



TUGAS AKHIR - KI141502

**STUDI PERMASALAHAN K-MOST PROMISING PRODUCTS
BERBASIS INTERVAL WAKTU PADA DATA MULTIDIMENSI
DENGAN SERIAL WAKTU**

HAFARA FIRDAUSI
NRP 05111540000043

Dosen Pembimbing 1
Bagus Jati Santoso, S.Kom., Ph.D.

Dosen Pembimbing 2
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

Halaman ini sengaja dikosongkan



TUGAS AKHIR - KI141502

**STUDI PERMASALAHAN K-MOST PROMISING PRODUCTS
BERBASIS INTERVAL WAKTU PADA DATA MULTIDIMENSI
DENGAN SERIAL WAKTU**

HAFARA FIRDAUSI
NRP 05111540000043

Dosen Pembimbing 1
Bagus Jati Santoso, S.Kom., Ph.D.

Dosen Pembimbing 2
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

Halaman ini sengaja dikosongkan



UNDERGRADUATE THESES - KI141502

**OBTAINING K-MOST PROMISING PRODUCTS BASED ON
TIME INTERVAL ON MULTIDIMENSIONAL TIME SERIES
DATA**

HAFARA FIRDAUSI
NRP 05111540000043

Supervisor 1
Bagus Jati Santoso, S.Kom., Ph.D.

Supervisor 2
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

INFORMATICS DEPARTMENT
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

Halaman ini sengaja dikosongkan

**STUDI PERMASALAHAN K-MOST PROMISING
PRODUCTS BERBASIS INTERVAL WAKTU PADA DATA
MULTIDIMENSI DENGAN SERIAL WAKTU**

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada

Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:

Hafara Firdausi

NRP: 05111540000043

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Bagus Jati Santoso, S.Kom., Ph.D.
NIP: 198611252018031001

.....
(Pembimbing 1)

Henning Titi Ciptaningtyas, S.Kom., M.Kom.
NIP: 198407082010122004

.....
(Pembimbing 2)

**SURABAYA
JUNI 2019**

Halaman ini sengaja dikosongkan

STUDI PERMASALAHAN K-MOST PROMISING PRODUCTS BERBASIS INTERVAL WAKTU PADA DATA MULTIDIMENSI DENGAN SERIAL WAKTU

Nama : HAFARA FIRDAUSI
NRP : 0511154000043
Departemen : Informatika FTIK-ITS
Pembimbing I : Bagus Jati Santoso, S.Kom., Ph.D.
Pembimbing II : Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Abstrak

Kemajuan ilmu pengetahuan dan teknologi telah mempengaruhi cara produsen dalam melakukan bisnis, yaitu dengan memanfaatkan data preferensi pelanggan untuk mendapatkan informasi, misalnya untuk mencari produk yang paling banyak diminati oleh pelanggan sehingga produsen dapat memilih produk dengan tepat untuk menarik lebih banyak pelanggan dan bertahan lama di pasar global.

Saat ini, sudah ada komputasi yang dapat menyelesaikan permasalahan tersebut. k-Most Promising Products (k-MPP) adalah sebuah strategi pemilihan produk dengan melakukan pencarian k-produk yang paling banyak diminati oleh pelanggan. Namun, hasil pencarian k-produk tidak mungkin tetap dari waktu ke waktu. Oleh karena itu, interval waktu adalah salah satu faktor yang harus dipertimbangkan dalam proses kueri karena sangat berpengaruh terhadap hasil pencarian. Permasalahan ini kemudian didefinisikan dalam penelitian ini sebagai "k-Most Promising Products berbasis interval waktu (k-MPPTI)".

Pada penelitian ini akan dirancang dan diimplementasikan struktur data dan algoritme yang tepat untuk menyelesaikan permasalahan tersebut. Data yang digunakan adalah data multidimensi, sehingga menggunakan struktur data yang memiliki fitur indexing, yaitu

array. Algoritme k-MPPTI menggunakan dua tipe kueri skyline, yaitu dynamic skyline dan reverse skyline. Selain itu, algoritme k-MPPTI juga mengimplementasikan teknik komputasi paralel supaya pemrosesan data menjadi lebih cepat.

Hasil uji coba menunjukkan bahwa algoritme k-MPPTI dapat memberikan hasil dengan waktu eksekusi ... kali lebih cepat dan penggunaan memori ... kali lebih hemat dibandingkan dengan algoritme konvensional Brute Force.

Kata Kunci: *Strategi pemilihan produk, Kueri, Dynamic skyline, Reverse skyline, Interval waktu*

OBTAINING K-MOST PROMISING PRODUCTS BASED ON TIME INTERVAL ON MULTIDIMENSIONAL TIME SERIES DATA

Name : HAFARA FIRDAUSI
NRP : 05111540000043
Major : Informatics Department Faculty of IT-ITS
Supervisor I : Bagus Jati Santoso, S.Kom., Ph.D.
Supervisor II : Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Abstract

Kemajuan ilmu pengetahuan dan teknologi telah mempengaruhi cara produsen dalam melakukan bisnis, yaitu dengan memanfaatkan data preferensi pelanggan untuk mendapatkan informasi, misalnya untuk mencari produk yang paling banyak diminati oleh pelanggan sehingga produsen dapat memilih produk dengan tepat untuk menarik lebih banyak pelanggan dan bertahan lama di pasar global.

Saat ini, sudah ada komputasi yang dapat menyelesaikan permasalahan tersebut. k-Most Promising Products (k-MPP) adalah sebuah strategi pemilihan produk dengan melakukan pencarian k-produk yang paling banyak diminati oleh pelanggan. Namun, hasil pencarian k-produk tidak mungkin tetap dari waktu ke waktu. Oleh karena itu, interval waktu adalah salah satu faktor yang harus dipertimbangkan dalam proses kueri karena sangat berpengaruh terhadap hasil pencarian. Permasalahan ini kemudian didefinisikan dalam penelitian ini sebagai "k-Most Promising Products berbasis interval waktu (k-MPPTI)".

Pada penelitian ini akan dirancang dan diimplementasikan struktur data dan algoritme yang tepat untuk menyelesaikan permasalahan tersebut. Data yang digunakan adalah data multidimensi, sehingga menggunakan struktur data yang memiliki fitur indexing, yaitu

array. Algoritme k-MPPTI menggunakan dua tipe kueri skyline, yaitu dynamic skyline dan reverse skyline. Selain itu, algoritme k-MPPTI juga mengimplementasikan teknik komputasi paralel supaya pemrosesan data menjadi lebih cepat.

Hasil uji coba menunjukkan bahwa algoritme k-MPPTI dapat memberikan hasil dengan waktu eksekusi ... kali lebih cepat dan penggunaan memori ... kali lebih hemat dibandingkan dengan algoritme konvensional Brute Force.

Keywords: *Product selection strategy, Query, Dynamic skyline, Reverse skyline, Time interval*

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah Swt. atas pertolongan dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

STUDI PERMASALAHAN K-MOST PROMISING PRODUCTS BERBASIS INTERVAL WAKTU PADA DATA MULTIDIMENSI DENGAN SERIAL WAKTU.

Penelitian Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya.

Dengan selesainya Tugas Akhir ini, diharapkan apa yang telah dikerjakan oleh penulis dapat memberikan manfaat bagi perkembangan ilmu pengetahuan, terutama di bidang teknologi informasi, serta bagi diri penulis sendiri selaku peneliti.

Penulis juga mengucapkan banyak terima kasih kepada semua pihak yang telah memberikan dukungan, baik secara langsung maupun tidak langsung, selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

1. Ibu, Bapak, kedua Adik, Akbar dan Alam, serta segenap keluarga yang senantiasa memberikan perhatian, dukungan, serta kasih sayang yang menjadi semangat dan motivasi bagi diri penulis untuk menyelesaikan Tugas Akhir.
2. Bapak Bagus Jati Santoso, S.Kom., Ph.D. selaku dosen pembimbing yang telah banyak meluangkan waktu untuk membimbing dan memberikan ilmu, saran, dan motivasi

kepada penulis baik selama menempuh masa kuliah maupun selama pengerjaan Tugas Akhir ini.

3. Ibu Henning Titi Ciptaningtyas, S.Kom., M.Kom. selaku dosen pembimbing yang telah memberikan ilmu dan masukan kepada penulis.
4. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku Kepala Departemen Informatika ITS pada masa pengerjaan Tugas Akhir, Bapak Radityo Anggoro, S.Kom., M.Sc. selaku koordinator Tugas Akhir, dan segenap dosen dan karyawan Informatika yang telah memberikan ilmu, waktu, dan pengalamannya.
5. Teman-teman *Sempol Bunda*, Ajeng, Salma, Napik, Bela, dan Yola, yang telah menemani dan mewarnai masa-masa perkuliahan penulis sejak jaman mahasiswa baru.
6. Seluruh teman-teman Laboratorium Arsitektur dan Jaringan Komputer (AJK), Mas Syukron, Mas Fatih, Nahda, Satria, Awan, Mas Penyok, Fuad, Didin, Hana, Raldo, Aguel, Khawari, Tamtam, Haura, Lia, Sulton, Mail, Yoga, dan Fawwaz, yang telah menemani, mengganggu, dan membantu penulis selama mengerjakan Tugas Akhir di lab.
7. Teman-teman *Penguasa Kosan*, Ajeng, Salma, Napik, Prames, Kikik, Balqis, Tije, Nilam, dan Rini, yang pernah mengajarkan cara bersenang-senang.
8. Emak kos terbaik, Mak Ju, atas segala bantuannya selama ini dan teman-teman kosan 36, Jakiya, Mutek, Marisa, Mbak Tatak, Alya, Firda, dan Anca.
9. Teman-teman *Data Engineers*, Hana dan Rio, sebagai teman seperjuangan dan seperbimbingan.
10. Seluruh teman-teman TC 2015 yang tidak bisa penulis sebutkan satu persatu namanya.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian

hari. Semoga melalui Tugas Akhir ini Penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, Juni 2019

Hafara Firdausi

Halaman ini sengaja dikosongkan

DAFTAR ISI

SAMPUL	i
LEMBAR PENGESAHAN	vii
ABSTRAK	ix
ABSTRACT	xi
KATA PENGANTAR	xiii
DAFTAR ISI	xvii
DAFTAR TABEL	xxi
DAFTAR GAMBAR	xxiii
DAFTAR KODE SUMBER	xxv
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Batasan Masalah	3
1.4 Tujuan	4
1.5 Manfaat	4
1.6 Metodologi	4
1.7 Sistematika Penulisan	6
BAB II TINJAUAN PUSTAKA	9
2.1 Daftar Notasi	9
2.2 Data	10
2.2.1 Data Multidimensi	11
2.2.2 Data Serial Waktu	11
2.2.3 Data Multidimensi dengan Serial Waktu	12
2.3 <i>Skyline</i>	13
2.4 Dominansi Dinamis	15
2.5 <i>Dynamic Skyline</i>	17
2.6 <i>Reverse Skyline</i>	18
2.7 Kueri <i>k</i> -Most Promising Products (<i>k</i> -MPP)	20
2.7.1 <i>Uniform Product Adoption</i> (UPA)	20

2.7.2	Strategi Pemilihan Produk	22
2.8	Python	22
2.9	Flask	23
BAB III ANALISIS DAN PERANCANGAN SISTEM	25	
3.1	Daftar Notasi	25
3.2	Analisis Sistem	27
3.2.1	Analisis Permasalahan	27
3.2.2	Deskripsi Umum Sistem	27
3.2.3	Fungsi Sistem	28
3.2.4	Analisis Kebutuhan Fungsional	28
3.3	Perancangan Sistem	30
3.3.1	Struktur Data	30
3.3.2	Algoritme Utama	34
3.3.3	Algoritme <i>Brute Force</i>	53
3.3.4	Arsitektur Aplikasi	53
BAB IV IMPLEMENTASI	55	
4.1	Lingkungan Implementasi	55
4.2	Implementasi Algoritme k-MPPTI	57
4.2.1	Implementasi Algoritme <i>Data Precomputing</i>	57
4.2.2	Implementasi Algoritme <i>Query Processing</i>	57
4.3	Implementasi Algoritme Pembanding (<i>Brute Force</i>)	57
4.4	Implementasi Antarmuka Pengguna	57
BAB V UJI COBA DAN EVALUASI	59	
5.1	Lingkungan Uji Coba	59
5.2	Data Uji Coba	59
5.2.1	Data <i>Independent</i> (IND)	59
5.2.2	Data <i>Anticorrelated</i> (ANT)	60
5.2.3	Data <i>Correlated</i> (COR)	60
5.3	Skenario Uji Coba	60
5.3.1	Skenario Uji Coba <i>Performance</i>	61
BAB VI KESIMPULAN DAN SARAN	73	
6.1	Kesimpulan	73
6.2	Saran	73

DAFTAR PUSTAKA	75
Lampiran A Kode Sumber	77
BIODATA PENULIS	109

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Tabel 2.1	Daftar Notasi (1)	9
Tabel 2.2	Contoh Data <i>Time Series</i>	12
Tabel 2.3	Contoh Data Multidimensi dengan Serial Waktu (1)	12
Tabel 2.4	Contoh Data Multidimensi dengan Serial Waktu (2)	13
Tabel 2.5	Contoh <i>Dataset</i> (a) Produk P dan (b) Preferensi Pelanggan C	16
Tabel 3.1	Daftar Notasi (2)	25
Tabel 3.2	Kebutuhan Fungsional	29
Tabel 3.3	<i>Key</i> dari <i>Data Storage</i>	32
Tabel 3.4	Atribut dari <i>Event Queue</i>	33
Tabel 3.5	Contoh <i>Dataset</i> (a) Produk P dan (b) Preferensi Pelanggan C	37
Tabel 3.6	<i>Event Queue</i>	38
Tabel 3.7	Hasil Perhitungan Kontribusi Pasar Berbasis Interval Waktu	52
Tabel 3.8	Hasil Perhitungan Kontribusi Pasar Berbasis Interval Waktu (2)	52
Tabel 5.1	Atribut dari objek	60

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Gambar 2.1	Titik Skyline dari Data Produk pada Tabel 2.5	14
Gambar 2.2	(a) Komputasi <i>Dynamic Skyline</i> dari Pelanggan c_4 dan (b) <i>Dynamic Skyline</i> dari Pelanggan c_{10}	18
Gambar 2.3	Komputasi <i>Reverse Skyline</i> dari Produk p_8	19
Gambar 3.1	Struktur Data <i>Dictionary</i> Produk	31
Gambar 3.2	Struktur Data <i>Dictionary</i> Pelanggan	31
Gambar 3.3	Contoh <i>Pandora Box</i> dari <i>Dataset 3.5</i>	34
Gambar 3.4	Diagram Alur Algoritme k-MPPTI	35
Gambar 3.5	lini Masa Data Produk dan Pelanggan	37
Gambar 3.6	<i>Event</i> dalam Lini Masa Data Produk dan Pelanggan	38
Gambar 3.7	Diagram Alur Proses <i>Product Insertion</i>	41
Gambar 3.8	Diagram Alur Proses <i>Product Deletion</i>	42
Gambar 3.9	Diagram Alur Proses <i>Customer Insertion</i>	44
Gambar 3.10	Diagram Alur Proses <i>Customer Deletion</i>	45
Gambar 3.11	Diagram Alur Komputasi <i>Reverse Skyline</i>	47
Gambar 3.12	Pemrosesan Paralel	50
Gambar 3.13	Ilustrasi Interval Waktu Pencarian	51
Gambar 3.14	Perhitungan Kontribusi Pasar pada <i>Pandora Box</i> Berdasarkan Interval Waktu Pencarian	52
Gambar 3.15	Perancangan Arsitektur Aplikasi	54
Gambar 4.1	Algoritme <i>DetermineGSP</i>	56
Gambar 5.1	Pengaruh jumlah sel terhadap waktu komputasi tiap operasi dalam satuan detik	62
Gambar 5.2	Pengaruh jumlah sel grid terhadap penggunaan memori dalam satuan megabita	63
Gambar 5.3	Pengaruh jumlah objek terhadap waktu komputasi tiap operasi dalam satuan detik	64

Gambar 5.4 Pengaruh jumlah objek terhadap penggunaan memori dalam satuan megabita	65
Gambar 5.5 Pengaruh jumlah <i>instance</i> terhadap waktu komputasi tiap operasi dalam satuan detik	66
Gambar 5.6 Pengaruh jumlah <i>instance</i> terhadap penggunaan memori dalam satuan megabita	67
Gambar 5.7 Pengaruh d_e terhadap waktu komputasi tiap operasi dalam satuan detik	68
Gambar 5.8 Pengaruh d_e terhadap penggunaan memori dalam satuan megabita	69
Gambar 5.9 Pengaruh dimensi terhadap waktu komputasi tiap operasi dalam satuan detik	70
Gambar 5.10 Pengaruh dimensi terhadap penggunaan memori dalam satuan megabita	71
Gambar 5.11 Pengaruh jenis data terhadap waktu komputasi tiap operasi dalam satuan detik	72

DAFTAR KODE SUMBER

Kode Sumber 1.1	Kode sumber kelas <i>Edge</i>	77
Kode Sumber 1.2	Kode sumber kelas <i>Node</i>	77
Kode Sumber 1.3	Kode sumber kelas <i>Object</i>	77
Kode Sumber 1.4	Kode sumber pemrosesan utama	77
Kode Sumber 1.5	Kode sumber penentuan <i>turning-point</i> .	82
Kode Sumber 1.6	Kode sumber graf sementara	87
Kode Sumber 1.7	Kode sumber penentuan grid	88
Kode Sumber 1.8	Kode sumber titik dua dimensi	92
Kode Sumber 1.9	Kode sumber kotak dua dimensi	94
Kode Sumber 1.10	Kode sumber visualisasi HMTL	99
Kode Sumber 1.11	Kode sumber visualisasi utama dengan JavaScript	99
Kode Sumber 1.12	Kode sumber visualisasi dengan CSS .	107

Halaman ini sengaja dikosongkan

BAB I

PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Kemajuan ilmu pengetahuan dan teknologi telah membawa dampak yang cukup besar di bidang bisnis, termasuk mempengaruhi cara produsen dalam melakukan bisnis secara lebih efisien. Produsen dapat mengumpulkan data preferensi pelanggan terhadap produk dan fitur produk dari data penjualan mereka. Selain itu, maraknya penggunaan situs web untuk menjual produk secara *online* juga memungkinkan produsen mendapatkan data preferensi pelanggan terhadap fitur produk lain.

Data preferensi pelanggan yang terkumpul dapat dimanfaatkan untuk mendapatkan informasi-informasi penting yang menguntungkan, misalnya untuk mencari produk apa saja yang paling banyak diminati oleh pelanggan sehingga produsen dapat menentukan produk mana yang harus dipilih untuk strategi *targeted marketing* supaya lebih tepat sasaran dan bertahan lama di pasar global.

Saat ini, sudah ada komputasi yang dapat menyelesaikan masalah pemilihan produk dengan memanfaatkan data preferensi pelanggan. *k-Most Promising Products (k-MPP)* [1] adalah sebuah strategi pemilihan produk dengan melakukan pencarian k produk yang paling banyak diminati oleh pelanggan.

Komputasi *k-MPP* menggunakan dua tipe kueri *skyline*, yaitu *dynamic skyline* [2] dan *reverse skyline* [3]. Kueri *dynamic skyline* digunakan untuk mengambil data produk terbaik berdasarkan sudut pandang pelanggan, sedangkan kueri *reverse skyline* digunakan untuk mengambil data pelanggan potensial berdasarkan sudut pandang produsen.

Kemudian muncul sebuah pertanyaan, “*Apakah produk yang paling banyak diminati pelanggan akan selalu sama dari waktu ke waktu?*”. Tentu saja para produsen lain tidak akan berdiam diri. Produk-produk baru akan terus bermunculan seiring dengan berjalannya waktu, sehingga produk yang paling diminati pelanggan pun ikut berubah karena adanya produk-produk baru yang dapat mengungguli produk sebelumnya.

Sebagai contoh, *smartphone A* adalah produk yang paling banyak diminati oleh pelanggan pada bulan Januari hingga Juni 2018, namun pada bulan Juli posisinya tergeser oleh *smartphone B* yang fitur-fiturnya lebih disukai oleh pelanggan. Dari ilustrasi tersebut, dapat diketahui bahwa interval waktu juga merupakan faktor penting yang harus dipertimbangkan dalam proses pencarian *k* produk yang paling banyak diminati oleh pelanggan karena sangat berpengaruh terhadap hasil pencarian.

Pertanyaan baru yang mungkin akan diajukan oleh produsen atau analis pemasaran adalah “*k produk apa saja yang paling banyak diminati oleh pelanggan pada bulan Januari hingga Desember 2018?*”. Dalam hal ini, bulan Januari hingga Desember 2018 disebut dengan interval waktu kueri dan data produk yang berbasis interval waktu disebut dengan data *time series* atau serial waktu [4].

Untuk menjawab pertanyaan tersebut, dibutuhkan pendekatan lain untuk menyelesaikan kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data

multidimensi dengan serial waktu. Algoritme yang dibangun juga mengimplementasikan teknik komputasi paralel supaya pemrosesan data menjadi lebih cepat [1].

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana desain dan implementasi struktur data dan algoritme untuk menjawab kueri *k-Most Promising Products* (*k-MPP*) berbasis interval waktu pada data multidimensi dengan serial waktu?
2. Bagaimana kinerja dari struktur data dan algoritme yang dibangun untuk menjawab kueri *k-Most Promising Products* (*k-MPP*) berbasis interval waktu pada data multidimensi dengan serial waktu?
3. Bagaimana strategi yang optimal untuk meningkatkan efisiensi komputasi *k-Most Promising Products* (*k-MPP*) berbasis interval waktu pada data multidimensi dengan serial waktu?

1.3 Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan sebagai berikut:

1. Struktur data dan algoritme dalam komputasi *k-Most Promising Products* (*k-MPP*) hanya dapat menyimpan dan memproses nilai numerik.
2. Implementasi struktur data dan algoritme menggunakan bahasa pemrograman Python.
3. *Dataset* yang digunakan adalah data asli dan sintetis.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Merancang dan mengimplementasikan struktur data dan algoritme untuk menjawab kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktu.
2. Mengevaluasi kinerja dari struktur data dan algoritme yang dibangun untuk menjawab kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktu.
3. Mengimplementasikan strategi yang optimal untuk meningkatkan efisiensi komputasi *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktu.

1.5 Manfaat

Manfaat yang diharapkan dari penulisan Tugas Akhir ini adalah untuk mengetahui struktur data dan algoritme yang tepat untuk menjawab kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktusecara optimal dan efisien.

Selain itu, Tugas Akhir ini juga diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi karena algoritme ini dapat digunakan dalam berbagai hal, khususnya bagi produsen untuk membuat bisnisnya menjadi lebih baik dan tepat sasaran.

1.6 Metodologi

Metodologi yang digunakan dalam penggerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir

Tahap awal untuk memulai penggerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir yang berisi gagasan untuk menjawab kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktu. Proposal ini berisi tentang deskripsi pendahuluan dari Tugas Akhir yang akan dibuat, terdiri atas hal yang menjadi latar belakang diajukannya usulan Tugas Akhir, rumusan masalah yang diangkat, batasan masalah, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu, dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir.

2. Studi literatur

Pada tahap ini dilakukan pencarian informasi dan literatur mengenai metode yang dapat digunakan dalam merancang dan mengimplementasikan struktur data dan algoritme untuk menjawab kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktu. Informasi-informasi tersebut bisa didapatkan dari buku, jurnal, maupun internet.

