagus Dharma Iswara, akrab dipanggil Bagus, lahir pada tanggal 6 Nopember 1997 di Bekasi. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Departemen Informatika Institut Teknologi Sepuluh Nopember. Memiliki beberapa hobi antara lain bermain dan mendengarkan musik. Selama masa kuliah penulis mengambil bidang minat Komputasi Berbasis Jaringan (KBJ). Selama menempuh pendidikan di kampus, penulis juga aktif dalam beberapa kegiatan organisasi kemahasiswaan, antara lain sebagai Staff Pengembangan Profesi Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ajaran 2016/2017. Penulis juga aktif dalam organisasi Tim Pembina Kerohanian Hindu sebagai Wakil Kepala Departemen Media Kreatif pada tahun ajaran 2016/2017. Penulis juga aktif dalam kepanitiaan Schematics, antara lain sebagai Staff REEVA yaitu kegiatan penutupan atau konser dari Schematics pada tahun ajaran 2017/2018. Selama menempuh pendidikan di kampus, penulis juga aktif berkontribusi dalam kegiatan kepanitiaan Pusat Validasi ITS sebagai panita validasi pada tahun 2018.