3. Analisis dan perancangan perangkat lunak

Pada tahap ini dilakukan analisis dan perancangan struktur data dan algoritme yang digunakan untuk menjawab kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktuberdasarkan literatur yang telah dipelajari.

4. Implementasi perangkat lunak

Pada tahap ini dilakukan implementasi atau realiasi dari hasil analisis dan perancangan struktur data dan algoritme yang telah dibuat ke dalam bentuk program.

5. Uji coba dan evaluasi

Pada tahap ini dilakukan uji coba dari struktur data dan algoritme yang telah diimplementasikan. Pengujian akan

dilakukan dengan dua cara, yaitu:

(a) Pengujian waktu eksekusi (*runtime*)

Pengujian yang berfokus pada waktu eksekusi dari struktur data dan algoritme yang dibangun untuk menjawab kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktu.

(b) Pengujian penggunaan memori (*memory usage*)

Pengujian yang berfokus pada konsumsi memori dari struktur data dan algoritme yang dibangun untuk menjawab kueri *k-Most Promising Products (k-MPP)* berbasis interval waktu pada data multidimensi dengan serial waktu.

Setelah dilakukan uji coba, maka dilakukan evaluasi terhadap kinerja struktur data dan algoritme yang telah diimplementasikan, dengan harapan dapat diperbaiki ke depannya.

6. Penyusunan buku Tugas Akhir

Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi pengerjaan dan laporan hasil pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

Berikut adalah sistematika penulisan buku Tugas Akhir:

1. BAB I: PENDAHULUAN

Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan Tugas Akhir.

2. BAB II: TINJAUAN PUSTAKA

Bab ini berisi dasar teori mengenai permasalahan dan algoritme penyelesaian yang digunakan dalam Tugas Akhir

3. BAB III: ANALISIS DAN PERANCANGAN SISTEM

Bab ini berisi analisis dan perancangan struktur data dan algoritme yang digunakan dalam penyelesaian permasalahan.

4. BAB IV: IMPLEMENTASI

Bab ini berisi implementasi berdasarkan analisis dan perancangan struktur data dan algortime yang telah dilakukan pada tahap analisis dan perancangan sistem.

5. BAB V: UJI COBA DAN EVALUASI

Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.

6. BAB VI: PENUTUP

Bab ini berisi kesimpulan dan saran yang didapat dari hasil uji coba dan evaluasi yang telah dilakukan.

Halaman ini sengaja dikosongkan

BAB II

TINJAUAN PUSTAKA

Bab ini menjelaskan dasar teori yang digunakan dalam analisis, perancangan, dan implementasi struktur data dan algoritme untuk menjawab permasalahan *k-Most Promising Products* (*k*-MPP) berbasis interval waktu pada data multidimensi dengan serial waktu yang diangkat dalam Tugas Akhir ini.

2.1 Daftar Notasi

Tabel 2.1 menunjukkan daftar notasi yang digunakan untuk memudahkan beberapa penjelasan pada bab ini berikut dengan deskripsinya.

Tabel 2.1 Daftar Notasi (1)

Notasi	Deskripsi
P	<i>Dataset</i> produk
C	<i>Dataset</i> pelanggan (preferensi pelanggan)
D	$P \cup C$
ob	Sebuah objek data pada D
$ob_1 \prec ob_2$	Objek data ob_1 mendominasi ob_2
$ob_1 \prec_{ob_3} ob_2$	Objek data ob_1 mendominasi ob_2 berdasarkan ob_3
p	Sebuah produk dalam P , $p \in P$
c	Seorang pelanggan dalam C , $c \in C$
d	Jumlah dimensi pada D

Notasi	Deskripsi
i	Dimensi ke-1, ..., d
j	Timestamp ke-1, 2, ..., dst
O	<i>Orthant</i> atau daerah pada komputasi <i>reverse skyline</i>
m	<i>Midpoint</i> antar produk pada komputasi <i>reverse skyline</i>
$DSL(c)$	Hasil <i>dynamic skyline</i> dari pelanggan c
$RSL(p)$	Hasil <i>reverse skyline</i> dari produk p
$Pr(c, p P)$	Probabilitas produk p dibeli oleh pelanggan c
$E(C, p P)$	Kontribusi pasar p
$E(C, P' P)$	Kontribusi pasar subset P' dari P
$k - MPP$	k -Most Promising Products

2.2 Data

Data merupakan elemen yang esensial dalam sebuah sistem informasi. Data adalah informasi faktual (seperti pengukuran atau statistik) yang digunakan sebagai dasar untuk analisis, diskusi, maupun perhitungan [11].

Meski begitu, data mentah tidaklah berarti dan harus diproses terlebih dahulu supaya menghasilkan informasi yang bermanfaat. Sehingga, dibutuhkanlah sebuah algoritme pemrosesan data yang menerima data sebagai *input*, kemudian memprosesnya menjadi informasi tertentu sesuai dengan kebutuhan pengguna dan mengeluarkannya sebagai *output*.

2.2.1 Data Multidimensi

Model data multidimensi adalah sebuah cara pandang yang melihat data dari berbagai sudut pandang atau dimensi. Model data ini memiliki struktur yang disesuaikan untuk mengoptimalkan analisis berdasarkan data dari *relational database* dan diolah sehingga informasi dapat dikategorikan. Model data multidimensi merupakan variasi dari model relasional yang menggunakan struktur multidimensi untuk menyusun data dan menjelaskan relasi antar data.

Struktur multidimensi merepresentasikan dimensi-dimensi data dalam bentuk kubus. Jika sebuah data multidimensi memiliki lebih dari tiga dimensi, maka disebut dengan *hypercube* [5]. Dalam implementasinya, data multidimensi disajikan dalam bentuk *array* multidimensi yang masing-masing nilai dalam selnya dapat diakses menggunakan sebuah indeks.

Data multidimensi banyak digunakan untuk analisis. Selama beberapa tahun terakhir, konsep data multidimensi telah menjadi hal yang fundamental dalam sistem pengambil keputusan, seperti sistem *data warehouse* [5].

2.2.2 Data Serial Waktu

Data *time series* atau serial waktu adalah nilai-nilai suatu variabel yang berurutan menurut waktu. Data *time series* memiliki nilai dan *timestamp*, sehingga data diurutkan berdasarkan waktu atau *timestamp*-nya.

Pada Tabel 2.2, diberikan contoh sebuah data *time series* S . Supaya sederhana, kita asumsikan bahwa *timestamp* adalah bilangan bulat positif. Nilai $s_1 \in S$ pada *timestamp* j dinotasikan sebagai $s_1[j]$, sehingga *time series* s_1 jika ditulis secara berurutan menjadi $s_1[1], s_1[2], \dots$, dan seterusnya [4].

Tabel 2.2 Contoh Data *Time Series*

id	timestamp				
	1	2	3	4	5
s_1	8	2	5	10	12
s_2	14	4	10	7	8
s_3	15	6	11	7	3
s_4	3	8	12	9	13
s_5	15	9	10	2	7

2.2.3 Data Multidimensi dengan Serial Waktu

Untuk menjawab permasalahan yang diangkat pada Tugas Akhir ini, data yang digunakan merupakan penggabungan dari kedua jenis data di atas, yaitu data multidimensi dengan serial waktu.

Data multidimensi dengan serial waktu adalah data *multi-attribute* yang memiliki *timestamp* dan berurutan menurut waktu. Pada Tabel 2.3, diberikan contoh sebuah data produk yang memiliki nilai atribut dan *timestamp*.

Tabel 2.3 Contoh Data Multidimensi dengan Serial Waktu (1)

id	timestamp									
	1		2		3		4		5	
	d_1	d_2	d_1	d_2	d_1	d_2	d_1	d_2	d_1	d_2
p_1	2	8	2	8	2	8	2	8	2	8
p_2	-	-	-	-	-	-	4	10	4	10
p_3	6	11	6	11	6	11	-	-	-	-
p_4	-	-	-	-	-	-	-	-	8	12
p_5	9	10	9	10	9	10	9	10	-	-

Contoh data pada Tabel 2.3 sekilas hampir sama dengan data pada Tabel 2.2, namun memiliki atribut atau dimensi lebih dari satu. Jika asumsinya nilai pada atribut data selalu tetap, maka data pada Tabel 2.3 dapat ditulis menjadi Tabel 2.4. Timestamp yang banyak dan berurutan dapat ditulis menjadi interval waktu (*timestamp in - timestamp out*), dinotasikan dengan $[i : j]$. Interval waktu menggambarkan bahwa sebuah data memiliki waktu hidup tertentu.

Tabel 2.4 Contoh Data Multidimensi dengan Serial Waktu (2)

id	ts_in	ts_out	dim1	dim2
p_1	1	8	2	8
p_2	4	14	4	10
p_3	1	3	6	11
p_4	5	15	8	12
p_5	1	4	9	10

Data yang digunakan pada Tugas Akhir ini adalah dataset produk P dan pelanggan C . Ilustrasinya, setiap produk $p \in P$ memiliki waktu kapan ia pertama kali diproduksi dan kapan ia tidak diproduksi lagi, sedangkan setiap pelanggan $c \in C$ memiliki waktu kapan ia lahir dan kapan ia meninggal dunia.

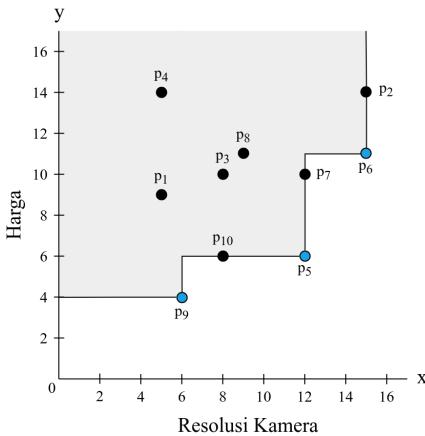
2.3 Skyline

Komputasi *skyline* telah menarik perhatian yang cukup besar dari peneliti sejak diperkenalkan pada komunitas basis data [6], terutama mengenai metode progresif yang dapat mengembalikan hasil kueri dengan cepat tanpa perlu membaca keseluruhan data [2]. Tujuan dari komputasi *skyline* adalah mencari data yang “menarik” dari suatu himpunan data [6], yaitu data yang tidak didominasi oleh data lain atau data yang paling unggul.

Diberikan dataset produk P yang setiap datanya direpresentasikan sebagai titik d -dimensi. Sebuah titik p_1 dikatakan mendominasi titik lain p_2 , dinotasikan dengan $p_1 \prec p_2$, jika nilai p_1 tidak lebih besar dari p_2 pada semua dimensi dan ada nilai p_1 yang lebih kecil dari p_2 minimal pada satu dimensi. Secara matematika, relasi $p_1 \prec p_2$ dapat terbentuk jika dan hanya jika:

- (a) $p_1^i \leq p_2^i, \forall i \in [1, \dots, d]$
- (b) $p_1^i < p_2^i, \exists i \in [1, \dots, d]$

Misalnya, seseorang ingin mencari produk *smartphone* terbaik, yaitu *smartphone* yang memiliki harga termurah dan memiliki resolusi kamera terbesar. Pada Tabel 2.5, diberikan data produk P yang memiliki atribut resolusi kamera (dim_1) dan harga (dim_2). Setiap datanya direpresentasikan sebagai titik pada bidang dua dimensi, yakni sumbu x adalah resolusi kamera dan sumbu y adalah harga *smartphone*.



Gambar 2.1 Titik Skyline dari Data Produk pada Tabel 2.5

Berdasarkan Gambar 2.1, produk *smartphone* yang terbaik

adalah p_5 , p_6 , dan p_9 karena tidak ada titik yang lebih baik dari titik-titik tersebut pada semua dimensi, sedangkan produk p_{10} tidak dapat menjadi *skyline* karena didominasi oleh produk p_5 pada dimensi x . Begitu juga produk p_7 yang didominasi p_5 pada dimensi y dan produk p_2 yang didominasi p_6 pada dimensi y . Produk p_5 , p_6 , dan p_9 disebut dengan titik *skyline* atau *skyline point*.

Saat ini, komputasi *skyline* telah banyak digunakan sebagai operator pengambil keputusan multikriteria dan perencanaan bisnis [7]. Ada beberapa pengembangan dari komputasi *skyline*, seperti *dynamic skyline* dan *reverse skyline*.

2.4 Dominansi Dinamis

Berdasarkan definisi "Skyline" yang telah dijelaskan pada sub-bagian sebelumnya, jika diberikan *dataset* yang sama, maka hasil *skyline* dari *dataset* tersebut pasti akan selalu sama. Oleh karena itu, para ahli juga menyebut *original skyline* sebagai *static skyline* [7].

Ada suatu kasus ketika perhitungan *skyline* didasarkan pada titik kueri. Jika diberikan *dataset* yang sama, namun titik kuerinya berbeda, maka hasil *skyline*-nya pun berbeda tergantung pada titik kueri. *Skyline* ini disebut dengan *dynamic skyline* karena memiliki sifat dominansi dinamis.

Diberikan *dataset* produk P dan *dataset* pelanggan (preferensi pelanggan) C yang setiap datanya direpresentasikan sebagai objek data d -dimensi dan hanya dapat menyimpan nilai numerik pada setiap dimensinya. Data produk dan pelanggan pada dimensi ke- i dinotasikan sebagai p^i dan c^i , $i \leq d$. Untuk menggambarkan objek data secara umum digunakan notasi ob .

Suatu objek data ob_1 dikatakan mendominasi objek data ob_2 secara dinamis berdasarkan objek data ob_3 , dinotasikan dengan

$ob_1 \prec_{ob_3} ob_2$, jika nilai ob_1 dekat dengan ob_3 pada semua dimensi dan ada nilai ob_1 yang lebih dekat dengan ob_3 dibandingkan nilai ob_2 dengan ob_3 minimal pada satu dimensi. Secara matematika, relasi $ob_1 \prec_{ob_3} ob_2$ terbentuk jika dan hanya jika:

- $$(a) \quad |ob_3^i - ob_1^i| \leq |ob_3^i - ob_2^i|, \forall i \in [1, \dots, d]$$
- $$(b) \quad |ob_3^i - ob_1^i| < |ob_3^i - ob_2^i|, \exists i \in [1, \dots, d] \quad (2.1)$$

Pada Tabel 2.5, diberikan contoh *dataset* produk dan preferensi pelanggan. Berdasarkan preferensi pelanggan c_1 , produk p_1 dikatakan mendominasi produk p_2 , dinotasikan dengan $p_1 \prec_{c_1} p_2$, karena memenuhi kedua syarat dominansi dinamis yakni (a) $|c_1^1 - p_1^1| = |5 - 5| = 0 \leq |c_1^1 - p_2^1| = |5 - 15| = 10$ dan (b) $|c_1^2 - p_1^2| = |2 - 9| = 7 < |c_1^2 - p_2^2| = |2 - 14| = 12$.

Tabel 2.5 Contoh *Dataset*
(a) Produk P dan (b) Preferensi Pelanggan C

(a)			(b)		
id	dim1	dim2	id	dim1	dim2
p_1	5	9	c_1	5	2
p_2	15	14	c_2	8	10
p_3	8	10	c_3	15	10
p_4	5	14	c_4	9	7
p_5	12	6	c_5	10	12
p_6	15	11	c_6	12	14
p_7	12	10	c_7	7	13
p_8	9	11	c_8	15	8
p_9	6	4	c_9	5	5
p_{10}	8	6	c_{10}	10	5

Sebaliknya, jika berdasarkan preferensi pelanggan c_6 , maka

produk p_2 -lah yang mendominasi p_1 , dinotasikan dengan $p_2 \prec_{c_6} p_1$, karena (a) $|c_6^1 - p_2^1| = |12 - 15| = 3 \leq |c_6^1 - p_1^1| = |12 - 5| = 7$ dan (b) $|c_6^2 - p_2^2| = |14 - 14| = 0 < |c_6^2 - p_1^2| = |14 - 9| = 5$. Dalam hal ini, preferensi pelanggan disebut dengan titik kueri karena dapat mempengaruhi sifat dominansi antar produk.

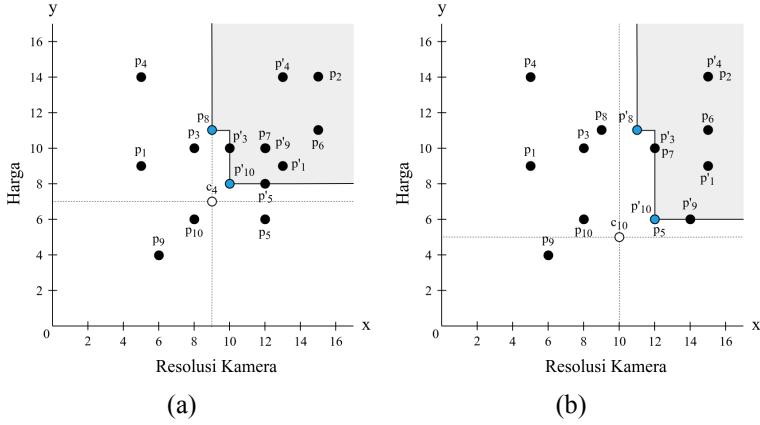
Mengambil contoh lain, produk p_1 tidak mendominasi p_2 berdasarkan pelanggan c_3 karena ada salah satu syarat dominansi dinamis yang tidak terpenuhi, (a) $|c_3^1 - p_1^1| = |15 - 5| = 10 \not\leq |c_3^1 - p_2^1| = |15 - 15| = 0$ dan (b) $|c_3^2 - p_1^2| = |10 - 9| = 1 < |c_3^2 - p_2^2| = |10 - 14| = 4$. Produk p_1 dan p_2 dikatakan saling mendominasi berdasarkan pelanggan c_2 .

2.5 Dynamic Skyline

Kueri *dynamic skyline* dalam komputasi k -MPP digunakan untuk mencari produk terbaik dari sudut pandang pelanggan [1], sehingga yang menjadi titik kueri adalah pelanggan. *Dynamic skyline* [2] dari seorang pelanggan $c_1 \in C$, dinotasikan dengan $DSL(c_1)$, berisi semua produk $p_1 \in P$ yang tidak didominasi oleh produk lain $p_2 \in P$ berdasarkan preferensi pelanggan c_1 , $p_2 \not\prec_{c_1} p_1$.

Dynamic skyline dapat dihitung menggunakan algoritme komputasi *skyline* tradisional [6], yaitu mentransformasikan semua titik $p \in P$ ke ruang data baru dengan menganggap titik kueri c sebagai titik asal dan jarak absolut titik p ke c digunakan sebagai fungsi pemetaan seperti yang ditunjukkan pada Gambar 2.2. Fungsi pemetaan f^i didefinisikan sebagai $f^i(p^i) = |c^i - p^i|$.

Menggunakan *dataset* pada Tabel 2.5, *dynamic skyline* dari pelanggan c_4 adalah $DSL(c_4) = \{p_8, p_{10}\}$, karena produk tersebut tidak didominasi oleh produk lain berdasarkan preferensi pelanggan c_4 . Berbeda halnya dengan c_{10} yang memiliki hasil *dynamic skyline* $DSL(c_{10}) = \{p_5, p_8, p_{10}\}$.



Gambar 2.2 (a) Komputasi *Dynamic Skyline* dari Pelanggan c_4 dan (b) *Dynamic Skyline* dari Pelanggan c_{10}

2.6 Reverse Skyline

Dalam komputasi k -MPP, kueri *reverse skyline* digunakan untuk mencari pelanggan potensial dari sudut pandang produsen [1], sehingga yang menjadi titik kueri adalah produk. *Reverse skyline* [3] dari sebuah produk $p_1 \in P$, dinotasikan dengan $RSL(p_1)$, berisi semua pelanggan $c \in C$ yang memiliki p_1 pada hasil *dynamic skyline*-nya.

Ada beberapa tahapan yang harus dilakukan dalam komputasi *reverse skyline* [1]. Pertama, menentukan *orthant* dari produk, dinotasikan dengan O . Setiap produk p memiliki 2^d *orthant* pada data d -dimensi. Kedua, menghitung *midpoint* atau titik tengah antara produk kueri dan produk lainnya, misalnya p_1 (sebagai titik kueri) dan p_2 , dihitung menggunakan rumus berikut:

$$m_2^i = \frac{(p_1^i + p_2^i)}{2} \quad (2.2)$$

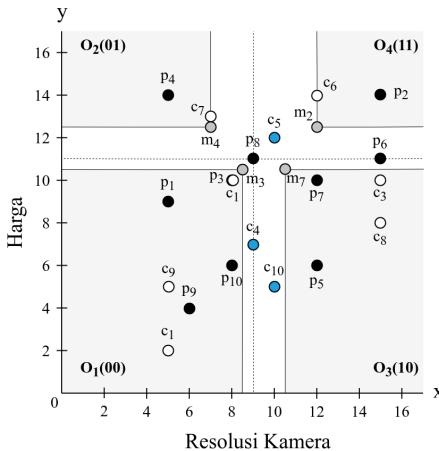
Kemudian menentukan *midpoint skyline* (juga dikenal sebagai

mid-skyline [8]) pada setiap *orthant*.

Langkah ketiga, mengecek apakah pelanggan $c \in C$ didominasi oleh *midpoint skyline* m berdasarkan produk p_1 atau tidak. Pelanggan c dikatakan didominasi oleh *midpoint skyline* m jika dan hanya jika:

- $|p_1^i - m^i| \leq |p_1^i - c^i|, \forall i \in [1, \dots, d]$
 - $|p_1^i - m^i| < |p_1^i - c^i|, \exists i \in [1, \dots, d]$
- (2.3)

Apabila c tidak didominasi oleh *midpoint skyline* m berdasarkan produk p_1 , maka c menjadi hasil dari *reverse skyline* p_1 , dinotasikan dengan $RSL(p_1)$. Untuk lebih jelasnya, komputasi *reverse skyline* ditunjukkan pada Gambar 2.3.



Gambar 2.3 Komputasi *Reverse Skyline* dari Produk p_8

Sebagai contoh, berdasarkan *dataset* yang diberikan pada Tabel 2.5, *reverse skyline* dari produk p_8 adalah pelanggan c_4 , c_5 , dan c_{10} , dinotasikan dengan $RSL(p_8) = \{c_4, c_5, c_{10}\}$ karena masing-masing pelanggan tersebut memiliki p_8 pada hasil *dynamic*

skyline-nya.

2.7 Kueri *k*-Most Promising Products (*k*-MPP)

Kueri *k*-Most Promising Products (*k*-MPP) adalah sebuah strategi pemilihan produk yang dikenalkan oleh Islam dan Liu dalam penelitiannya [1].

2.7.1 Uniform Product Adoption (UPA)

Uniform Product Adoption (UPA) mengasumsikan bahwa semua produk $p \in P$ yang muncul pada hasil *dynamic skyline* pelanggan $c \in C$ akan saling berkompetisi satu sama lain untuk menarik pelanggan c , sehingga produk-produk tersebut memiliki probabilitas yang sama untuk dibeli oleh pelanggan c .

Probabilitas produk p dibeli oleh pelanggan c , dinotasikan dengan $Pr(c, p|P)$ dapat dijelaskan oleh persamaan berikut:

$$Pr(c, p|P) = \begin{cases} \frac{1}{|DSL(c)|} & \text{if } p \in DSL(c) \\ 0 & \text{otherwise} \end{cases} \quad (2.4)$$

Berdasarkan Persamaan 2.4, dapat dipastikan bahwa setiap produk yang muncul dalam $DSL(c)$ memiliki kesempatan yang sama untuk dipilih oleh pelanggan c . Sebaliknya, produk yang tidak muncul dalam $DSL(c)$ tidak memiliki kesempatan sama sekali untuk dipilih oleh c .

Sebagai contoh menggunakan *dataset* pada Tabel 2.5, probabilitas produk p_8 dibeli oleh pelanggan c_4 adalah $Pr(c_4, p_8|P) = \frac{1}{|DSL(c_4)|} = \frac{1}{2}$, sedangkan probabilitas produk p_8 dibeli oleh pelanggan c_{10} adalah $\frac{1}{|DSL(c_{10})|} = \frac{1}{3}$.

2.7.1.1 Market Contribution

Market contribution atau kontribusi pasar sebuah produk $p \in P$ diukur dari total jumlah pelanggan yang mungkin lebih memilih membeli produk p dibandingkan produk lain p' .

Asumsinya jika seorang pelanggan memiliki dua produk atau lebih dalam hasil *dynamic skyline*-nya, maka ia akan memberikan bobot yang sama pada produk-produk tersebut sebagaimana yang sudah dijelaskan pada Persamaan 2.4. Sehingga, kontribusi pasar sebuah produk dihitung dari hasil akumulasi bobot yang didapatkan dari semua pelanggan $c \in C$.

Kontribusi pasar produk p , dinotasikan dengan $E(C, p|P)$, diperoleh dengan mengakumulasikan probabilitas produk dari setiap pelanggan $c \in C$, sebagai berikut:

$$E(C, p|P) = \sum_{\forall c \in C} Pr(c, p|P) \quad (2.5)$$

Karena probabilitas produk p dipilih oleh pelanggan yang tidak memiliki p pada hasil *dynamic skyline*-nya adalah nol (pada Persamaan 2.4), maka kita hanya perlu mengakumulasikan probabilitas produk dari setiap pelanggan c pada hasil $RSL(p)$. Sehingga, Persamaan 2.5 dapat disederhanakan menjadi:

$$E(C, p|P) = \sum_{\forall c \in RSL(p)} Pr(c, p|P) \quad (2.6)$$

Sebagai contoh menggunakan *dataset* pada Tabel 2.5, kontribusi pasar dari produk p_8 adalah $E(C, p_8|P) = Pr(c_4, p_8|P) + Pr(c_5, p_8|P) + Pr(c_{10}, p_8|P) = \frac{1}{2} + 1 + \frac{1}{3} = \frac{11}{6}$ atau 1.833.

Perhitungan kontribusi pasar juga dapat dilakukan pada sekumpulan produk atau *subset* produk P' , dinotasikan dengan

$E(C, P'|P)$, yang dijelaskan pada Persamaan 2.7.

$$E(C, P'|P) = \sum_{\forall p \in P'} E(C, p|P) \quad (2.7)$$

2.7.2 Strategi Pemilihan Produk

Diberikan *dataset* produk P , *dataset* preferensi pelanggan C , dan bilangan bulat positif k yang lebih kecil dari $|P|$. Kueri *k-Most Promising Products* (k -MPP), dinotasikan oleh Persamaan 2.8, akan memilih *subset* k produk P' dari P yang memiliki kontribusi pasar lebih besar dibandingkan dengan *subset* k produk P'' dari P yang lain [1].

$$k - MPP(P, C, k) \quad (2.8)$$

Jika merangkum semua penjelasan di atas, langkah-langkah yang harus dilakukan untuk memproses kueri k -MPP adalah: (1) menghitung *reverse skyline* dari setiap produk $p \in P$, (2) menghitung *dynamic Skyline* dari setiap pelanggan $c \in RSL(p)$, dan (3) memilih k produk dari P yang memiliki kontribusi pasar terbesar.

2.8 Python

Python adalah bahasa pemrograman tingkat tinggi, *interpreted*, dan berorientasi objek yang didukung oleh struktur data *built-in* tingkat tinggi dan semantik yang dinamis [10]. Python dikembangkan oleh Guido van Rossum pada akhir 1980-an dan dikelola oleh *Python Software Foundation*. Saat ini, Python sudah tersedia dalam dua versi, yakni 2.x dan 3.x.

Kelebihan bahasa pemrograman Python adalah pada keterbacaannya karena memiliki sintaksis yang sederhana, sehingga dapat mengurangi biaya pemeliharaan (*maintenance*). Python mendukung banyak modul dan *package*, serta memiliki

banyak *standard library* yang didistribusikan secara gratis. Selain itu, karena Python adalah bahasa *interpreted*, Python tidak memakan biaya untuk kompilasi sehingga proses pengubahan, pengujian, dan debug menjadi lebih cepat.

Melakukan debug pada program Python sangatlah mudah karena tidak akan mengakibatkan *segmentation fault*. Sebagai gantinya, ia akan menimbulkan *Exception* apabila menemukan suatu *error* atau kesalahan. Ketika program tidak menangkap *Exception*, maka Python akan menampilkan *stack trace* yang dapat digunakan untuk menganalisis dan memperbaiki kesalahan yang terjadi [10].

Pada Tugas Akhir ini, bahasa pemrograman Python digunakan untuk mengimplementasikan struktur data dan algoritme pada sistem perangkat lunak yang akan dibangun.

2.9 Flask

Flask adalah kerangka kerja web berbahasa Python yang sederhana, ringan, dan mudah dikembangkan, sehingga Flask kerap disebut dengan *microframework*. Flask dibangun dari dua pustaka utama, yaitu Jinja *template engine* dan Werkzeug WSGI *toolkit*, serta memiliki lisensi BSD. Saat ini, Flask dikembangkan dan dikelola oleh *Pallets team* dan kontributor komunitas.

Kerangka kerja Flask digunakan untuk mengimplementasikan aplikasi web dan layanan *web server* yang digunakan pada Tugas Akhir ini karena ringan dan lebih mudah digunakan dibandingkan dengan *framework* Python Django. Selain itu, Flask juga memiliki banyak dokumentasi dan tutorial yang dapat diikuti.

Halaman ini sengaja dikosongkan

BAB III

ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini akan dijelaskan mengenai analisis dan perancangan sistem perangkat lunak yang akan dibangun, meliputi struktur data, algoritme, dan arsitektur aplikasi.

3.1 Daftar Notasi

Tabel 3.1 menunjukkan daftar notasi yang digunakan dalam bab ini beserta deskripsinya.

Tabel 3.1 Daftar Notasi (2)

Notasi	Deskripsi
P	<i>Dataset</i> produk
C	<i>Dataset</i> pelanggan (preferensi pelanggan)
D	$P \cup C$
E	Himpunan <i>event</i>
p	Sebuah produk dalam P , $p \in P$
c	Seorang pelanggan dalam C , $c \in C$
e	Sebuah <i>event</i> dalam E , $e \in E$
p_{in}	Produk masuk
p_{out}	Produk keluar
c_{in}	Pelanggan masuk
c_{out}	Pelanggan keluar

Notasi	Deskripsi
P_{active}	Himpunan produk yang sedang aktif (di dalam lini masa)
C_{active}	Himpunan pelanggan yang sedang aktif (di dalam lini masa)
D_{active}	$P_{active} \cup C_{active}$
d	Jumlah dimensi pada D
i	Dimensi ke-1, ..., d
j	Timestamp ke-1, 2, ..., dst
$diff$	Selisih nilai
p_s	Produk sebagai subjek yang membandingkan
p_o	Produk sebagai objek pembanding
O	<i>Orthant</i>
$pos_x(p)$	Posisi p pada sumbu x
max_x	Nilai maksimum pada sumbu x
m	<i>Midpoint</i> antar produk
$DSL(c)$	Hasil <i>dynamic skyline</i> dari pelanggan c
$RSL(p)$	Hasil <i>reverse skyline</i> dari produk p
$Pr(c, p P)$	Probabilitas produk p dibeli oleh pelanggan c
$E(C, p P)$	Kontribusi pasar p
$E(C, P' P)$	Kontribusi pasar subset P' dari P
$k - MPPTI$	<i>k-Most Promising Products in Time Intervals</i>
k	Jumlah data
$[t_i : t_e]$	Interval waktu
$PBox$	<i>Pandora Box</i>

Notasi	Deskripsi
ts	<i>Timestamp</i>

3.2 Analisis Sistem

Analisis sistem dijelaskan dalam empat bagian, yakni analisis permasalahan, deskripsi umum sistem, fungsi sistem, dan analisis kebutuhan fungsional.

3.2.1 Analisis Permasalahan

Permasalahan yang ingin diselesaikan pada Tugas Akhir ini adalah bagaimana menjawab kueri *k-Most Promising Products* berbasis interval waktu (*k-MPPTI*). Interval waktu, dinotasikan dengan $[t_i : t_e](t_i \leq t_e)$, digunakan untuk menentukan rentang waktu pencarian.

Permasalahan ini tidak dapat langsung diselesaikan menggunakan metode dan algoritme yang sudah ada [1]. Sehingga, diperlukan pendekatan lain yang akan dijelaskan pada bagian perancangan sistem.

3.2.2 Deskripsi Umum Sistem

Secara umum, sistem yang akan dibangun adalah sebuah sistem berbasis web yang dapat membantu pengguna untuk memilih k -produk yang paling menjanjikan. Dikatakan "menjanjikan" jika produk tersebut memiliki kontribusi pasar yang besar.

Sistem ini memiliki dua proses utama, yaitu (1) *data precomputing* untuk menghitung kontribusi pasar masing-masing produk dan (2) proses utama (selanjutnya akan disebut dengan *query processing*) untuk memproses dan menampilkan hasil kueri pencarian yang dimasukkan oleh pengguna.

Sistem ini dibangun menggunakan arsitektur *client-server*. Aplikasi *client* didesain berbasis web dengan memanfaatkan Flask *microframework*, HTML, CSS, dan JavaScript. Selain itu, Flask juga digunakan sebagai *web server*.

3.2.3 Fungsi Sistem

Sistem yang akan dibangun memiliki beberapa fungsi utama sebagai berikut:

1. Dapat menerima masukan data berupa file dari pengguna
2. Dapat menampilkan informasi dan pratinjau data yang dimasukkan oleh pengguna
3. Dapat menampilkan visualisasi data
4. Dapat melakukan proses *data precomputing* menggunakan algoritme yang dipilih oleh pengguna
5. Dapat menerima masukan kueri pencarian
6. Dapat memproses kueri pencarian
7. Dapat menampilkan hasil kueri
8. Dapat menampilkan waktu eksekusi

3.2.4 Analisis Kebutuhan Fungsional

Sistem yang dibuat harus mampu memenuhi beberapa fungsi utama yang telah dijelaskan pada sub-bagian sebelumnya. Fungsi-fungsi ini merupakan hasil dari analisis kebutuhan fungsional dari pengguna yang dijelaskan pada Tabel 3.2.

Tabel 3.2 Kebutuhan Fungsional

Kode	Deskripsi Kebutuhan
F-001	Mengunggah data
F-002	Melihat informasi dan pratinjau data
F-003	Melihat visualisasi data
F-004	Memilih algoritme yang digunakan untuk <i>data precomputing</i>
F-005	Memasukkan kueri pencarian
F-006	Melihat hasil kueri
F-007	Melihat waktu eksekusi

Penjelasan rinci dari masing-masing kebutuhan fungsional pada tabel 3.2 dijelaskan sebagai berikut:

1. Mengunggah data

Pengguna dapat mengunggah data produk dan preferensi pelanggan dalam bentuk file berekstensi csv.

2. Melihat informasi dan pratinjau data

Pengguna dapat melihat informasi dan pratinjau dari data yang dimasukkan berupa tabel sebanyak dua puluh baris. Informasi yang ditampilkan antara lain jumlah baris, jumlah kolom, dan nama kolom.

3. Melihat visualisasi data

Pengguna juga dapat melihat visualisasi dari data yang dimasukkan berupa lini masa sederhana.

4. Memilih algoritme yang digunakan untuk *data precomputing*

Pengguna dapat memilih algoritme yang akan digunakan untuk *data precomputing*, yaitu algoritme *k-MPPTI* dan *Brute Force*.

5. Memasukkan kueri pencarian

Pengguna dapat memasukkan kueri pencarian berupa jumlah

produk (k) dan interval waktu.

6. Melihat hasil kueri

Pengguna dapat melihat hasil kueri pencarian berupa k -produk dengan jumlah kontribusi pasar terbesar beserta skor kontribusi pasar-nya.

7. Melihat waktu eksekusi

Pengguna dapat melihat informasi terkait waktu eksekusi.

3.3 Perancangan Sistem

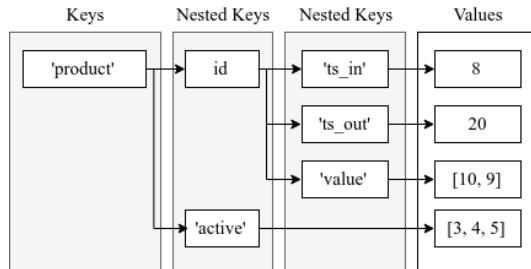
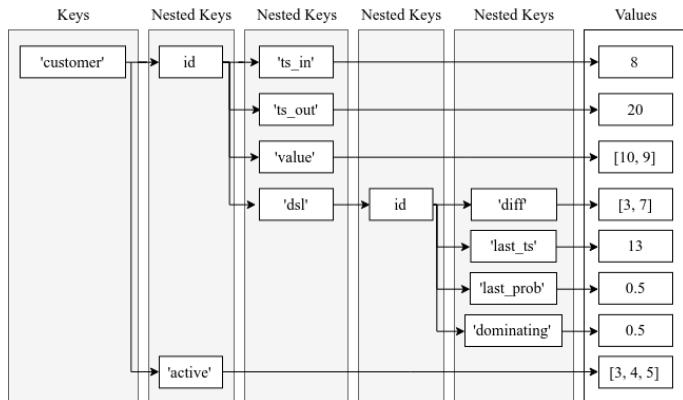
Perancangan sistem akan dibagi menjadi empat bagian, yakni struktur data, algoritme utama, algoritme pembanding menggunakan metode *brute force*, dan arsitektur aplikasi.

3.3.1 Struktur Data

Struktur data adalah suatu cara untuk menyimpan, menyusun, mengelompokkan, dan merepresentasikan suatu data. Ada tiga struktur data utama yang digunakan dalam komputasi k -MPPTI, yaitu *Data Storage*, *Event Queue*, dan *Pandora Box*.

3.3.1.1 *Data Storage*

Data Storage adalah sebuah struktur data *dictionary* yang digunakan untuk menyimpan data produk dan pelanggan. Struktur data *dictionary* lebih efisien untuk pencarian data karena menggunakan konsep *key-value pairs*, berbeda dengan struktur data *list* atau *array* yang menggunakan indeks untuk mengakses nilai suatu data.

Gambar 3.1 Struktur Data *Dictionary* ProdukGambar 3.2 Struktur Data *Dictionary* Pelanggan

Struktur data *dictionary* yang digunakan berbentuk *nested dictionary* yang terdiri dari dua *key* utama, yaitu '*product*' yang menyimpan data produk dan '*customer*' yang menyimpan data pelanggan. Struktur *nested key* masing-masing data dijelaskan pada Gambar 3.1 dan 3.2 dan deskripsinya dijelaskan pada Tabel 3.3.

Tabel 3.3 *Key* dari *Data Storage*

Key	Deskripsi
'product'	Menyimpan data produk
'customer'	Menyimpan data pelanggan
<i>id</i>	ID data produk atau pelanggan dijadikan sebagai <i>key</i>
'active'	Menyimpan ID data produk atau pelanggan yang sedang aktif dalam bentuk <i>array</i>
'ts_in'	Menyimpan <i>timestamp</i> atau waktu masuk
'ts_out'	Menyimpan <i>timestamp</i> atau waktu keluar
'value'	Menyimpan nilai data produk atau pelanggan pada semua dimensi dalam bentuk <i>array</i>
'dsl'	Menyimpan hasil <i>dynamic skyline</i> dalam bentuk <i>dictionary</i> dengan <i>id</i> produk sebagai <i>key</i>
'diff'	Menyimpan selisih antara nilai data produk dan pelanggan pada masing-masing dimensi
'last_ts'	Menyimpan <i>timestamp</i> terakhir saat diperbarui ke <i>Pandora Box</i>
'last_prob'	Menyimpan probabilitas terakhir saat diperbarui ke <i>Pandora Box</i>
'dominating'	Menyimpan ID produk lain yang pernah didominasi

3.3.1.2 *Event Queue*

Event adalah titik-titik tempat terjadinya perubahan di dalam himpunan data, yaitu jika ada data yang masuk atau keluar. Ada empat jenis *event* yang terjadi dalam penelitian ini, yaitu:

1. Data Produk masuk
2. Data Produk keluar
3. Data Pelanggan masuk
4. Data Pelanggan keluar

Produk dan pelanggan disebut dengan pemilik *event*, sedangkan masuk dan keluar disebut dengan aksi *event*.

Event Queue adalah sebuah struktur data *queue* yang berfungsi untuk menyimpan *event-event* yang terjadi di dalam himpunan data. *Queue* memiliki prinsip FIFO (*First In First Out*), sehingga *event* akan diproses secara berurutan menurut antrian waktu. *Event Queue* menyimpan empat informasi, yaitu *timestamp*, pemilik *event*, ID pemilik *event*, dan aksi *event*. Untuk lebih jelasnya, atribut *Event Queue* dijelaskan pada Tabel 3.4.

Tabel 3.4 Atribut dari *Event Queue*

Atribut	Deskripsi
<i>timestamp</i>	Waktu terjadinya <i>event</i>
<i>owner</i>	Pemilik <i>event</i> (produk = 0, pelanggan = 1)
<i>ownerId</i>	ID pemilik <i>event</i>
<i>action</i>	Jenis aksi yang dilakukan (masuk = 0, keluar = 1)

3.3.1.3 *Pandora Box*

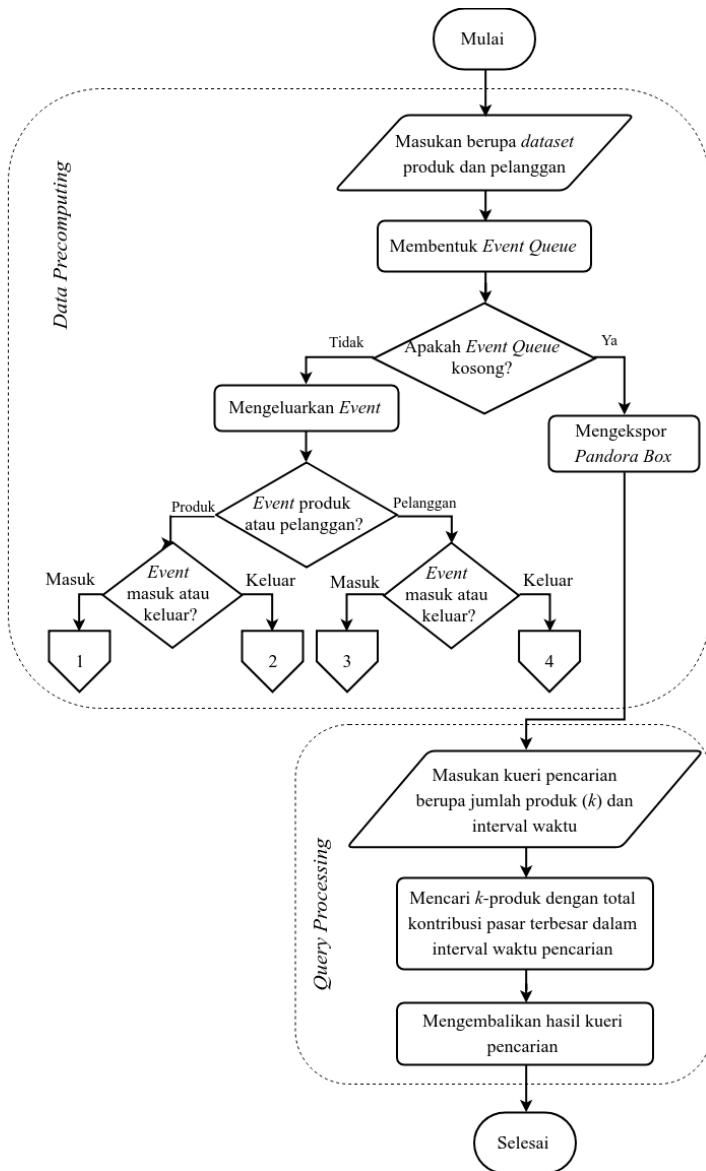
Pandora Box adalah sebuah struktur data *array* dua dimensi yang terdiri dari sumbu *x* (*time series*) dan sumbu *y* (produk). Struktur data ini digunakan untuk menyimpan skor kontribusi pasar produk setiap waktu. Menggunakan contoh *dataset* pada Tabel 3.5, maka model *Pandora Box* yang terbentuk adalah seperti pada Gambar 3.3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p ₁	0	1	2	1	0.67	0	0	0	0	0	0	0	0	0	0
p ₂	0	0	0	0	0	1.33	1.33	1.33	0.5	1	1	1	1	0	0
p ₃	0	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	0.5	1	0.5
p ₄	0	0	0	2	1.67	1.33	1.33	1.33	0.5	0	0	0	0	0	0
p ₅	0	0	0	0	1.67	1.33	1.33	1.33	1	1.5	1.5	1.5	1.5	2	1.5

Gambar 3.3 Contoh *Pandora Box* dari *Dataset* 3.5

3.3.2 Algoritme Utama

Sebagaimana yang telah dijelaskan sebelumnya bahwa algoritme *k-MPPTI* terdiri dari dua tahap pemrosesan, yaitu *data precomputing* dan *query processing*. Secara garis besar, alur kerja sistem secara umum disajikan dalam bentuk diagram alur yang dapat dilihat pada Gambar 3.4.



Gambar 3.4 Diagram Alur Algoritme k-MPPTI

Tahap *data precomputing* bertujuan untuk menghitung skor kontribusi pasar masing-masing produk berdasarkan preferensi pelanggan. Diawali dengan pembentukan *Event Queue* untuk mencatat semua *event* yang terjadi selama pemrosesan data. Kemudian, memproses *event-event* tersebut menggunakan algoritme pemrosesan berdasarkan jenis *event*-nya. Terakhir adalah mengekspor *Pandora Box* untuk digunakan sebagai masukan pada tahap *query processing*.

Tahap kedua adalah *query processing* yang bertujuan untuk memproses kueri pencarian yang dimasukkan oleh pengguna berupa jumlah produk (k) dan interval waktu pencarian. Diawali dengan mencari produk sejumlah k yang memiliki total skor kontribusi pasar terbesar selama interval waktu pencarian, kemudian mengembalikan hasil kueri pencarian berupa k -produk yang paling menjanjikan kepada pengguna.

Untuk memudahkan interaksi antara pengguna dan sistem, dibuatlah aplikasi berbasis web yang memudahkan pengguna memasukkan data produk dan pelanggan, melihat pratinjau dan visualisasi data, memasukkan kueri pencarian, serta melihat hasil kueri pencarian.

3.3.2.1 Data Precomputing

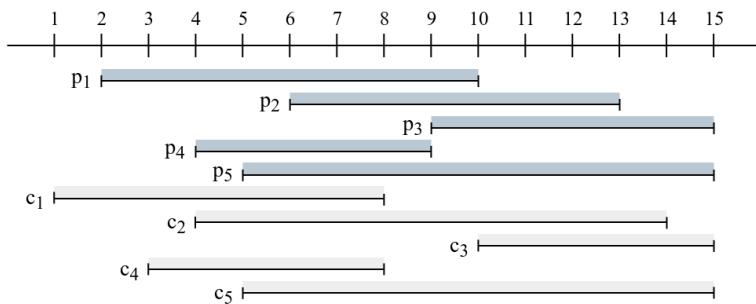
Data precomputing adalah sebuah proses yang dapat menunjang performa algoritme *query processing* supaya dapat bekerja lebih efektif dan efisien. Tidak adanya proses *data precomputing* menyebabkan pengulangan komputasi data setiap kali seseorang memasukkan kueri pencarian. Karena data yang digunakan adalah *historical data*, yaitu data yang dikumpulkan dari kejadian yang telah lalu, maka komputasi data cukup dilakukan satu kali saja di awal (*precomputing*). Berbeda halnya jika data yang digunakan adalah *streaming data* yang nilainya terus berubah dalam periode waktu tertentu.

Tabel 3.5 Contoh *Dataset*
(a) Produk P dan (b) Preferensi Pelanggan C

ID	Timestamp		Nilai	
	t_i	t_e	d_1	d_2
p_1	2	10	6	3
p_2	6	13	4	12
p_3	9	15	6	15
p_4	4	9	9	5
p_5	5	15	12	10

ID	Timestamp		Nilai	
	t_i	t_e	d_1	d_2
c_1	1	8	2	8
c_2	4	14	4	10
c_3	10	15	6	11
c_4	3	8	8	12
c_5	5	15	9	10

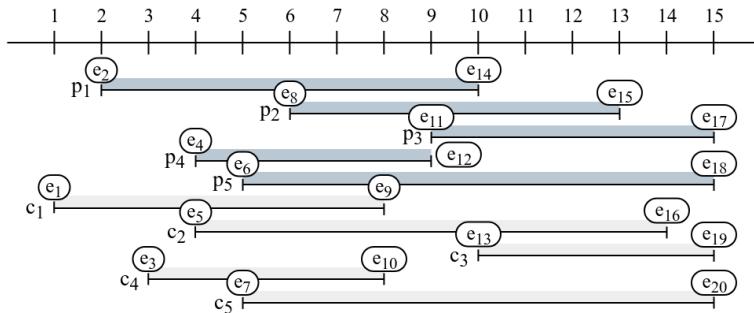
Pada Tabel 3.5, diberikan contoh *dataset* produk P dan preferensi pelanggan C yang setiap datanya direpresentasikan sebagai titik d -dimensi dengan serial waktu $[t_i : t_e]$. Pendekatan yang digunakan untuk memproses data multidimensi dengan serial waktu adalah menggunakan lini masa yang diilustrasikan pada Gambar 3.5.



Gambar 3.5 lini Masa Data Produk dan Pelanggan

Lini masa atau alur waktu adalah suatu representasi kronologis urutan peristiwa atau kejadian (*event*). Lini masa dapat

dibuat menurut era, abad, tahun, bulan, minggu, atau hari, namun untuk pemodelan ini, waktu direpresentasikan sebagai bilangan bulat positif. Di dalam lini masa, terdapat titik-titik yang mewakili kejadian penting (*event*) yang dinotasikan dengan $e \in E$.



Gambar 3.6 *Event* dalam Lini Masa Data Produk dan Pelanggan

Ada empat jenis *event* di dalam lini masa, yaitu data produk masuk, produk keluar, pelanggan masuk, dan pelanggan keluar. *Event-event* tersebut dicatat dan dimasukkan ke dalam *Event Queue*, kemudian diproses satu persatu secara berurutan. Contoh *Event Queue* yang terbentuk dari *dataset* pada Tabel 3.5 ditunjukkan pada Tabel 3.6.

Tabel 3.6 *Event Queue*

ID Event	Timestamp	ID Data	Aksi
e_1	1	c_1	Masuk
e_2	2	p_1	Masuk
e_3	3	c_4	Masuk
e_4	4	p_4	Masuk
e_5	4	c_2	Masuk
e_6	5	p_5	Masuk
e_7	5	c_5	Masuk

ID Event	Timestamp	ID Data	Aksi
e_8	6	p_2	Masuk
e_9	8	c_1	Keluar
e_{10}	8	c_4	Keluar
e_{11}	9	p_3	Masuk
e_{12}	9	p_4	Keluar
e_{13}	10	c_3	Masuk
e_{14}	10	p_1	Keluar
e_{15}	13	p_2	Keluar
e_{16}	14	c_2	Keluar
e_{17}	15	p_3	Keluar
e_{18}	15	p_5	Keluar
e_{19}	15	c_3	Keluar
e_{20}	15	c_5	Keluar

Ada empat jenis algoritme pemrosesan berdasarkan jenis *event*-nya, yaitu: (1) *Product Insertion*, (2) *Product Deletion*, (3) *Customer Insertion*, dan (4) *Customer Deletion*. Masing-masing proses itu membutuhkan dua jenis komputasi *skyline*, yaitu *dynamic skyline* dan *reverse skyline*, sebagai metode perhitungan probabilitas dan kontribusi pasar [1].

3.3.2.1.1 Proses *Product Insertion*

Proses *Product Insertion* adalah proses yang dijalankan ketika ada data produk yang masuk, dinotasikan dengan p_{in} . Proses ini sangat penting dilakukan karena ada kemungkinan jika produk baru dapat mendominasi produk lama, sehingga hasil *dynamic skyline* seorang pelanggan $c \in C$ dan perhitungan probabilitasnya ikut berubah.

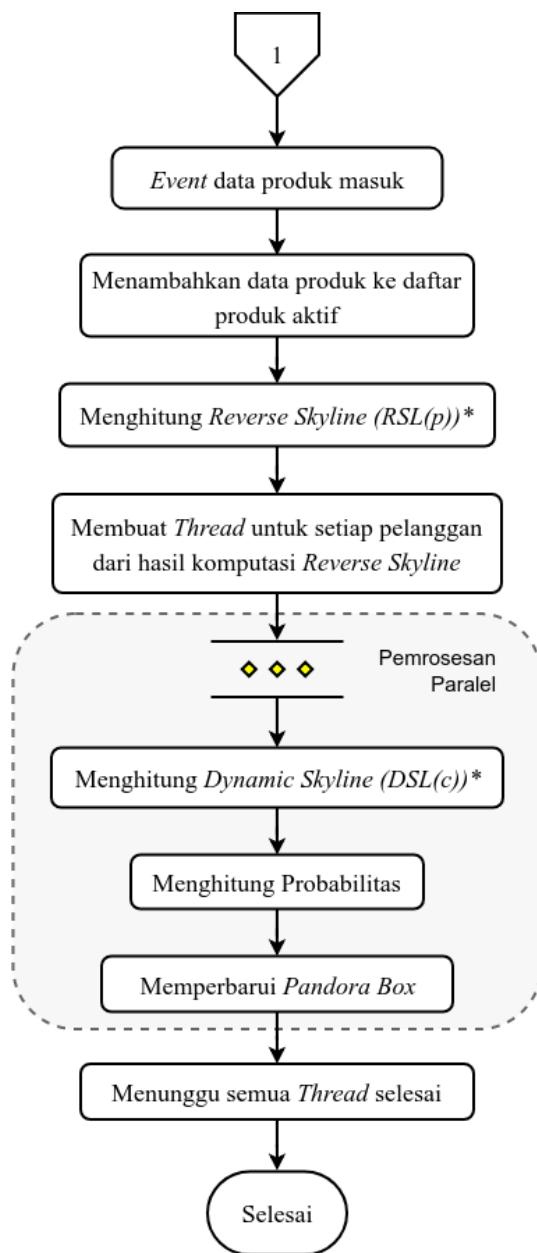
Secara garis besar, algoritme pemrosesan disajikan dalam

bentuk diagram alur pada Gambar 3.7. Pemrosesan diawali dengan (1) menambahkan produk p_{in} ke dalam daftar produk aktif P_{active} . Kemudian, (2) menghitung $RSL(p_{in})$. Dari hasil $RSL(p_{in})$ akan didapatkan hasil berupa sejumlah pelanggan yang menganggap p_{in} sebagai hasil DSL -nya. Dilanjutkan dengan (3) menghitung $DSL(c)$ untuk masing-masing $c \in RSL(p_{in})$ dan (4) menghitung probabilitas masing-masing produk. Diakhiri dengan (5) memperbarui *Pandora Box*.

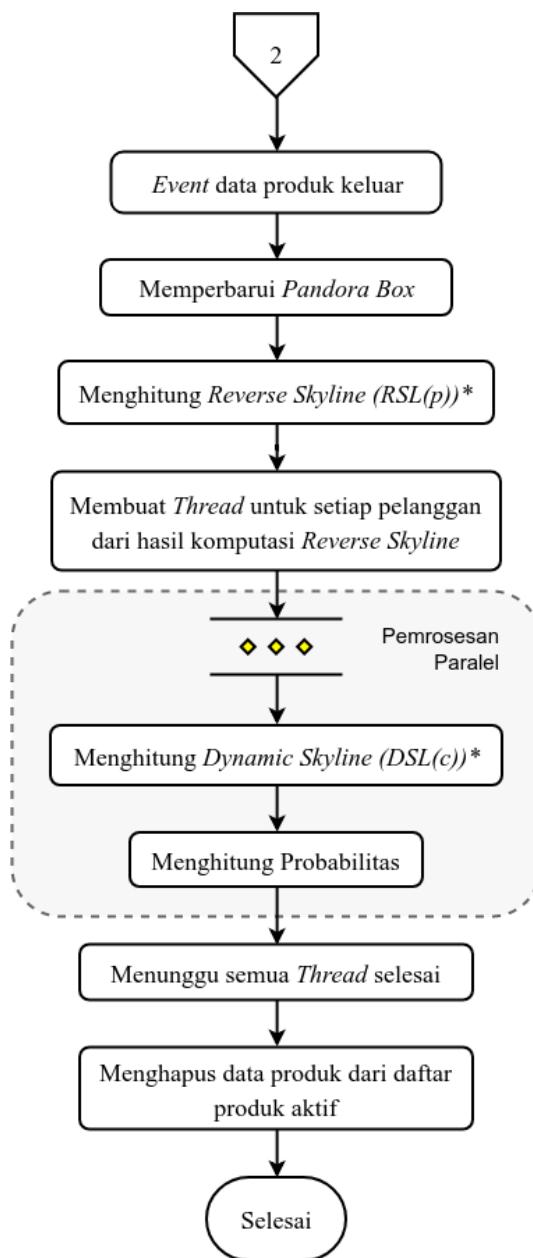
3.3.2.1.2 Proses *Product Deletion*

Proses *Product Deletion* adalah proses yang dijalankan ketika ada data produk yang keluar, dinotasikan dengan p_{out} . Proses ini sangat penting dilakukan karena ada kemungkinan jika sebuah produk yang pernah menjadi hasil *dynamic skyline* seorang pelanggan $c \in C$ keluar, maka produk lain yang pernah didominasi akan menjadi hasil $DSL(c)$ yang baru.

Secara garis besar, algoritme pemrosesan disajikan dalam bentuk diagram alur pada Gambar 3.8. Pemrosesan diawali dengan (1) memperbarui *Pandora Box* untuk mengisi indeks $PBox$ sebelumnya yang kosong. Kemudian (2) menghitung $RSL(p_{out})$. Dari hasil $RSL(p_{out})$ akan didapatkan hasil berupa sejumlah pelanggan yang menganggap p_{out} sebagai hasil DSL -nya. Dilanjutkan dengan (3) menghitung $DSL(c)$ untuk masing-masing $c \in RSL(p_{out})$ yang dimaksudkan untuk mencari produk lain yang pernah didominasi. Diakhiri dengan (4) menghitung probabilitas masing-masing produk, serta (5) menghapus produk p_{out} dari daftar produk aktif P_{active} .



Gambar 3.7 Diagram Alur Proses *Product Insertion*

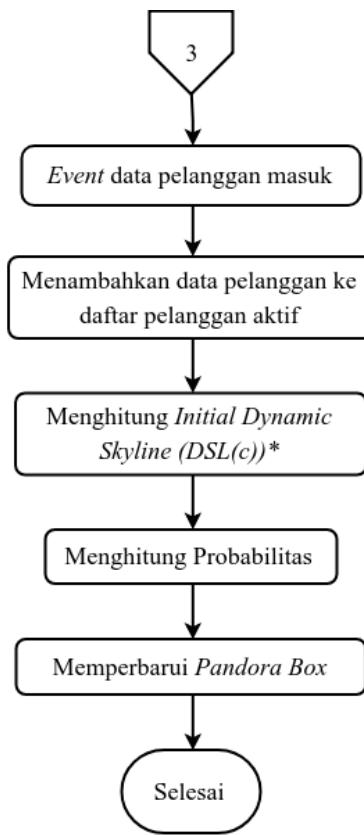
Gambar 3.8 Diagram Alur Proses *Product Deletion*

3.3.2.1.3 Proses *Customer Insertion*

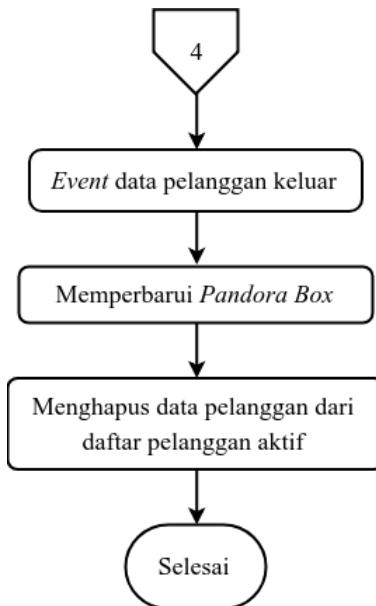
Proses *Customer Insertion* adalah proses yang dijalankan ketika ada data pelanggan yang masuk, dinotasikan dengan c_{in} . Secara garis besar, algoritme pemrosesan disajikan dalam bentuk diagram alur pada Gambar 3.9. Pemrosesan diawali dengan (1) menambahkan pelanggan c_{in} ke dalam daftar pelanggan aktif C_{active} . Kemudian (2) menghitung $Initial\ DSL(c_{in})$ untuk mendapatkan hasil *dynamic skyline* awal. Diakhiri dengan (3) menghitung probabilitas dan (4) memperbarui *Pandora Box*.

3.3.2.1.4 Proses *Customer Deletion*

Proses *Customer Deletion* adalah proses yang dijalankan ketika ada data pelanggan yang keluar, dinotasikan dengan c_{out} . Secara garis besar, algoritme pemrosesan disajikan dalam bentuk diagram alur pada Gambar 3.10. Pemrosesan diawali dengan (1) memperbarui *Pandora Box* dengan cara untuk mengisi indeks $PBox$ sebelumnya yang kosong, kemudian diakhiri dengan (2) menghapus pelanggan c dari daftar pelanggan aktif C_{active} .



Gambar 3.9 Diagram Alur Proses *Customer Insertion*



Gambar 3.10 Diagram Alur Proses *Customer Deletion*

3.3.2.1.5 Komputasi *Reverse Skyline*

Komputasi *reverse skyline* digunakan untuk mencari pelanggan potensial dari sudut pandang produsen [1]. *Reverse skyline* [3] dari sebuah produk $p_1 \in P$, dinotasikan dengan $RSL(p_1)$, berisi semua pelanggan $c \in C$ yang memiliki p_1 pada hasil *dynamic skyline*-nya.

Komputasi *reverse skyline* diawali dengan menentukan *orthant* dari produk, dinotasikan dengan O . Dalam geometri, *orthant* adalah analog dalam ruang data d -dimensi atau biasa dikenal sebagai kuadran dalam bidang dua dimensi. Setiap produk p memiliki 2^d *orthant* pada data d -dimensi.

Orthant ditandai menggunakan bilangan biner. Sebagai

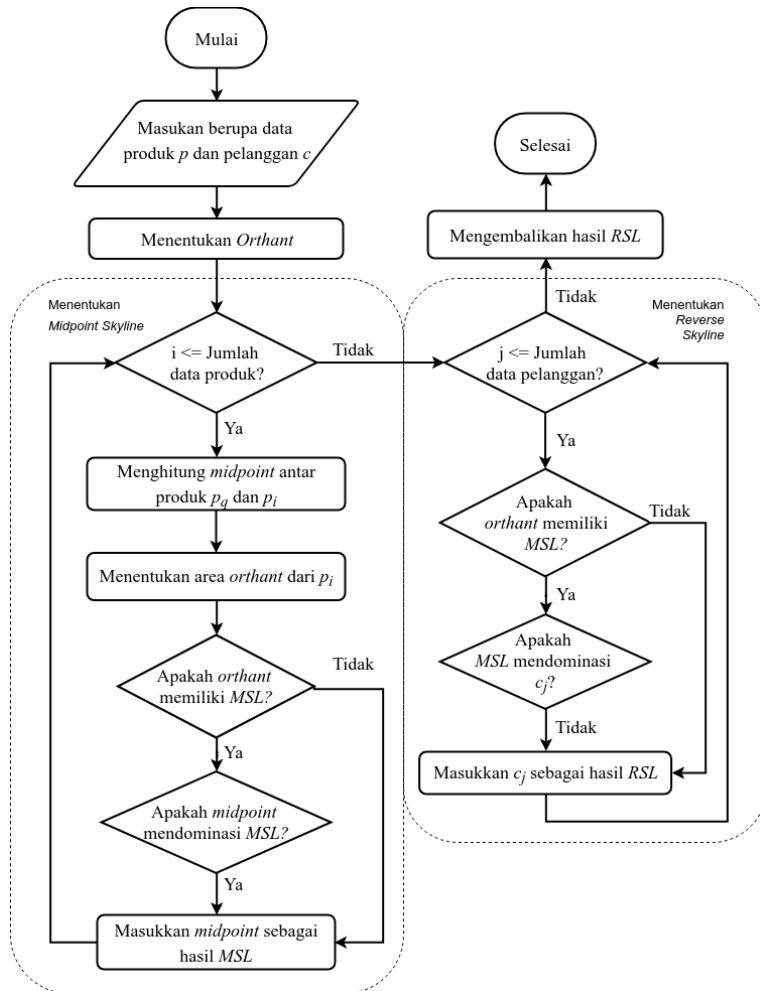
contoh, terdapat empat *orthant* pada bidang dua dimensi, yaitu O_{00} , O_{01} , O_{10} , dan O_{11} , dan delapan *orthant* pada bidang tiga dimensi, yaitu O_{000} , O_{001} , O_{010} , O_{011} , O_{100} , O_{101} , O_{110} dan O_{111} . Penggunaan bilangan biner bertujuan untuk menandai batas wilayah sebuah *orthant*. Misalnya, *orthant* O_{010} dari produk p_1 memiliki wilayah dengan batas-batas sebagai berikut: sumbu $x[0 : pos_x(p_1)]$, sumbu $y[pos_y(p_1) : max_y]$, dan sumbu $z[0 : pos_z(p_1)]$.

Langkah selanjutnya adalah menghitung *midpoint* atau titik tengah antara produk kueri dan produk lainnya, misalnya p_1 (sebagai titik kueri) dan $p_2 \in P$, menggunakan rumus berikut:

$$m_2^i = \frac{(p_1^i + p_2^i)}{2} \quad (3.1)$$

Kemudian, menentukan *midpoint skyline* atau *mid-skyline* [8] pada setiap *orthant*.

Langkah terakhir adalah mengecek setiap pelanggan $c \in C$ apakah didominasi oleh hasil *mid-skyline* pada masing-masing *orthant* atau tidak (Persamaan 2.3). Jika c didominasi, maka c tidak dapat menjadi hasil *reverse skyline*.



Gambar 3.11 Diagram Alur Komputasi *Reverse Skyline*

3.3.2.1.6 Komputasi *Dynamic Skyline*

Komputasi *dynamic skyline* digunakan untuk mencari produk

terbaik dari sudut pandang pelanggan [1]. *Dynamic skyline* [2] dari seorang pelanggan $c_1 \in C$, dinotasikan dengan $DSL(c_1)$, berisi semua produk $p_1 \in P$ yang tidak didominasi oleh produk lain $p_2 \in P$ berdasarkan preferensi pelanggan c_1 , $p_2 \not\prec_{c_1} p_1$.

Secara umum, proses komputasi *dynamic skyline* diawali dengan perhitungan selisih absolut dari nilai masing-masing dimensi antara pelanggan dan produk, dinotasikan dengan:

$$diff^i = |c_1^i - p^i| \quad (3.2)$$

Selanjutnya, mengecek dominansi dinamis antar produk dengan membandingkan selisih absolut-nya. Misalnya, ada dua produk yang akan dibandingkan, dinotasikan dengan p_s sebagai subjek yang dibandingkan dan p_o sebagai objek pembanding. Berdasarkan syarat dominansi dinamis (Persamaan 2.1), p_s dikatakan mendominasi p_o jika dan hanya jika:

$$\begin{aligned} (a) \quad & diff_s^i \leq diff_o^i, \forall i \in [1, \dots, d] \\ (b) \quad & diff_s^i < diff_o^i, \exists i \in [1, \dots, d] \end{aligned} \quad (3.3)$$

Pengecekan dominansi dinamis ini dilakukan secara iteratif sampai dipastikan suatu p_1 tidak didominasi oleh p_2 lain sama sekali. Jika p_1 pernah didominasi, maka p_1 tidak dapat menjadi hasil *dynamic skyline*.

Komputasi *DSL* dalam *k-MPPTI* dibagi menjadi 3 jenis, yaitu (1) *Initial DSL*, digunakan ketika ada data pelanggan yang masuk, (2) *DSL – PI*, digunakan ketika ada data produk yang masuk, dan (3) *DSL – PD*, digunakan ketika ada data produk yang keluar.

3.3.2.1.7 Perhitungan Probabilitas

Setelah mendapatkan hasil *dynamic skyline* dan *reverse skyline*, selanjutnya adalah menghitung probabilitas masing-masing produk $p \in P_{active}$ dipilih oleh pelanggan $c \in C_{active}$ yang dinotasikan oleh persamaan berikut:

$$Pr_t(c, p | P_{active}) = \begin{cases} \frac{1}{|DSL(c)|} & \text{if } p \in DSL(c) \\ 0 & \text{otherwise} \end{cases} \quad (3.4)$$

Karena probabilitas produk p dipilih oleh pelanggan yang tidak memiliki p pada hasil *dynamic skyline*-nya adalah nol, maka perhitungan probabilitas dapat disederhanakan menjadi:

$$Pr_t(c, p | P_{active}), \forall c \in RSL(p) \quad (3.5)$$

3.3.2.1.8 Perhitungan Kontribusi Pasar

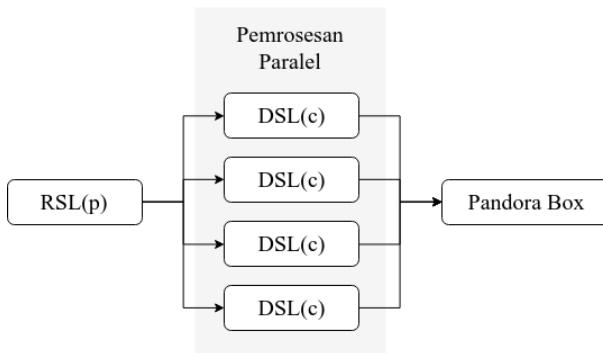
Setelah mendapatkan hasil perhitungan probabilitas, selanjutnya adalah menghitung kontribusi pasar dengan cara mengakumulasikan probabilitas produk dari setiap pelanggan $c \in C_{active}$ sebagaimana yang dijelaskan pada Persamaan 3.6.

$$E_t(C, p | P_{active}) = \sum_{\forall c \in RSL(p)} Pr_t(c, p | P_{active}) \quad (3.6)$$

3.3.2.1.9 Pemrosesan Paralel

Untuk meningkatkan efisiensi waktu komputasi, algoritme k -MPPTI mengimplementasikan konsep pemrosesan paralel, yaitu suatu bentuk komputasi dua atau lebih tugas yang dilakukan secara bersamaan dan beroperasi dengan prinsip bahwa masalah besar seringkali dapat dibagi dan dipecah menjadi masalah yang lebih kecil, kemudian dipecahkan secara bersamaan (paralel) [9].

Pemrosesan paralel dilakukan dengan cara menggunakan satu atau lebih CPU atau prosesor untuk menjalankan program atau multi-*thread*. Karena dilakukan secara bersamaan, maka pemrosesan ini hanya dapat dilakukan jika suatu tugas tidak membutuhkan masukan dari keluaran tugas sebelumnya, misalnya komputasi $DSL(c)$ untuk setiap $c \in RSL(p)$ yang diilustrasikan pada Gambar 3.12.



Gambar 3.12 Pemrosesan Paralel

3.3.2.2 *Query Processing*

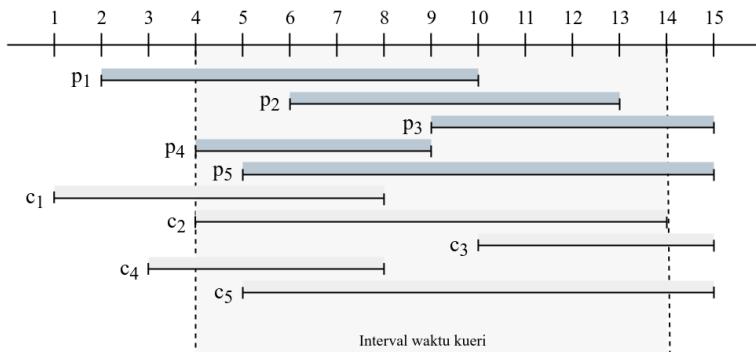
Query processing adalah algoritme pencarian produk sejumlah k yang paling menjanjikan dalam interval waktu tertentu. Selanjutnya kueri ini disebut dengan k -MPPTI (k -Most Promising Products in Time Interval) yang dinotasikan oleh Persamaan 3.7.

$$k - MPPTI(k, [t_i : t_e]) \quad (3.7)$$

Kueri k -MPPTI hanya membutuhkan dua masukan saja, yaitu bilangan bulat positif k yang lebih kecil dari $|P|$ sebagai jumlah produk yang dicari dan interval waktu pencarian yang terdiri dari waktu awal dan waktu akhir $[t_i : t_e]$. Berbeda dengan

kueri k -MPP (Persamaan 2.8), kueri k -MPPTI tidak membutuhkan masukan *dataset* produk P dan preferensi pelanggan C lagi karena sudah melalui tahap *data precomputing* yang menghasilkan *Pandora Box*.

Algoritme ini mengadopsi strategi pemilihan produk k -MPP, yaitu memilih *subset* k produk P' dari P yang memiliki kontribusi pasar lebih besar dibandingkan dengan *subset* k produk P'' dari P yang lain [1]. Bedanya, k -MPPTI menambahkan fitur pembatasan interval waktu pencarian.



Gambar 3.13 Ilustrasi Interval Waktu Pencarian

Ada dua langkah pemrosesan yang harus dilakukan, yaitu (1) mengakumulasi skor kontribusi pasar setiap produk $p \in P$ dalam interval waktu pencarian. Total kontribusi pasar berbasis interval waktu dinotasikan sebagai $MC_{[t_i:t_e]}(p), \forall p \in P$. Kemudian (2) mengurutkan total skor dari yang terbesar dan mengembalikan produk sejumlah k teratas sebagai hasil dari kueri pencarian.

Misalnya, seorang pengguna ingin mencari 3 produk yang paling menjanjikan dalam interval waktu 4 hingga 14, dinotasikan dengan $k - MPPTI(3, [4 : 14])$. Berdasarkan hasil perhitungan total kontribusi pasar berbasis interval waktu pada Tabel 3.7, produk

p_5 , p_2 dan p_4 adalah 3 produk paling menjanjikan dalam interval waktu 4 hingga 14.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p_1	0	1	2	1	0.67	0	0	0	0	0	0	0	0	0	0
p_2	0	0	0	0	0	1.33	1.33	1.33	0.5	1	1	1	1	0	0
p_3	0	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	1	0.5	
p_4	0	0	0	2	1.67	1.33	1.33	1.33	0.5	0	0	0	0	0	0
p_5	0	0	0	0	1.67	1.33	1.33	1.33	1	1.5	1.5	1.5	1.5	2	1.5

Interval waktu kueri

Gambar 3.14 Perhitungan Kontribusi Pasar pada *Pandora Box*
Berdasarkan Interval Waktu Pencarian

Tabel 3.7 Hasil Perhitungan Kontribusi Pasar Berbasis Interval Waktu

$MC_{[4:14]}(p_5)$	14.67
$MC_{[4:14]}(p_2)$	8.5
$MC_{[4:14]}(p_4)$	8.17
$MC_{[4:14]}(p_3)$	3
$MC_{[4:14]}(p_1)$	1.67

Interval waktu pencarian sangat mempengaruhi hasil kueri. Sebagai bukti, jika interval waktu pencarinya diubah dari 1 hingga 6, [1 : 6], maka hasil kueri 3 produk teratas yang paling menjanjikan adalah p_4 , p_1 , dan p_5 .

Tabel 3.8 Hasil Perhitungan Kontribusi Pasar Berbasis Interval Waktu (2)

$MC_{[1:6]}(p_4)$	5
$MC_{[1:6]}(p_1)$	4.67
$MC_{[1:6]}(p_5)$	3
$MC_{[1:6]}(p_2)$	1.33
$MC_{[1:6]}(p_3)$	0

3.3.3 Algoritme *Brute Force*

Algoritme *brute force* adalah algoritme yang mengimplementasikan metode *brute force*, yaitu melakukan percobaan terhadap semua kemungkinan dan hanya mengandalkan kekuatan pemrosesan komputer. Algoritme ini digunakan sebagai banding dari algoritme k -MPPTI.

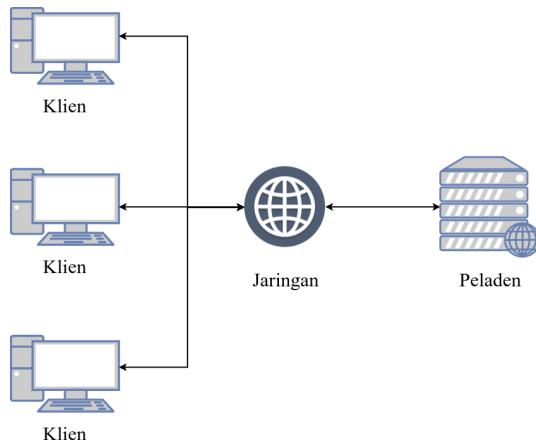
Secara garis besar, struktur data dan pendekatan yang digunakan hampir sama dengan k -MPPTI, namun tidak ada tahap komputasi *RSL*, sehingga komputasi *DSL* dilakukan pada semua $c \in C$ dengan cara membandingkan semua produk $p \in P$ satu persatu. Selain itu, komputasi tidak dilakukan dengan secara paralel.

3.3.4 Arsitektur Aplikasi

Sistem akan diimplementasikan menggunakan arsitektur *client-server* seperti yang diilustrasikan pada Gambar 3.15. Terdapat dua komponen utama dalam arsitektur ini, yaitu klien (*client*), pihak yang meminta atau menerima layanan, dan peladen (*server*), pihak yang memberikan atau mengirim layanan. Komponen-komponen ini terhubung ke jaringan, baik melalui kabel maupun nirkabel untuk melakukan transmisi data.

Klien mengimplementasikan antarmuka pengguna grafis atau APG (Inggris: *graphical user interface* atau GUI) berbasis web, sedangkan peladen mengimplementasikan *back-end service* yang berisi algoritme komputasi k -MPPTI yang terdiri atas algoritme *data precomputing* dan *query processing*. Web dibangun menggunakan Flask *microframework* dan bahasa Python, HTML, CSS, serta Javascript, sedangkan *back-end service* diimplementasikan menggunakan bahasa Python. Flask juga sekaligus berperan sebagai peladen web (*web server*).

Pengguna melakukan masukan data melalui web, kemudian



Gambar 3.15 Perancangan Arsitektur Aplikasi

data tersebut dikirimkan ke *back-end service* untuk dilakukan proses *data precomputing*. Setelah hasil *data precomputing* selesai, pengguna melakukan masukan kueri pencarian. Kueri pencarian tersebut akan dikirimkan ke *back-end service* untuk dilakukan proses *query processing*. Hasil yang didapatkan akan dikembalikan ke klien, disertai dengan visualisasi data.

BAB IV

IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai implementasi dari perancangan struktur data dan algoritme untuk menyelesaikan permasalahan *k-Most Promising Products (k-MPP)* berbasis interval waktu yang telah dijelaskan pada Bab III. Penjelasan implementasi terdiri dari penjelasan kelas dan fungsi yang dibuat, disertai dengan *pseudocode* untuk masing-masing fungsi tersebut.

4.1 Lingkungan Implementasi

Lingkungan implementasi dalam pembuatan Tugas Akhir ini meliputi perangkat keras dan perangkat lunak dengan spesifikasi sebagai berikut:

1. Perangkat Keras:
 - Prosesor 2.6 GHz Intel Core i5 (I5-4278U)
 - Memori 8 GB 1600 MHz DDR3
2. Perangkat Lunak:
 - Sistem operasi macOS Mojave Versi 10.14.5
 - *Text editor* Visual Studio Code Versi 1.33.1
 - Bahasa Pemrograman Python 3.7.3
 - Flask *Microframework* 1.0.2

Algorithm 1 DetermineGSP

- 1: **Input:** grid index G , a distance d_ε , uncertain data object X , action(insertion/deletion)
- 2: **Output:** an updated grid index G
- 3: create empty queue Q
- 4: create temporary graph Gr
- 5: access edge e enclosing X and enqueue n_i and n_j
- 6: enqueue n_i and n_j with each distance
- 7: **while** Q is not empty **do**
- 8: sort Q by distance from X
- 9: dequeue Q as n
- 10: **if** $d_{n,X} \leq d_\varepsilon$ **then**
- 11: insert grid enclosing n to Gr
- 12: **if** action is insertion **then** call *Insertion()*
- 13: **else** call *Deletion()*
- 14: **for all** node m as neighbor of n **do**
- 15: **if** m has not visited **then**
- 16: enqueue m with it's distance
- 17: mark m as visited
- 18: **for all** edge e which has updated n_s or n_e in Gr **do**
- 19: find GSP as gsp
- 20: call *ComputeTurningPoint(gsp)*

Gambar 4.1 Algoritme *DetermineGSP*

4.2 Implementasi Algoritme k-MPPTI

4.2.1 Implementasi Algoritme *Data Precomputing*

4.2.1.1 *Class*

4.2.2 Implementasi Algoritme *Query Processing*

4.3 Implementasi Algoritme Pembanding (*Brute Force*)

4.4 Implementasi Antarmuka Pengguna

Halaman ini sengaja dikosongkan

BAB V

UJI COBA DAN EVALUASI

Bab ini membahas hasil uji coba aplikasi yang telah dirancang dan diimplementasikan. Uji coba dilakukan untuk mengetahui kinerja aplikasi dengan lingkungan uji coba yang telah ditentukan.

5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan komponen-komponen yang terdiri dari:

1. Processor Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz x 4
2. RAM 6 GB

Semua pengujian menggunakan Memory Heap pada JVM sebanyak 4 GB (dengan opsi -Xmx4g).

5.2 Data Uji Coba

Terdapat tiga jenis data yang akan digunakan dalam pengujian Tugas Akhir ini. Tiga jenis data tersebut terdiri dari data *independent*, data *anticorrelated*, dan data *correlated*.

5.2.1 Data *Independent* (IND)

Data *independent* adalah sebuah data buatan dengan cara melakukan *random* secara utuh dengan jumlah yang telah ditentukan.

5.2.2 Data *Anticorrelated* (ANT)

Data *anticorrelated* adalah data yang memiliki hubungan negatif. Artinya, jika suatu nilai suatu atribut bertambah, maka atribut lain berkurang dengan rasio yang sama. Data *anticorrelated* ini memungkinkan setiap objek tidak mendominasi dan didominasi oleh objek lain.

5.2.3 Data *Correlated* (COR)

Data *correlated* adalah data yang memiliki hubungan erat. Dalam artian, jika nilai suatu atribut bertambah, maka atribut lain juga bertambah dengan rasio yang sama. Dalam kasus ini, suatu objek pasti mendominasi dan didominasi objek yang lain.

5.3 Skenario Uji Coba

Untuk menguji *performance* dari algoritme yang diusulkan, kami mengujinya dengan beberapa variasi pada faktor-faktor yang mempengaruhi jalannya algoritme, yaitu: jumlah objek, jumlah sel grid, jumlah *instance* tiap objek, dan jarak d_ε . Berikut kami sajikan tabel variasi faktor-faktor yang disebutkan beserta nilai *default*.

Tabel 5.1 Atribut dari objek

Parameter	Default	Rentang
Jumlah sel grid	256^2	$32^2, 64^2, 128^2, 256^2, 512^2$
Jumlah objek (K)	5	0.1, 1, 5, 10, 20
Jumlah <i>instance</i> tiap objek	50	10, 50, 100, 200, 400
d_ε (%)	1	0.1, 0.5, 1, 2, 3
Dimensi data	2	2, 3, 4, 5, 6

Uji coba dilakukan pada peta California[?] dengan

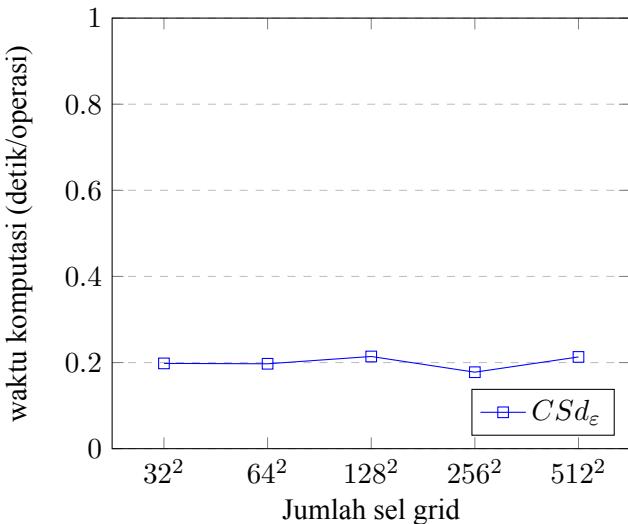
irisan garis lintang antara 32.0 hingga 37.0 dan garis bujur antara -120.0 hingga -114.0. Pada koordinat tersebut didapat *node* sebanyak 8716 dan *edge* sebanyak 9077.

5.3.1 Skenario Uji Coba *Performance*

Pada sub-bab ini, kami menampilkan tabel uji *performance* dalam bentuk grafik. Perlu diperhatikan bahwa beberapa grafik pada bab ini menggunakan skala logaritma pada sumbu y. Uji coba dilakukan dengan membandingkan dengan metode *naive*. Metode *naive* mencari jarak terdekat setiap *node* ketika terdapat objek baru masuk pada sistem dan memasukkan objek pada *node* yang berjarak kurang dari d_ε . Metode ini juga tidak menggunakan penanda *isImpossible*, artinya, jika terdapat objek baru masuk pada suatu *node*, maka metode ini membandingkan objek tersebut dengan semua objek yang ada.

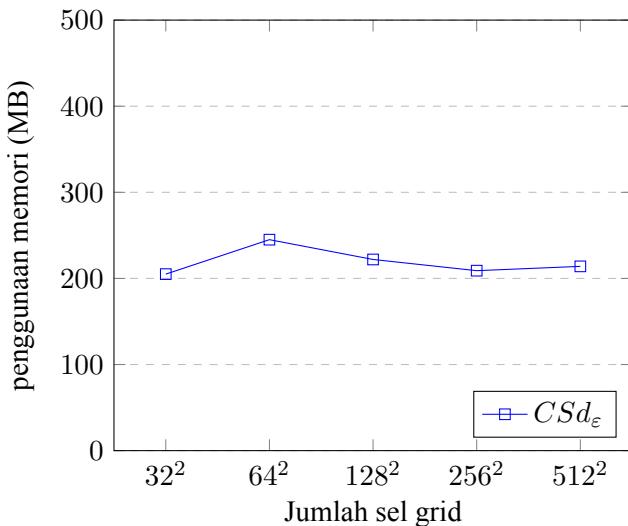
5.3.1.1 Skenario Uji Coba *Performance* Terhadap Perubahan Jumlah Sel

Perubahan jumlah sel pada grid indeks tidak memberikan pengaruh banyak pada waktu komputasi dan penggunaan memori. Hal tersebut dikarenakan, jika jumlah sel sedikit, maka proses pengambilan data semakin cepat. Tetapi di sisi lain, pemrosesan data melambat karena objek yang dimuat semakin banyak. Jika jumlah sel banyak, pengambilan data semakin lama, tetapi pemrosesan data semakin cepat karena objek yang dimuat semakin sedikit.



Gambar 5.1 Pengaruh jumlah sel terhadap waktu komputasi tiap operasi dalam satuan detik

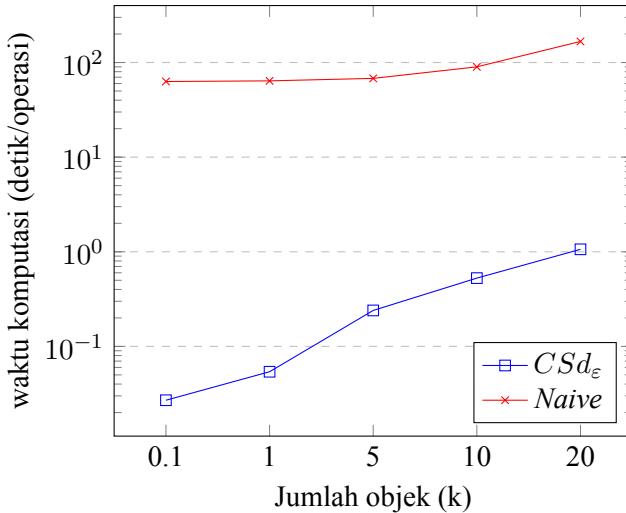
Jumlah sel tidak banyak mempengaruhi penggunaan memori dan waktu komputasi. Hal ini dikarenakan adanya *trade-off* antara proses memuat data dengan komputasi. Grid indeks yang memiliki sel sedikit menjadikan data yang dimuat lebih banyak sehingga menjadikan data yang diproses lebih banyak. Tetapi di sisi lain, sistem tidak banyak mencari data secara berulang-ulang karena setiap sel sudah mengaver area yang besar. Sedangkan grid indeks yang memiliki sel yang banyak menjadikan proses komputasi lebih efisien karena melibatkan data yang lebih sedikit. Tetapi di sisi lain, sistem harus melakukan pencarian data berulang-ulang karena sedikitnya data yang didapat pada setiap sel.



Gambar 5.2 Pengaruh jumlah sel grid terhadap penggunaan memori dalam satuan megabita

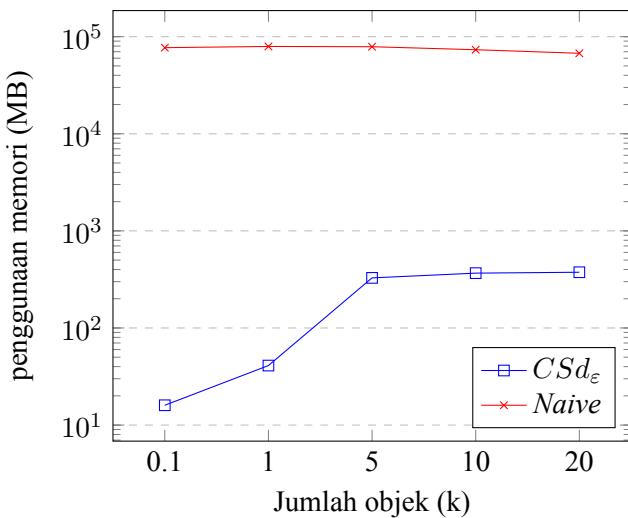
5.3.1.2 Skenario Uji Coba *Performance* Terhadap Perubahan Jumlah Objek

Penggunaan waktu CPU pada algoritme $CSd_\varepsilon - SQ$ jauh mengungguli algoritme *naive*. Hal ini dikarenakan algoritme *naive* menghitung jarak semua *node* dengan objek yang masuk menggunakan *shortest-path*, sedangkan $CSd_\varepsilon - SQ$ hanya menggunakan *node* yang diperlukan dan $CSd_\varepsilon - SQ$ tidak menggunakan algoritme *shortest-path*.



Gambar 5.3 Pengaruh jumlah objek terhadap waktu komputasi tiap operasi dalam satuan detik

Ketika jumlah objek bertambah, waktu pemrosesan juga bertambah, hal ini dikarenakan bertambahnya objek yang terdapat pada *node*. Dengan bertambahnya objek pada *node*, algoritme perlu membandingkan dengan objek yang lebih banyak untuk mencari probabilitas masing-masing objek menjadi *SP*.

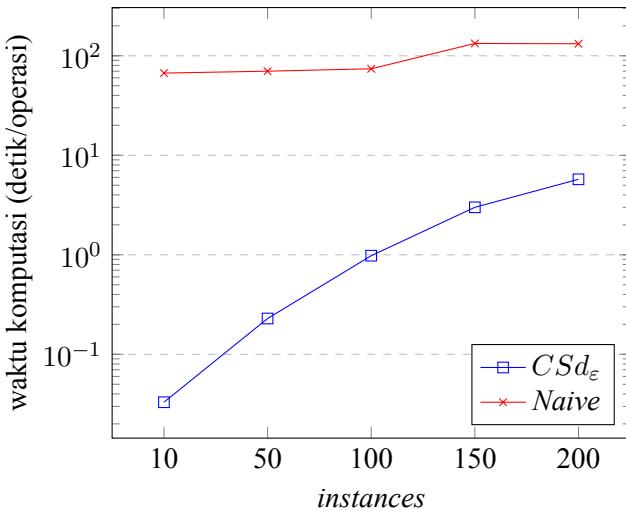


Gambar 5.4 Pengaruh jumlah objek terhadap penggunaan memori dalam satuan megabita

Terkait penggunaan memori, metode *naive* membutuhkan memori yang sangat banyak karena banyaknya *node* yang perlu diproses menggunakan algoritme *shortest-path*. Sedangkan metode $CSd_{\epsilon} - SQ$ membutuhkan memori yang tidak banyak karena hanya menggunakan data *node* yang diperlukan saja dengan struktur grid.

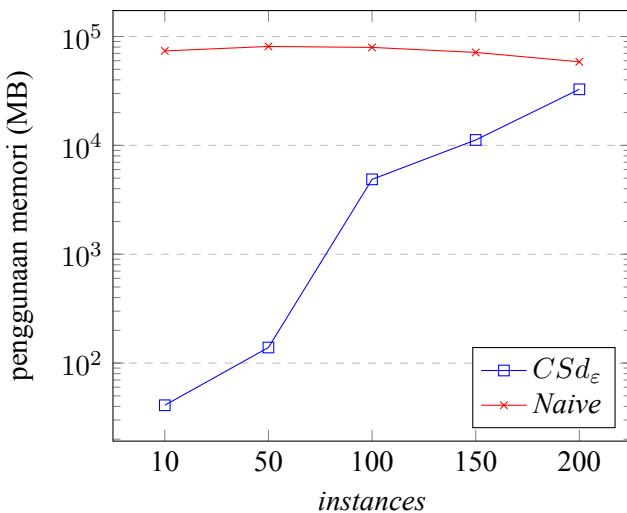
5.3.1.3 Skenario Uji Coba *Performance* Terhadap Perubahan Jumlah *Instance* pada Objek

Jumlah *instance* pada objek mempengaruhi waktu komputasi dan penggunaan memori. Hal ini dikarenakan proses penghitungan probabilitas melibatkan *instances* di objek. Dari sisi memori, banyaknya *instance* membuat sistem harus mengalokasikan memori lebih untuk proses penyimpanan dan komputasi.



Gambar 5.5 Pengaruh jumlah *instance* terhadap waktu komputasi tiap operasi dalam satuan detik

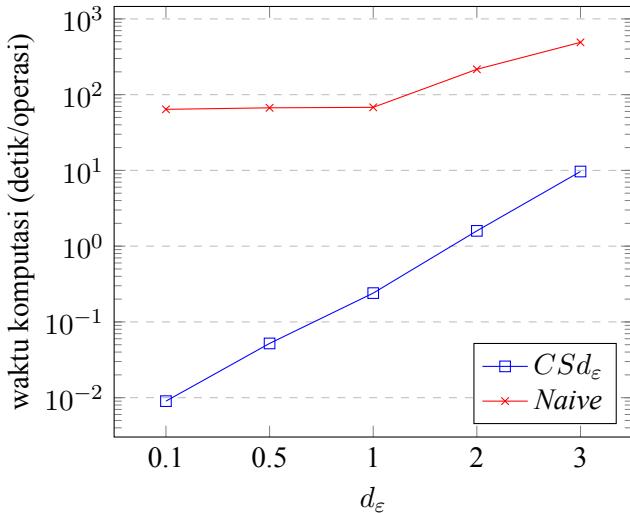
Pada metode *naive*, objek perubahan waktu komputasi terlihat ketika jumlah *instance* diatas 100. Hal ini dikarenakan waktu komputasi lebih banyak digunakan untuk penghitungan jarak terpendek dari setiap *node* ke objek, sehingga jumlah *instance* yang sedikit tidak berpengaruh banyak terhadap waktu komputasi.



Gambar 5.6 Pengaruh jumlah *instance* terhadap penggunaan memori dalam satuan megabita

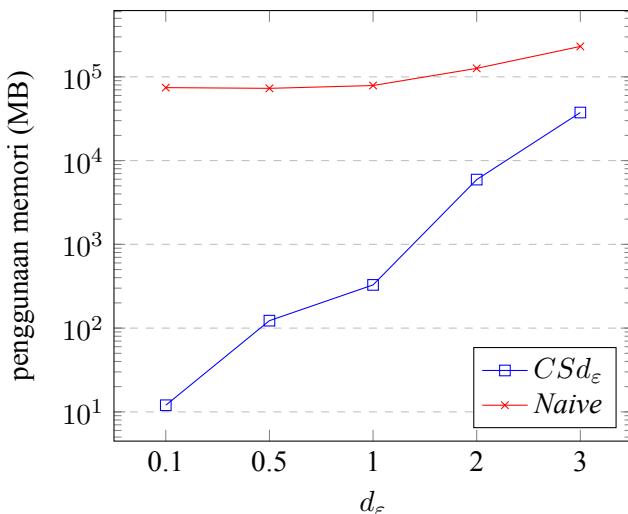
5.3.1.4 Skenario Uji Coba *Performance* Terhadap Perubahan Panjang Jarak Maksimal d_{ε}

Jarak d_{ε} sangat mempengaruhi *performance* karena d_{ε} menentukan jarak terjauh *node* yang dapat menyimpan objek baru. Dengan bertambahnya nilai d_{ε} , objek dapat menjangkau lebih banyak *node*. Dengan demikian, objek yang ditampung pada *node* menjadi semakin banyak. Dengan semakin banyaknya objek, proses penghitungan probabilitas *skyline* menjadi semakin lama karena harus menghitung banyak objek. Pada $CSd_{\varepsilon} - SQ$, semakin besar nilai d_{ε} , semakin banyak grid yang diakses sehingga membutuhkan waktu yang lebih banyak. Pada metode *naive*, terdapat perubahan waktu komputasi yang signifikan ketika nilai d_{ε} diatas 1.



Gambar 5.7 Pengaruh d_ε terhadap waktu komputasi tiap operasi dalam satuan detik

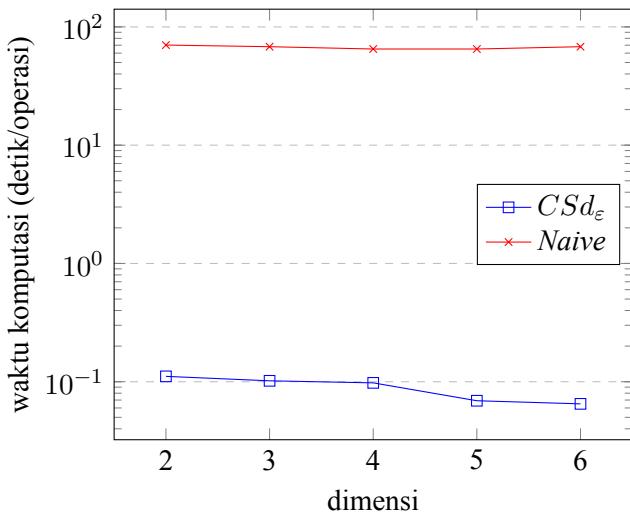
Penggunaan memori sangat tergantung dari jumlah objek yang diproses. Nilai d_ε yang besar menjadikan objek yang diproses semakin banyak karena setiap *node* memiliki jangkauan yang lebih jauh. Banyaknya objek yang diproses menjadikan penggunaan memori semakin besar.



Gambar 5.8 Pengaruh d_ε terhadap penggunaan memori dalam satuan megabita

5.3.1.5 Skenario Uji Coba *Performance* Terhadap Perubahan Dimensi Data

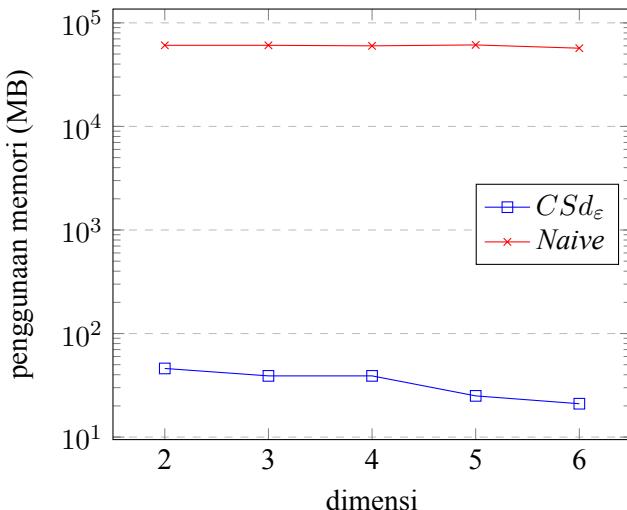
Dimensi dari *uncertain data* tidak banyak mempengaruhi *performance* dari algoritma. Pada Gambar 5.9, waktu komputasi pada algoritma CSd_ε cenderung turun. Hal tersebut dikarenakan antar objek tidak banyak yang mendominasi atau didominasi karena semakin banyaknya variabel yang perlu dibandingkan. Sebagai imbasnya, penggunaan memori semakin sedikit karena proses komputasi semakin sedikit.



Gambar 5.9 Pengaruh dimensi terhadap waktu komputasi tiap operasi dalam satuan detik

Secara umum objek yang memiliki semakin banyak dimensi menjadikan objek tersebut semakin sedikit *overlap*-nya dengan objek lain. Pada algoritma ini, proses penghitungan *skyline* hanya dilakukan pada objek-objek yang overlap dengan objek yang dicari probabilitasnya. Dengan demikian, semakin banyak dimensi menjadikan waktu komputasi semakin kecil.

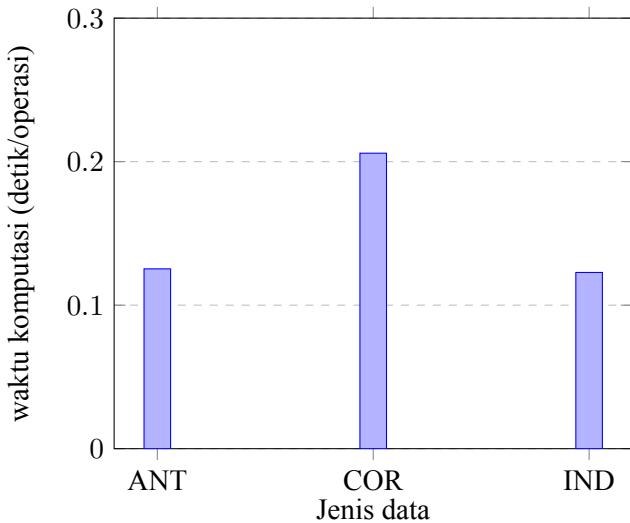
Pengalokasian memori berlaku pada objek-objek yang *overlap*. Semakin sedikit objek yang *overlap*, maka semakin sedikit memori yang dibutuhkan untuk pemrosesan probabilitas *skyline*.



Gambar 5.10 Pengaruh dimensi terhadap penggunaan memori dalam satuan megabita

5.3.1.6 Skenario Uji Coba *Performance* Terhadap Jenis Data

Data *anticorrelated* tidak banyak mempengaruhi komputasi karena pada penghitungan probabilitas *skyline* objek X , tidak banyak objek yang terdapat pada $PDD(X)$. Dengan data *correlated*, hampir setiap objek X memiliki objek yang overlap dengan $PDD(X)$, sehingga proses penghitungan probabilitas *skyline* menjadi lebih lama. Data *independence* tidak berbeda jauh dengan data *anticorrelated* karena objek yang terdapat pada *node* sedikit sehingga tidak banyak objek X yang memiliki probabilitas *skyline*.



Gambar 5.11 Pengaruh jenis data terhadap waktu komputasi tiap operasi dalam satuan detik

BAB VI

KESIMPULAN DAN SARAN

Pada bab ini dijelaskan mengenai kesimpulan dan saran dari hasil uji coba yang telah dilakukan.

6.1 Kesimpulan

Dari proses desain hingga uji coba, dapat diambil beberapa hasil sebagai berikut:

1. Tugas akhir ini mengusulkan struktur data grid indeks dan metode CSd_ε untuk pengolahan *skyline query* pada *uncertain data streaming* oleh titik bergerak dan objek tidak bergerak. Struktur data grid indeks memecah struktur data graf tradisional menjadi sel-sel yang berisi *node*, *edge*, dan objek. Penyimpanan objek dalam bentuk *SW-Tree* pada setiap *node* membuat proses komputasi lebih cepat.
2. Biaya komputasi pada metode CSd_ε jauh lebih baik dibandingkan metode *naive* dari sisi waktu komputasi dan penggunaan memori. Komputasi metode CSd_ε lebih cepat 600 kali dibandingkan metode *naive*. Dari sisi penggunaan memori, metode CSd_ε lebih hemat 1500 kali dibandingkan metode *naive*.

6.2 Saran

Berikut beberapa saran terkait pengembangan lebih lanjut:

1. Pendefinisian jarak d_ε dapat dilakukan secara dinamis. Apabila pencarian objek dengan jarak d_ε tidak menemukan

hasil yang diminta, jarak d_ε dapat diperbesar secara dinamis hingga mendapatkan hasil yang sesuai.

2. Pengembangan algoritma untuk memproses objek *uncertain* yang dapat bergerak secara dinamis.
3. Pada algoritme ini proses pembaruan *instance* dari *uncertain* objek dilakukan dengan menghapus dan menambahkan objek baru. Hal ini tentunya tidak efisien. Diperlukan algoritme pembaruan objek agar lebih efisien dalam hal waktu komputasi dan penggunaan memori.

DAFTAR PUSTAKA

- [1] M. S. Islam and C. Liu, "Know Your Customer: Computing K-Most Promising Products," *The VLDB Journal*, pp. 545–570, 2016.
- [2] D. Papadias, Y. Tao, G. Fu and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM Transactions on Database Systems*, Vol. 30, No. 1, pp. 41–82, 2005.
- [3] E. Dellis and B. Seeger, "Efficient Computation of Reverse Skyline Queries," *VLDB Endowment*, pp. 291-302, 2007.
- [4] B. Jiang and J. Pei, "Online Interval Skyline Queries on Time Series," *IEEE International Conference on Data Engineering*, pp. 1036-1047, 2009.
- [5] M. Golfarelli and S. Rizzi, "Introduction to Data Warehousing," in *Data Warehouse Design: Modern Principles and Methodologies*, New York: McGraw-Hill, 2009, pp. 1-42.
- [6] S. Borzsonyi, D. Kossmann and K. Stocker, "The Skyline Operator," In: *ICDE*, pp. 421-430, 2001.
- [7] L. Zou, L. Chen, M. T. Özsü and D. Zhao, "Dynamic Skyline Queries in Large Graphs," *DASFAA'10 Proceedings of the 15th International Conference on Database Systems for Advanced Applications - Volume Part II*, pp. 62-78, 2010.
- [8] X. Wu, Y. Tao, R. C.-W. Wong, L. Ding and J. X. Yu, "Finding the Influence Set through Skylines," *EDBT*, pp. 1030-1041, 2009.

- [9] G.S. Almasi and A. Gottlieb, *Highly Parallel Computing*. Redwood City, CA: Benjamin-Cummings Publishers, 1989.
- [10] Python.org. "What is Python? Executive Summary". [Online]. Available: <https://www.python.org/doc/essays/blurb/>. [Diakses: 18 Mei 2019].
- [11] Merriam-webster.com. "Definition of DATA". [Online]. Available: <https://www.merriam-webster.com/dictionary/data>. [Diakses: 13 Mei 2019]

LAMPIRAN A

KODE SUMBER

```
1 package ta.grid
2
3 case class Edge(id: Int, i: Int, j: Int, length: Double, objects: Set[Object]) {}
```

Kode Sumber 1.1 Kode sumber kelas *Edge*

```
1 package ta.grid
2
3 import collection.spatial.{HyperPoint, RTree}
4 import ta.geometry.Point2d
5
6 sealed abstract class AbstractNode
7
8 case class Node(id: Int, x: Double, y: Double, var tree: RTree[Point2d],
9 var objects: Set[Object]) extends AbstractNode
```

Kode Sumber 1.2 Kode sumber kelas *Node*

```
1 package ta.grid
2
3 import ta.geometry.{Point2d, Rect2d}
4
5 case class Object(id: Int, edgeId: Int, var skyProb: Double,
6 isImpossible: Boolean, nodeId: Int, rect: Rect2d, distance: Double,
7 position: Double) {
8   def points(grid: Grid) : List[Point2d] = {
9     grid.getRawObject(this.id).get.points
10   }
11
12   def asImpossible(): Object = {
13     Object(id, edgeId, skyProb, isImpossible = true, nodeId, rect, distance, position)
14   }
15
16   def updateSkyProb(newSkyProb: Double): Object = {
17     Object(id, edgeId, newSkyProb, isImpossible, nodeId, rect, distance, position)
18   }
19 }
```

Kode Sumber 1.3 Kode sumber kelas *Object*

```
1 package ta.algoritm
2
```

```

3   import collection.spatial.RTree
4   import scalax.collection.Graph
5   import scalax.collection.edge.WLkUnDiEdge
6   import ta.graph.TempGraph
7   import ta.grid._
8   import ta.stream.{ExpiredObject, RawObject, Stream}
9   import ta.Constants._
10  import ta.geometry.{Point2d, Rect2d}
11  import ta.algorithm.TurningPoint._
12
13  import scala.collection.JavaConverters._
14  import scala.collection.immutable.Set
15  import scala.collection.mutable
16
17  case class NodeQueue(nodeId: Int, distance: Double)
18
19  object TheAlgorithm {
20    def TheAlgorithm(grid: Grid, stream: Stream): Grid = {
21      var Q: mutable.Queue[NodeQueue] = mutable.Queue[NodeQueue]()
22      val tempGraph = new TempGraph
23
24      var visitedNodes: Set[Int] = Set()
25      var updatedNodes: Set[Int] = Set()
26
27      val rawObject = stream match {
28        case _rawObject: RawObject =>
29          grid.addObjectToEdge(_rawObject)
30          grid.addRawObject(_rawObject)
31          _rawObject
32        case ExpiredObject(id) =>
33          val _rawObject = grid.getRawObject(id).get
34          _rawObject
35      }
36
37      val objectList: java.util.List[Point2d] = rawObject.points.toList.asJava
38      val rect = new Rect2d(objectList)
39
40      var addedGrid: Set[GridLocation] = Set()
41
42      val edge = grid.getEdge(rawObject.edgeId).get
43      val nodei = grid.getNode(edge.i).get
44      val nodej = grid.getNode(edge.j).get
45
46      val gridNodeI = grid.getGridLocation(nodei)
47      val gridNodeJ = grid.getGridLocation(nodej)
48
49      val EdgesNodes(edgesNodeI, nodesNodeI) = grid.getDataGrid(gridNodeI)
50      tempGraph.addEdge(nodesNodeI, edgesNodeI)
51      addedGrid += gridNodeI
52
53      if (!addedGrid.contains(gridNodeJ)) {
54        val EdgesNodes(edgesNodeJ, nodesNodeJ) = grid.getDataGrid(gridNodeJ)
55        tempGraph.addEdge(nodesNodeJ, edgesNodeJ)
56        addedGrid += gridNodeJ
57      }
58
59      val e = tempGraph.getEdge(rawObject.edgeId)
60      val distanceNodeI = e.length * rawObject.position
61      val distanceNodeJ = e.length * (1 - rawObject.position)
62
63      Q.enqueue(NodeQueue(nodei.id, distanceNodeI))
64      visitedNodes = visitedNodes + nodei.id
65      Q.enqueue(NodeQueue(nodej.id, distanceNodeJ))
66      visitedNodes = visitedNodes + nodej.id

```

```

67
68     while (Q.nonEmpty) {
69         Q = Q.sortBy(_.distance)
70         val NodeQueue(currentNodeId, distance) = Q.dequeue()
71
72         if (distance < D_EPSILON) {
73             val currentNode = grid.getNode(currentNodeId).get
74
75             val gridLocation = grid.getGridLocation(currentNode)
76
77             if (!addedGrid.contains(gridLocation)) {
78                 val EdgesNodes(edgesN, nodesN) = grid.getDataGrid(gridLocation)
79                 tempGraph.addEdges(nodesN, edgesN)
80                 addedGrid += gridLocation
81             }
82
83             val updatedNode = stream match {
84                 case _: RawObject =>
85                     insertToNode(grid, currentNode, rawObject, distance, rect)
86                 case ExpiredObject(objectId) =>
87                     deleteFromNode(grid, currentNode, objectId, rect)
88             }
89
90             tempGraph.updateNode(updatedNode)
91             grid.updateNode(updatedNode)
92             updatedNodes += updatedNode.id
93
94             val neighborNodesEdges = tempGraph.getNeighborNodesEdges(currentNodeId)
95
96             neighborNodesEdges.keys.foreach { nodeId =>
97                 if (!visitedNodes.contains(nodeId)) {
98                     val distanceUnvisitedNode = distance + neighborNodesEdges(nodeId)
99                     Q.enqueue(NodeQueue(nodeId, distanceUnvisitedNode))
100                     visitedNodes += nodeId
101                 }
102             }
103         }
104     }
105
106     computeTurningPoint(grid, tempGraph.edgesGraph, tempGraph.nodesGraph, updatedNodes)
107
108     if (stream.isInstanceOf[ExpiredObject]) {
109         grid.removeObjectFromEdge(stream.getId)
110         grid.removeRawObject(stream.getId)
111     }
112
113     grid
114 }
115
116 def updateGrid(grid: Grid, graph: Graph[Node, WLkUnDiEdge]): Grid = {
117     grid.updateNodes(graph.nodes.toOuter)
118
119     grid
120 }
121
122 def computeTurningPoint(grid: Grid, edges: mutable.Map[Int, Edge],
123 nodes: mutable.Map[Int, Node], updatedNodes: Set[Int]): Unit = {
124     edges.values
125         .filter(e => updatedNodes.contains(e.i) | updatedNodes.contains(e.j))
126         .foreach { e =>
127             val nodeS = nodes(e.i)
128             val nodeE = nodes(e.j)
129
130             processLandmark(grid, nodeS, e, nodeE)

```

```

31         }
32     }
33
34     def deleteFromNode(grid: Grid, currentNode: Node, objectId: Int, rect: Rect2d): Node = {
35       val objectMaybe = currentNode.objects.find(_.id == objectId)
36
37       if (objectMaybe.isEmpty) {
38         return currentNode
39       }
40
41       val toBeDeletedPoints = objectMaybe.get.points(grid)
42
43       for (point <- toBeDeletedPoints) {
44         currentNode.tree.remove(point)
45       }
46
47       var overlappedObjects = findPDROverlappedObjects(currentNode, rect)
48
49       overlappedObjects = overlappedObjects.map {
50         case q@Object(_, _, _, true, _, _, _) => q
51         case q@Object(_, _, _, _, _, _, _) =>
52           val skyProb = SkyPrX(currentNode.tree, q.id)
53           q.updateSkyProb(skyProb)
54       }
55
56       var newObjects = currentNode.objects.map { o =>
57         val updatedObjectMaybe = overlappedObjects.find(_.id == o.id)
58
59         updatedObjectMaybe match {
60           case None => o
61           case Some(obj) => obj
62         }
63       }
64
65       newObjects = newObjects.filterNot(_.id == objectId)
66
67       Node(currentNode.id, currentNode.x, currentNode.y, currentNode.tree, newObjects)
68     }
69
70     def insertToNode(grid: Grid, node: Node,
71                      rawObject: RawObject,
72                      distance: Double,
73                      rect: Rect2d): Node = {
74
75       var overlappedObjects = findPDROverlappedObjects(node, rect)
76
77       rawObject.points.foreach(p => node.tree.add(p))
78
79       val updatedOverlappedObjects = overlappedObjects.map(q => {
80         val ddrRect = rect.getDDR.asInstanceOf[Rect2d]
81         val qRect = q.rect
82
83         if (ddrRect.contains(qRect) & distance < q.distance) {
84           q.asImpossible()
85         } else {
86           val objProb = getDominationProbability(node.tree, ddrRect, q.id)
87           if (objProb > (1 - P_THRESHOLD) & distance < q.distance) {
88             q.asImpossible()
89           } else {
90             val skyProb = SkyPrX(node.tree, q.id)
91             q.updateSkyProb(skyProb)
92           }
93         }
94       })

```

```

195
196     val updatedObjects = node.objects.map(o => {
197         val updated = updatedOverlappedObjects.find(_.id == o.id)
198
199         if (updated.nonEmpty)
200             updated.get
201         else
202             o
203     })
204
205     val PDDoverlappedObjects = findPDDOverlappedObjects(node, rect, rawObject.id)
206     val pddTree = new RTree(new Point2d.Builder, 2, 8, RTree.Split.AXIAL)
207     PDDoverlappedObjects.foreach { o =>
208         val points = grid.getRawObject(o.id).get.points
209         points.foreach { p =>
210             pddTree.add(p)
211         }
212     }
213     val skyProbU = SkyPrX(pddTree, rawObject.id)
214
215     val finalObjects = updatedObjects +
216         Object(rawObject.id, rawObject.edgeId, skyProbU, isImpossible = false,
217             node.id, rect, distance, rawObject.position)
218     Node(node.id, node.x, node.y, node.tree, finalObjects)
219 }
220
221 def SkyPrX(tree: RTree[Point2d], objectId: Int): Double = {
222     val X = scala.collection.mutable.Set[Point2d]()
223     tree.forEach { p =>
224         if (p.o == objectId) X.add(p)
225     }
226
227     X.map(x => x.p * SkyPrx(tree, X, x))
228     .sum
229 }
230
231 def SkyPrx(tree: RTree[Point2d], X: mutable.Set[Point2d], x: Point2d): Double = {
232     val entries = mutable.Set[Point2d]()
233     tree.forEach(p => entries.add(p))
234
235     entries
236         .toList
237         .filterNot(e => X.contains(e))
238         .groupBy(_.o)
239         .values
240         .map(Y => PrYnotdominatem(Y, x))
241         .product
242     }
243
244 def PrYnotdominatem(Y: List[Point2d], x: Point2d): Double = {
245     Y.filter(y => isyNotDominatex(y, x))
246     .foldLeft(0.0)((acc, e) => acc + e.p)
247 }
248
249 def isyNotDominatex(y: Point2d, x: Point2d): Boolean = {
250     if (y.x <= x.x & y.y <= x.y) {
251         false
252     } else {
253         true
254     }
255 }
256
257 def getDominationProbability(tree: RTree[Point2d], ddrRect: Rect2d, objectId: Int):
258     Double = {

```

```

159     val result = new Array[Point2d](N_POINTS)
160     tree.search(ddrRect, result)
161
162     result
163       .filter(_.isInstanceOf[Point2d])
164       .filter(_.o == objectId)
165       .foldLeft(0.0)((acc, e) => acc + e.p)
166   }
167
168
169 def findPDROverlappedObjects(node: Node, rect: Rect2d): Set[Object] = {
170   val tree = node.tree
171   val PDRBox: Rect2d = rect.getPDR.asInstanceOf[Rect2d]
172   val overlappedPoints: Array[Point2d] = new Array[Point2d](N_POINTS)
173   tree.search(PDRBox, overlappedPoints)
174
175   val objectIds = overlappedPoints.toList
176   .filter(_.isInstanceOf[Point2d]).map(_.o).toSet
177   val objects = objectIds.map(id => {
178     val a = node.objects.find(_.id == id)
179     a.get
180   })
181
182   objects
183 }
184
185 def findPDDOverlappedObjects(node: Node, rect: Rect2d, currentId: Int)
186   : Set[Object] = {
187   val tree = node.tree
188   val pddBox: Rect2d = rect.getPDD.asInstanceOf[Rect2d]
189   val overlappedPoints: Array[Point2d] = new Array[Point2d](N_POINTS)
190   tree.search(pddBox, overlappedPoints)
191
192   val objectIds = overlappedPoints.toList
193   .filter(_.isInstanceOf[Point2d])
194   .map(_.o).toSet - currentId
195
196   val objects = objectIds.map(id => {
197     val a = node.objects.find(_.id == id)
198     a.get
199   })
200
201   objects
202 }
203 }
```

Kode Sumber 1.4 Kode sumber pemrosesan utama

```

1 package ta.algorithm
2
3 import collection.spatial.RTree
4
5 import scala.collection.mutable
6 import ta.grid.Node
7 import ta.grid.Edge
8 import ta.grid.Object
9 import ta.Constants._
10 import ta.geometry._
11 import ta.landmark._
12 import ta.algorithm.TheAlgorithm._
13 import ta.grid.Grid
```

```

14
15 import scala.collection.JavaConverters._
16
17 case class TP(dStart: Double, dEnd: Double, SP: Set[Object])
18
19 object TurningPoint {
20   def processLandmark(grid: Grid, nodeS: Node, edge: Edge, nodeE: Node): Unit = {
21     val spNodesS = nodeS.objects.filter(o => !o.isImpossible &
22                                         o.skyProb >= P_THRESHOLD)
23     val spNodeE = nodeE.objects.filter(o => !o.isImpossible &
24                                         o.skyProb >= P_THRESHOLD)
25     val Se = edge.objects
26
27     val findsimilar = (objects: Set[Object], obj: Object) =>
28       objects.find(o => o.id == obj.id & o != obj)
29
30     val SP = spNodesS ++ spNodeE ++ Se
31
32     val dataPoints = SP.map { o =>
33       o.id -> grid.getRawObject(o.id).get.points
34     }.toMap
35
36     val Q = mutable.Queue[Landmark]()
37
38     def findDominatedObjects(obj: Object, objects: Set[Object]): Set[Object] = {
39       objects.filter(o => {
40         val pointsL = dataPoints(obj.id)
41         val points = dataPoints(o.id)
42
43         val tree = new RTree(new Point2d.Builder(), 2, 8, RTree.Split.AXIAL)
44
45         pointsL.foreach(p => tree.add(p))
46         points.foreach(p => tree.add(p))
47
48         val ddrRect = new Rect2d(pointsL.asJava).getDDR.asInstanceOf[Rect2d]
49
50         val objProb = getDominationProbability(tree, ddrRect, o.id)
51
52         if (objProb > 1 - P_THRESHOLD)
53           true
54         else
55           false
56       })
57     }
58
59     val SeId = Se.map(_.id)
60
61     Se.foreach { obj =>
62       val llDistance = obj.distance - D_EPSILON
63       if (llDistance >= 0 & llDistance <= edge.length) {
64         val landmarkLeft =
65           new LandmarkLeft(obj.distance - D_EPSILON, Some(edge.id), obj.id)
66         Q.enqueue(landmarkLeft)
67       }
68
69       val lrDistance = obj.distance + D_EPSILON
70       if (lrDistance <= edge.length & lrDistance >= 0) {
71         val landmarkRight =
72           new LandmarkRight(obj.distance + D_EPSILON, Some(edge.id), obj.id)
73         Q.enqueue(landmarkRight)
74       }
75     }
76   }
77   spNodesS

```

```

78     .filterNot(o => SeId.contains(o.id))
79     .foreach { obj =>
80       val distance = findDistanceFromNodeS(obj, edge, spNodeE)
81       val distanceLandmark = D_EPSILON + distance
82       if (distanceLandmark >= 0 & distanceLandmark <= edge.length) {
83         val landmarkRight =
84           new LandmarkRight(distanceLandmark, Some(edge.id), obj.id)
85         Q.enqueue(landmarkRight)
86       }
87     }
88
89   spNodeE
90     .filterNot(o => SeId.contains(o.id))
91     .foreach { obj =>
92       val distanceObject = findDistanceFromNodeE(obj, edge, spNodeS)
93       val distanceLandmark = distanceObject - D_EPSILON
94       if (distanceLandmark >= 0 & distanceLandmark <= edge.length) {
95         val landmarkLeft =
96           new LandmarkLeft(distanceLandmark, Some(edge.id), obj.id)
97
98         val similar = findSimilar(spNodeS, obj)
99         if (similar.isDefined) {
100           if (math.abs(similar.get.distance - obj.distance) != edge.length) {
101             Q.enqueue(landmarkLeft)
102           }
103         } else {
104           Q.enqueue(landmarkLeft)
105         }
106       }
107     }
108
109 SP.foreach { o =>
110   val distanceO = findDistance(o, edge, spNodeS, spNodeE)
111
112   val dominatedObjects = findDominatedObjects(o, SP.filterNot(_.id == o.id))
113   dominatedObjects.foreach { dominatedObj =>
114     val distanceDominatedObj = findDistance(dominatedObj, edge, spNodeS, spNodeE)
115     var distanceLandmark: Double = -1.0
116
117     if (distanceO > distanceDominatedObj) {
118       // landmark mid left
119       distanceLandmark = -1.0
120       if (spNodeS.contains(dominatedObj) & !SeId.contains(dominatedObj.id)) {
121         distanceLandmark = (distanceO - distanceDominatedObj)/2
122       } else if (SeId.contains(dominatedObj.id)) {
123         distanceLandmark = (distanceO + distanceDominatedObj)/2
124       }
125
126     if (distanceLandmark >= 0 & distanceLandmark <= edge.length) {
127       val landmark =
128         new LandmarkLeftMid(distanceLandmark, Some(edge.id), o.id, dominatedObj.id)
129
130       val isExists = Q.exists {
131         case mid: LandmarkLeftMid =>
132           if (mid.objId == o.id & mid.objDominatedId == dominatedObj.id) {
133             true
134           } else {
135             false
136           }
137         case _ =>
138           false
139       }
140
141       if (!isExists) {

```

```

142         Q.enqueue(landmark)
143     }
144   }
145 } else {
146   // landmark mid right
147   distanceLandmark = -1.0
148   if (spNodeE.contains(dominatedObj) & !SeId.contains(dominatedObj.id)) {
149     distanceLandmark = (distanceDominatedObj - distanceO)/2
150   } else if (SeId.contains(dominatedObj.id)) {
151     distanceLandmark = (distanceO + distanceDominatedObj)/2
152   }
153
154   if (distanceLandmark >= 0 & distanceLandmark <= edge.length) {
155     val landmark =
156       new LandmarkRightMid(distanceLandmark, Some(edge.id), o.id, dominatedObj.id)
157
158     val isExists = Q.exists {
159       case mid: LandmarkRightMid =>
160         if (mid.objId == o.id & mid.objDominatedId == dominatedObj.id) {
161           true
162         } else {
163           false
164         }
165       case _ =>
166         false
167     }
168
169     if (!isExists) {
170       Q.enqueue(landmark)
171     }
172   }
173 }
174
175 processLandmark(Q.toList, spNodeS, spNodeE, edge)
176 }
177
178 def findDistanceFromNodeS(obj: Object, edge: Edge, spNodeE: Set[Object]): Double = {
179   if (obj.edgeId == edge.id) {
180     edge.length * obj.position
181   } else {
182     val objInSpNodeEMaybe = spNodeE.find(_.id == obj.id)
183     if (objInSpNodeEMaybe.isDefined) {
184       if (objInSpNodeEMaybe.get.distance < obj.distance) {
185         obj.distance + edge.length
186       } else {
187         obj.distance * -1
188       }
189     } else {
190       obj.distance * -1
191     }
192   }
193 }
194
195 def findDistanceFromNodeE(obj: Object, edge: Edge, spNodeS: Set[Object]): Double = {
196   if (obj.edgeId == edge.id) {
197     edge.length * obj.position
198   } else {
199     val objInSpNodeSMaybe = spNodeS.find(_.id == obj.id)
200     if (objInSpNodeSMaybe.isDefined) {
201       if (objInSpNodeSMaybe.get.distance < obj.distance) {
202         obj.distance * -1
203       } else {
204     }
205   }

```

```
206         obj.distance + edge.length
207     }
208   } else {
209     obj.distance + edge.length
210   }
211 }
212 }
213
214 def findDistance(obj: Object, edge: Edge, spNodeS: Set[Object],
215   spNodeE: Set[Object]): Double = {
216   if (obj.edgeId == edge.id) {
217     edge.length * obj.position
218   } else {
219     if (spNodeS.contains(obj)) {
220       findDistanceFromNodeS(obj, edge, spNodeE)
221     } else {
222       findDistanceFromNodeE(obj, edge, spNodeS)
223     }
224   }
225 }
226
227 def processLandmark(landmarks: List[Landmark], spNodeS: Set[Object],
228   spNodeE: Set[Object], edge: Edge): Unit = {
229   val sortedLandmarks = landmarks.sortBy(_.distance)
230   val objects = spNodeS ++ spNodeE ++ edge.objects
231
232   var SP = spNodeS
233
234   val filteredLandmarks = sortedLandmarks.filterNot(_.distance < 0)
235   var queue = scala.collection.mutable.Queue[Landmark](filteredLandmarks: _*)
236
237   var dStart: Double = 0
238   var dEnd: Double = edge.length
239
240   var turningPointList = Vector[TP]()
241
242   def isObjectInSP(SP: Set[Object], objId: Int): Boolean = {
243     SP.exists(_.id == objId)
244   }
245
246   while (queue.nonEmpty) {
247     val l = queue.dequeue()
248
249     l match {
250       case _: LandmarkLeft =>
251         val isLandmarkRightMidExist = queue.exists {
252           case a: LandmarkRightMid =>
253             a.objDominatedId == l.objId
254           case _ =>
255             false
256         }
257
258         val landmarkRightMaybe = queue.find { landmark =>
259           landmark.isInstanceOf[LandmarkRight] & landmark.objId == l.objId
260         }
261
262         if (landmarkRightMaybe.isDefined) {
263           queue = queue.filterNot(_ == landmarkRightMaybe.get)
264         } else {
265           if (!isLandmarkRightMidExist) {
266             val obj = objects.find(_.id == l.objId).get
267             turningPointList = turningPointList :+ TP(dStart, l.distance, SP)
268           }
269         }
270       }
271     }
272   }
273 }
```

```

$70          dStart = l.distance
$71          SP = SP + obj
$72      }
$73  }
$74  case : LandmarkRight =>
$75    if !(isObjectInSP(SP, l.objId)) {
$76      val obj = SP.find(_.id == l.objId).get
$77
$78      turningPointList = turningPointList :+ TP(dStart, l.distance, SP)
$79
$80      dStart = l.distance
$81      SP = SP - obj
$82    }
$83  case landmark: LandmarkLeftMid =>
$84
$85    if (isObjectInSP(SP, landmark.objId)
$86      & isObjectInSP(SP, landmark.objDominatedId)) {
$87      val objDominated = SP.find(_.id == landmark.objDominatedId).get
$88
$89      turningPointList = turningPointList :+ TP(dStart, l.distance, SP)
$90
$91      dStart = l.distance
$92      SP = SP - objDominated
$93    }
$94  case _ =>
$95    val landmark = l.asInstanceOf[LandmarkRightMid]
$96
$97    val isLandmarkRightMidExist = queue.exists {
$98      case a: LandmarkRightMid =>
$99        a.objDominatedId == landmark.objDominatedId
$100     case _ =>
$101       false
$102   }
$103
$104   if (isObjectInSP(SP, l.objId) & !isLandmarkRightMidExist) {
$105     val objDominated = objects.find(_.id == landmark.objDominatedId).get
$106
$107     turningPointList = turningPointList :+ TP(dStart, l.distance, SP)
$108
$109     dStart = l.distance
$110     SP = SP + objDominated
$111   }
$112 }
$113 }
$114
$115 turningPointList = turningPointList :+ TP(dStart, dEnd, SP)
$116 }
$117 }
```

Kode Sumber 1.5 Kode sumber penentuan *turning-point*

```

1 package ta.graph
2
3 import scalax.collection.immutable.Graph
4 import scalax.collection.edge.WLkUnDiEdge
5 import ta.grid.{Edge, Node}
6
7 import scala.collection.mutable
8
9 class TempGraph {
10   var graphInt: Graph[Int, WLkUnDiEdge] = Graph()
```

```

11  var nodesGraph: scala.collection.mutable.Map[Int, Node] = mutable.Map()
12  var edgesGraph: scala.collection.mutable.Map[Int, Edge] = mutable.Map()
13
14  def getNeighborNodesEdges(nodeId: Int): Map[Int, Double] = {
15    graphInt.get(nodeId).edges.map { e =>
16      if (e._1.toOuter == nodeId) {
17        e._2.toOuter -> e.weight
18      } else {
19        e._1.toOuter -> e.weight
20      }
21    }.toMap
22  }
23
24  def updateNode(node: Node): Unit = {
25    this.nodesGraph(node.id) = node
26  }
27
28  def getEdge(edgeId: Int): Edge = {
29    this.edgesGraph(edgeId)
30  }
31
32  def addEdges(nodes: Set[Node], edges: Set[Edge]): Unit = {
33    nodes.foreach { n =>
34      this.nodesGraph(n.id) = n
35    }
36
37    edges.foreach { e =>
38      this.graphInt = graphInt + WLkUnDiEdge(e.i, e.j)(e.length, e.id)
39      this.edgesGraph(e.id) = e
40    }
41  }
42
43  def getNode(nodeId: Int): Option[Node] = {
44    this.nodesGraph.get(nodeId)
45  }
46}

```

Kode Sumber 1.6 Kode sumber graf sementara

```

1 package ta.grid
2
3 import java.io.FileWriter
4
5 import scala.collection.immutable.Set
6 import ta.stream.RawObject
7 import ta.{RawEdge, RawNode}
8 import collection.spatial.RTree
9 import ta.algorithm.TheAlgorithm.SkyPrX
10 import ta.grid.Rect._
11 import ta.Constants._
12 import ta.geometry.Point2d
13
14 import scala.math.{floor, round}
15
16 case class EdgesNodes(edges: Set[Edge], nodes: Set[Node])
17 case class Table(edges: Set[Int], nodes: Set[Int])
18
19 class Grid extends Cloneable {
20   private var edges: Set[Edge] = Set()
21   var nodes: Set[Node] = Set()
22   private var rawObjects: Set[RawObject] = Set()

```

```

13
14
15
16
17
18
19
20
21
22
23
24 def createDataNaive = {
25   val filename = "n"+N_OBJECTS+"np"+N_POINTS+"g"+N_GRID_CELL+"d"+PERCENT_DISTANCE+"p"+P_THRESHOLD
26   val fwclear = new FileWriter("import/grid-obj-"+filename+".txt")
27   fwclear.close()
28   rawObjects.foreach { ro =>
29     /*
30      id edgeId pos sizePoints
31      foreach sizePoints
32        px py pp po
33     */
34     val fw = new FileWriter("import/grid-obj-"+filename+".txt", true)
35     fw.write(
36       ro.id + "u" + ro.edgeId + "u" + ro.position + "u" + ro.points.size + "u"
37     )
38     var strp = ""
39     ro.points.foreach { p =>
40       strp += p.x + "u" + p.y + "u" + p.p + "u" + p.o + "u"
41     }
42     fw.write(strp + "\n")
43     fw.close()
44   }
45
46   val fw2clear = new FileWriter("import/grid-node-"+filename+".txt")
47   fw2clear.close()
48   nodes.foreach { n =>
49     val fw2 = new FileWriter("import/grid-node-"+filename+".txt", true)
50     /*
51       nodeID objectSize
52       foreach objectSize
53         id distance skyprob edgeId pos
54       */
55     fw2.write(
56       n.id + "u" + n.objects.size + "u"
57     )
58     var str = ""
59     n.objects.foreach { o =>
60       str += o.id + "u" + o.distance + "u" + o.skyProb + "u" + o.edgeId + "u" + o.position + "u"
61     }
62     fw2.write(str + "\n")
63     fw2.close()
64   }
65 }
66
67 def updateAllSkyProb(): Unit = {
68   this.nodes = this.nodes.par.map { node =>
69     println(node.id)
70     node.objects = node.objects.map { o =>
71       if (o.isImpossible) {
72         o
73       } else {
74         o.skyProb = SkyPrX(node.tree, o.id)
75         o
76       }
77     }
78     node
79   }.toList.toSet
80 }
81
82 var tableGrid: Map[Int, Map[Int, Table]] = Map()
83
84 def getRawObjectobjectId: Int): Option[RawObject] = rawObjects.par.find(_.id == objectId)
85 def addRawObject(rawObject: RawObject): Unit =
86

```

```

87      this.rawObjects = this.rawObjects + rawObject
88    def removeRawObject(objectId: Int): Unit = {
89      val rawObject = this.rawObjects.par.find(_.id == objectId).get
90      this.rawObjects = this.rawObjects - rawObject
91    }
92
93    def addRawNodes(nodes: Set[RawNode]): Unit = {
94      nodes.map(raw => {
95        val emptyTree = new RTree(new Point2d.Builder(), 2, 8, RTree.Split.AXIAL)
96        Node(raw.id, raw.x, raw.y, emptyTree, Set())
97      }).foreach(addNode)
98    }
99
100   def getNode(nodeId: Int): Option[Node] = this.nodes.par.find(_.id == nodeId)
101   def addNode(node: Node): Unit = this.nodes = this.nodes + node
102   def addNodes(nodes: Set[Node]): Unit = this.nodes = this.nodes ++ nodes
103   def updateNode(node: Node): Unit = {
104     this.synchronized {
105       val currentNode = this.nodes.par.find(_.id == node.id).get
106       currentNode.tree = node.tree
107       currentNode.objects = node.objects
108     }
109   }
110
111   def updateNodes(nodes: Set[Node]): Unit = {
112     val updatedNodeIds = nodes.map(_.id)
113
114     this.nodes =
115       this.nodes
116         .filterNot(n => {
117           updatedNodeIds.contains(n.id)
118         })
119         .++(nodes)
120   }
121
122   def isNodeExist(nodeId: Int): Boolean = this.nodes.exists(_.id == nodeId)
123
124   /** Find all nodes inside GridLocation */
125   def getNodes(g: GridLocation): List[Node] = {
126     this.nodes.par.filter((n: Node) => {
127       (round(floor(n.x / GRID_WIDTH)) == g.x) & (round(floor(n.y / GRID_HEIGHT)) == g.y)
128     }).toList
129   }
130
131   def getNodesFromId(nodeIds: Set[Int]): List[Node] = {
132     this.nodes.par.filter((n: Node) => nodeIds.contains(n.id)).toList
133   }
134
135   /** Find all nodes connected to edges */
136   def getNodes(edges: List[Edge]): List[Node] = {
137     val nodeIds = edges.flatMap((e: Edge) => Set(e.i, e.j))
138
139     this.nodes.par.filter((n: Node) => nodeIds.contains(n.id)).toList
140   }
141
142   def addRawEdges(edges: Set[RawEdge]): Unit = {
143     val es = edges.par.map(rawEdge => {
144       val nodei = getNode(rawEdge.i).get
145
146       rawEdge.lengthMaybe match {
147         case Some(length) =>
148           Edge(rawEdge.id, rawEdge.i, rawEdge.j, length, Set())
149         case None =>
150           val nodej = getNode(rawEdge.j).get

```

```

51         val dx = nodej.x - nodei.x
52         val dy = nodej.y - nodei.y
53         val length = Math.sqrt(dx*dx + dy*dy)
54         Edge(rawEdge.id, rawEdge.i, rawEdge.j, length, Set())
55     }
56 ).toList.toSet
57
58     addEdges(es)
59 }
60
61 // edge
62 def getEdge(edgeId: Int): Option[Edge] = this.edges.par.find(_.id == edgeId)
63
64 def addEdge(edge: Edge): Unit = {
65     this.edges = this.edges + edge
66 }
67
68 def addEdges(edges: Set[Edge]): Unit = {
69     edges.foreach((e: Edge) => {
70         addEdge(e)
71     })
72 }
73
74 def addObjectToEdge(rawObject: RawObject): Unit = {
75     val edgeMaybe = this.edges.par.find(_.id == rawObject.edgeId)
76
77     if (edgeMaybe.isEmpty) {
78         println(rawObject)
79     }
80
81     val edge = edgeMaybe.get
82
83     val newObject = Object(
84         rawObject.id,
85         rawObject.edgeId,
86         100,
87         isImpossible = false,
88         edge.i,
89         createRect(rawObject.points),
90         rawObject.position * edge.length,
91         rawObject.position)
92
93     addObjectToEdge(newObject)
94 }
95
96 def addObjectToEdge(obj: Object): Unit = {
97     val e = this.edges.par.find(_.id == obj.edgeId).get
98     val newEdge = Edge(e.id, e.i, e.j, e.length, e.objects + obj)
99
100    this.edges = this.edges - e + newEdge
101 }
102
103 def removeObjectFromEdge(obj: Object): Unit = {
104     val e = this.edges.par.find(_.id == obj.edgeId).get
105
106     this.edges = this.edges - e + Edge(e.id, e.i, e.j, e.length, e.objects - obj)
107 }
108
109 def removeObjectFromEdge(objectId: Int): Unit = {
110     val o = this.rawObjects.par.find(_.id == objectId).get
111     val e = this.edges.par.find(_.id == o.edgeId).get
112     val deletedObject = e.objects.find(_.id == objectId).get
113
114     this.edges = this.edges - e + Edge(e.id, e.i, e.j, e.length, e.objects - deletedObject)

```

```

115     }
116
117     def getEdges(nodes: List[Node]): List[Edge] = {
118       val nodeIds = nodes.map((n: Node) => n.id)
119
120       this.edges.par.filter((e: Edge) => {
121         nodeIds.contains(e.i) | nodeIds.contains(e.j)
122       }).toList
123     }
124
125     def getGridLocation(node: Node): GridLocation = {
126       val gx = math.floor(node.x / GRID_WIDTH).toInt
127       val gy = math.floor(node.y / GRID_HEIGHT).toInt
128
129       GridLocation(gx, gy)
130     }
131
132     def getGridLocation(x: Double, y: Double): GridLocation = {
133       val gx = math.floor(x / GRID_WIDTH).toInt
134       val gy = math.floor(y / GRID_HEIGHT).toInt
135
136       GridLocation(gx, gy)
137     }
138
139     def getDataGrid(g: GridLocation): EdgesNodes = {
140       val _nodes = getNode(g)
141       val _nodeIds = _nodes.map(_.id)
142       val edges = getEdges(_nodes)
143       val nodeIds = edges
144         .flatMap(e => Set(e.i, e.j))
145         .filterNot(nid => _nodeIds.contains(nid))
146         .toSet
147
148       val nodes = getNodesFromId(nodeIds).toSet
149
150       EdgesNodes(edges.toSet, _nodes.toSet ++ nodes)
151     }
152   }

```

Kode Sumber 1.7 Kode sumber penentuan grid

```

1 package ta.geometry;
2
3 /*
4  * #\$L
5  * Conversant RTree
6  * ~~
7  * Conversantmedia.com © 2016, Conversant, Inc. Conversant® is a trademark of Conversant, Inc.
8  * ~~
9  * Licensed under the Apache License, Version 2.0 (the "License");
10 * you may not use this file except in compliance with the License.
11 * You may obtain a copy of the License at
12 *
13 *      http://www.apache.org/licenses/LICENSE-2.0
14 *
15 * Unless required by applicable law or agreed to in writing, software
16 * distributed under the License is distributed on an "AS IS" BASIS,
17 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
18 * See the License for the specific language governing permissions and
19 * limitations under the License.
20 * #\$%

```

```
21      *
22      * Modifications copyright (C) 2018 <syukronrm>
23      */
24
25  import collection.spatial.HyperPoint;
26  import collection.spatial.HyperRect;
27  import collection.spatial.RTree;
28  import collection.spatial.RectBuilder;
29
30 /**
31  * Created by jcovert on 6/15/15.
32  */
33 public final class Point2d implements HyperPoint {
34     public static final int X = 0;
35     public static final int Y = 1;
36
37     public final double x, y, p;
38     public final int o;
39
40     public Point2d(final double x, final double y, final double p, final int o) {
41         this.x = x;
42         this.y = y;
43         this.p = p;
44         this.o = o;
45     }
46
47     public Point2d(final double x, final double y) {
48         this.x = x;
49         this.y = y;
50         this.p = 0;
51         this.o = 0;
52     }
53
54     @Override
55     public int getNDim() {
56         return 2;
57     }
58
59     @Override
60     public Double getCoord(final int d) {
61         if(d==X) {
62             return x;
63         } else if(d==Y) {
64             return y;
65         } else {
66             throw new IllegalArgumentException("Invalid dimension");
67         }
68     }
69
70     @Override
71     public double distance(final HyperPoint p) {
72         final Point2d p2 = (Point2d)p;
73
74         final double dx = p2.x-x;
75         final double dy = p2.y-y;
76         return Math.sqrt(dx*dx + dy*dy);
77     }
78
79     @Override
80     public double distance(final HyperPoint p, final int d) {
81         final Point2d p2 = (Point2d)p;
82         if(d == X) {
83             return Math.abs(p2.x - x);
84         } else if (d == Y) {
```

```

85         return Math.abs(p2.y - y);
86     } else {
87         throw new IllegalArgumentException("Invalid dimension");
88     }
89 }
90
91 public boolean equals(final Object o) {
92     if(this == o) return true;
93     if(o==null || getClass() != o.getClass()) return false;
94
95     final Point2d p = (Point2d) o;
96     return RTree.isEqual(x, p.x) &&
97            RTree.isEqual(y, p.y) &&
98            RTree.isEqual(this.p, p.p) &&
99            this.o == p.p;
100 }
101
102
103 public int hashCode() {
104     return Double.hashCode(x) ^
105            31*Double.hashCode(y) ^
106            31*31*Double.hashCode(p) ^
107            31*31*Double.hashCode(o);
108 }
109
110 public final static class Builder implements RectBuilder<Point2d> {
111
112     @Override
113     public HyperRect getBBox(final Point2d point) {
114         return new Rect2d(point);
115     }
116
117     @Override
118     public HyperRect getMbr(final HyperPoint p1, final HyperPoint p2) {
119         final Point2d point1 = (Point2d)p1;
120         final Point2d point2 = (Point2d)p2;
121         return new Rect2d(point1, point2);
122     }
123 }
124 }
```

Kode Sumber 1.8 Kode sumber titik dua dimensi

```

1 package ta.geometry;
2
3 /*
4  * # $L
5  * Conversant RTree
6  * ~~
7  * Conversantmedia.com © 2016, Conversant, Inc. Conversant® is a trademark of Conversant, Inc.
8  * ~~
9  * Licensed under the Apache License, Version 2.0 (the "License");
10 * you may not use this file except in compliance with the License.
11 * You may obtain a copy of the License at
12 *
13 *      http://www.apache.org/licenses/LICENSE-2.0
14 *
15 * Unless required by applicable law or agreed to in writing, software
16 * distributed under the License is distributed on an "AS IS" BASIS,
17 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
18 * See the License for the specific language governing permissions and
```

```
19     * limitations under the License.
20     * #L§
21     *
22     * Modifications copyright (C) 2018 <syukronrm>
23     */
24
25 import collection.spatial.HyperPoint;
26 import collection.spatial.HyperRect;
27 import collection.spatial.RectBuilder;
28
29 import java.util.List;
30
31 /**
32  * Created by jcovert on 6/15/15.
33  */
34 public final class Rect2d implements HyperRect {
35     private Point2d min, max;
36
37     public Rect2d(List<Point2d> points) {
38         setPoints(points);
39     }
40
41     public Rect2d(final Point2d p) {
42         min = new Point2d(p.x, p.y);
43         max = new Point2d(p.x, p.y);
44     }
45
46     public Rect2d(final double x1, final double y1, final double x2, final double y2) {
47         min = new Point2d(x1, y1);
48         max = new Point2d(x2, y2);
49     }
50
51     public Rect2d(final Point2d p1, final Point2d p2) {
52         final double minX, minY, maxX, maxY;
53
54         if(p1.x < p2.x) {
55             minX = p1.x;
56             maxX = p2.x;
57         } else {
58             minX = p2.x;
59             maxX = p1.x;
60         }
61
62         if(p1.y < p2.y) {
63             minY = p1.y;
64             maxY = p2.y;
65         } else {
66             minY = p2.y;
67             maxY = p1.y;
68         }
69
70         min = new Point2d(minX, minY);
71         max = new Point2d(maxX, maxY);
72     }
73
74
75     @Override
76     public void setPoints(List points) {
77         double minX, minY, maxX, maxY;
78         minX = Double.MAX_VALUE;
79         minY = Double.MAX_VALUE;
80         maxX = Double.MIN_VALUE;
81         maxY = Double.MIN_VALUE;
82 }
```

```
83     for (Object point : points) {
84         Point2d p = (Point2d) point;
85
86         if (p.x < minX) {
87             minX = p.x;
88         } else if (p.x > maxX) {
89             maxX = p.x;
90         }
91
92         if (p.y < minY) {
93             minY = p.y;
94         } else if (p.y > maxY) {
95             maxY = p.y;
96         }
97     }
98
99     min = new Point2d(minX, minY);
100    max = new Point2d(maxX, maxY);
101 }
102
103 @Override
104 public HyperRect getMbr(final HyperRect r) {
105     final Rect2d r2 = (Rect2d)r;
106     final double minX = Math.min(min.x, r2.min.x);
107     final double minY = Math.min(min.y, r2.min.y);
108     final double maxX = Math.max(max.x, r2.max.x);
109     final double maxY = Math.max(max.y, r2.max.y);
110
111     return new Rect2d(minX, minY, maxX, maxY);
112 }
113
114 @Override
115 public HyperRect getPDR() {
116     final double minX = min.x;
117     final double minY = min.y;
118     final double maxX = Double.MAX_VALUE;
119     final double maxY = Double.MAX_VALUE;
120
121     return new Rect2d(minX, minY, maxX, maxY);
122 }
123
124 @Override
125 public HyperRect getDDR() {
126     final double minX = max.x;
127     final double minY = max.y;
128     final double maxX = Double.MAX_VALUE;
129     final double maxY = Double.MAX_VALUE;
130
131     return new Rect2d(minX, minY, maxX, maxY);
132 }
133
134 @Override
135 public HyperRect getPDD() {
136     final double minX = Double.MIN_VALUE;
137     final double minY = Double.MIN_VALUE;
138     final double maxX = max.x;
139     final double maxY = max.y;
140
141     return new Rect2d(minX, minY, maxX, maxY);
142 }
143
144 @Override
145 public HyperRect getDDD() {
146     final double minX = Double.MIN_VALUE;
```

```
|47      final double minY = Double.MIN_VALUE;
|48      final double maxX = min.x;
|49      final double maxY = min.y;
|50
|51      return new Rect2d(minX, minY, maxX, maxY);
|52  }
|53
|54  @Override
|55  public int getNDim() {
|56      return 2;
|57  }
|58
|59  @Override
|60  public HyperPoint getCentroid() {
|61      final double dx = min.x + (max.x - min.x)/2.0;
|62      final double dy = min.y + (max.y - min.y)/2.0;
|63
|64      return new Point2d(dx, dy);
|65  }
|66
|67  @Override
|68  public HyperPoint getMin() {
|69      return min;
|70  }
|71
|72  @Override
|73  public HyperPoint getMax() {
|74      return max;
|75  }
|76
|77  @Override
|78  public double getRange(final int d) {
|79      if(d == 0) {
|80          return max.x - min.x;
|81      } else if(d == 1) {
|82          return max.y - min.y;
|83      } else {
|84          throw new IllegalArgumentException("Invalid dimension");
|85      }
|86  }
|87
|88  @Override
|89  public boolean contains(final HyperRect r) {
|90      final Rect2d r2 = (Rect2d)r;
|91
|92      return min.x <= r2.min.x &&
|93          max.x >= r2.max.x &&
|94          min.y <= r2.min.y &&
|95          max.y >= r2.max.y;
|96  }
|97
|98  @Override
|99  public boolean intersects(final HyperRect r) {
|100     final Rect2d r2 = (Rect2d)r;
|101
|102     if(min.x > r2.max.x ||
|103         r2.min.x > max.x ||
|104         min.y > r2.max.y ||
|105         r2.min.y > max.y) {
|106         return false;
|107     }
|108
|109     return true;
|110 }
```

```

11
12     @Override
13     public double cost() {
14         final double dx = max.x - min.x;
15         final double dy = max.y - min.y;
16         return Math.abs(dx*dy);
17     }
18
19     @Override
20     public double perimeter() {
21         double p = 0.0;
22         final int nD = this.getNDim();
23         for(int d = 0; d<nD; d++) {
24             p += 2.0 * this.getRange(d);
25         }
26         return p;
27     }
28
29     @Override
30     public boolean equals(Object o) {
31         if (this == o) return true;
32         if (o == null || getClass() != o.getClass()) return false;
33
34         final Rect2d rect2D = (Rect2d) o;
35
36         return min.equals(rect2D.min) &&
37                max.equals(rect2D.max);
38     }
39
40     @Override
41     public int hashCode() {
42         return min.hashCode() ^ 31*max.hashCode();
43     }
44
45     public String toString() {
46         final StringBuilder sb = new StringBuilder();
47         sb.append('(');
48         sb.append(Double.toString(min.x));
49         sb.append(',');
50         sb.append(Double.toString(min.y));
51         sb.append(')');
52         sb.append('u');
53         sb.append('(');
54         sb.append(Double.toString(max.x));
55         sb.append(',');
56         sb.append(Double.toString(max.y));
57         sb.append(')');
58
59         return sb.toString();
60     }
61
62     public final static class Builder implements RectBuilder<Rect2d> {
63
64         @Override
65         public HyperRect getBBox(final Rect2d rect2D) {
66             return rect2D;
67         }
68
69         @Override
70         public HyperRect getMbr(final HyperPoint p1, final HyperPoint p2) {
71             return new Rect2d(p1.getCoord(0), p1.getCoord(1),
72                               p2.getCoord(0), p2.getCoord(1));
73         }
74     }

```

75 }

Kode Sumber 1.9 Kode sumber kotak dua dimensi

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Simulasi</title>
5          <link rel='stylesheet' href='/stylesheets/style.css' />
6      </head>
7      <body>
8          <div>
9              <label for="name">Source Node ID:</label>
10             <input type="number" id="src" name="src">
11         </div>
12         <div>
13             <label for="mail">Destination Node ID:</label>
14             <input type="number" id="dst" name="dst">
15         </div>
16         <div class="button">
17             <button onclick="findSP()">Find SP</button>
18         </div>
19     </body>
20     <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
21     <script src="https://d3js.org/d3.v3.min.js" charset="utf-8"></script>
22     <script src="https://d3js.org/d3-ease.v1.min.js"></script>
23     <script src='/javascripts/main.js'></script>
24 </html>
```

Kode Sumber 1.10 Kode sumber visualisasi HMTL

```

1  let nodes = [
2      [0, 263.35423368818044, -41.66044206649194],
3      [1, 238.60036976480356, 45.9301533546878],
4      [2, 360.9318400625076, 21.176289431310977],
5      [3, 254.29981085411475, 249.67349487786697],
6      [4, 399.01470763693357, 320.1267998905551],
7      [5, 391.39813412204836, 223.01548757576876],
8      [6, 374.2608437135567, 122.09588850353987],
9      [7, 337.6158254583112, -100.68888680685225],
10     [8, 592.7007805965549, -114.48418451744215],
11     [9, 430.9188510156549, -39.75629868777065],
12     [10, 579.6646264266258, 0.6608449271561767],
13     [11, 715.1475438849548, 87.38629950182124],
14     [12, 729.757100047713, -62.241253736093114],
15     [13, 851.1191130234571, 58.82759610948176],
16     [14, 682.6181804926973, 229.43095243282818],
17     [15, 901.739214937918, 224.74437550517135],
18     [16, 787.6016375995532, 329.4186640829],
19     [17, 907.1746963026433, 375.5806910625377],
20     [18, 788.6284402031968, 461.49970397547213],
21     [19, 657.0853235616883, 403.0615594941273],
22     [20, 552.7840272538181, 322.0309432692764]
23 ]
24
25 let edges = [
26     [1, 1, 0],
27     [2, 2, 0],
```

```
28      [3, 2, 11],
29      [4, 3, 11],
30      [5, 4, 31],
31      [6, 5, 31],
32      [7, 5, 41],
33      [8, 6, 21],
34      [9, 6, 51],
35      [10, 7, 0],
36      [11, 8, 7],
37      [12, 9, 7],
38      [13, 9, 8],
39      [14, 10, 6],
40      [15, 10, 9],
41      [16, 11, 10],
42      [17, 12, 8],
43      [18, 12, 10],
44      [19, 13, 11],
45      [20, 13, 12],
46      [21, 14, 11],
47      [22, 15, 13],
48      [23, 16, 14],
49      [24, 16, 15],
50      [25, 17, 15],
51      [26, 17, 16],
52      [27, 18, 16],
53      [28, 18, 17],
54      [29, 19, 16],
55      [30, 19, 18],
56      [31, 20, 4],
57      [32, 20, 14],
58      [33, 20, 19]
59  ]
60
61  let svgWidth = 1100;
62  let svgHeight = 1100;
63
64  let viewWidth = 800;
65  let viewHeight = 800;
66
67  let domainX = [100, 1000]
68  let domainY = [-200, 500]
69
70  var xScale = d3.scale.linear()
71          .domain(domainX)
72          .range([0, viewWidth]);
73
74  var xScaleReverse = d3.scale.linear()
75          .domain([0, viewWidth])
76          .range(domainX);
77
78  var yScale = d3.scale.linear()
79          .domain(domainY)
80          .range([viewHeight, 0]);
81
82  var yScaleReverse = d3.scale.linear()
83          .domain([viewHeight, 0])
84          .range(domainY);
85
86  let svg = d3.select('body')
87          .append('svg')
88          .attr('width', svgWidth)
89          .attr('height', svgHeight);
90
91  let svgGridX = svg.selectAll('gridline')
```

```

92         .data([0, 5, 10, 15, 20])
93         .enter()
94         .append('line');
95
96     svgGridX.attr('x1', function(d) { return xScale(d); })
97         .attr('y1', function(d) { return yScale(0); })
98         .attr('x2', function(d) { return xScale(d); })
99         .attr('y2', function(d) { return yScale(20); })
100        .attr('class', 'gridline');
101
102    let svgGridY = svg.selectAll('gridline')
103        .data([0 , 5, 10, 15, 20])
104        .enter()
105        .append('line');
106
107    svgGridY.attr('x1', function(d) { return xScale(0); })
108        .attr('y1', function(d) { return yScale(d); })
109        .attr('x2', function(d) { return xScale(20); })
110        .attr('y2', function(d) { return yScale(d); })
111        .attr('class', 'gridline')
112
113    let edgesEnter = svg.selectAll('edges')
114        .data(edges)
115        .enter();
116
117    function findNode(id) {
118        for(let i = 0; i < nodes.length; i++) {
119            if (nodes[i][0] == id) return nodes[i];
120        }
121
122        return null;
123    }
124
125    edgesEnter.append('line')
126        .attr('x1', function(d) {
127            return xScale(findNode(d[1])[1]); })
128        .attr('y1', function(d) {
129            return yScale(findNode(d[1])[2]); })
130        .attr('x2', function(d) {
131            return xScale(findNode(d[2])[1]); })
132        .attr('y2', function(d) {
133            return yScale(findNode(d[2])[2]); })
134        .attr('class', 'edges')
135
136    edgesEnter.append('text')
137        .attr('x', function(d) {
138            x = findMiddleEdge(findNode(d[1]),
139                findNode(d[2]))[0]
140            return xScale(x);
141        })
142        .attr('y', function(d){
143            y = findMiddleEdge(findNode(d[1]),
144                findNode(d[2]))[1]
145            return yScale(y);
146        })
147        .attr('class', 'edge-label')
148        .text(function(a, i) { return "e" + a[0]; });
149
150    function findLocation(edgeId, pos){
151        let edge;
152        for (let i = 0; i < edges.length; i++) {
153            if (edges[i][0] == edgeId) {
154                edge = edges[i]
155            }
156        }
157    }

```

```

56      }
57
58  let node1 = findNode(edge[1])
59  let node2 = findNode(edge[2])
60
61  let x = (node2[1] - node1[1]) * pos + node1[1]
62  let y = (node2[2] - node1[2]) * pos + node1[2]
63
64  return [x, y]
65 }
66
67 let nodesEnter = svg.selectAll('nodes')
68     .data(nodes)
69     .enter()
70
71 nodesEnter.append('rect')
72     .attr('x', function(d){
73         return xScale(d[1]) - 7.5;
74     })
75     .attr('y', function(d){
76         return yScale(d[2]) - 7.5;
77     })
78     .attr('class', 'nodes')
79     .attr('width', 15)
80     .attr('height', 15);
81
82 nodesEnter.append('text').attr('x', function(d){
83     return xScale(d[1]);
84 })
85     .attr('y', function(d){
86         return yScale(d[2]) + 3; })
87     .attr('class', 'node-label')
88     .attr('text-anchor', 'middle')
89     .text(function(a, i) { return "n" + a[0]; });
90
91 function findMiddleEdge(node1, node2) {
92     x = (node2[1] + node1[1]) / 2;
93     y = (node2[2] + node1[2]) / 2;
94
95     return [x, y];
96 }
97
98 function getNode(i) {
99     return nodes[i].map(x => x * 10);
100 }
101
102 function getEdge(edges, edgeId) {
103     for (let i = 0; i < edges.length; i++) {
104         if (edges[i][0] == edgeId)
105             return edges[i]
106     }
107
108     return null
109 }
110
111 function getNode(nodes, nodeId) {
112     for (let i = 0; i < nodes.length; i++) {
113         if (nodes[i][0] == nodeId)
114             return nodes[i]
115     }
116
117     return null
118 }
119

```

```

20  function getEdgeLength(edgeId) {
21    let edge = getEdge(edges, edgeId)
22    let nodeSId = edge[1]
23    let nodeEId = edge[2]
24    let nodes = getNode(nodes, nodeSId)
25    let nodeE = getNode(nodes, nodeEId)
26
27    let startX = nodeS[1]
28    let startY = nodeS[2]
29
30    let selisihX = nodeE[1] - nodeS[1]
31    let selisihY = nodeE[2] - nodeS[2]
32
33    return Math.sqrt(selisihX*selisihX + selisihY*selisihY)
34  }
35
36  function findTPCoordinatesByNodeId(nodeSId, nodeEId, TP) {
37    let nodeS = getNode(nodes, nodeSId)
38    let nodeE = getNode(nodes, nodeEId)
39
40    let startX = nodeS[1]
41    let startY = nodeS[2]
42
43    let selisihX = nodeE[1] - nodeS[1]
44    let selisihY = nodeE[2] - nodeS[2]
45
46    let length =
47      Math.sqrt(selisihX*selisihX + selisihY*selisihY)
48
49    return TP.map(function(tp) {
50      let ratio = tp / length
51
52      let posX = startX + selisihX * ratio
53      let x = xScale(posX) - 2
54
55      let posY = startY + selisihY * ratio
56      let y = yScale(posY) - 2
57
58      return {
59        x: x,
60        y: y
61      }
62    })
63  }
64
65  function findTPCoordinatesByEdgeId(edgeId, TP) {
66    let edge = getEdge(edges, edgeId)
67    let nodeSId = edge[1]
68    let nodeEId = edge[2]
69
70    return findTPCoordinatesByNodeId(nodeSId, nodeEId, TP)
71  }
72
73  function addTurningPointMarker(edgeId, TP) {
74    let tps = findTPCoordinatesByEdgeId(edgeId, TP)
75
76    let turningPointEnter = svg.selectAll('turning-point')
77      .data(tps)
78      .enter()
79
80    turningPointEnter.append('rect')
81      .attr('x', d => d.x)
82      .attr('y', d => d.y)
83      .attr('width', 5)

```

```

284     .attr('height', 5)
285     .attr('fill', 'blue')
286     .attr('class', 'turning-point turning-point-' + edgeId)
287   }
288
289   function insertToGrid(SP) {
290     let edgeId = SP.edge
291     let SPs = SP.SP
292     let TP = SP.TP
293
294     addTurningPointMarker(edgeId, TP)
295   }
296
297   function removeEdgeData(edgeId) {
298     svg.selectAll('.turning-point-' + edgeId).remove()
299   }
300
301   function getSP(listOfSP, edgeId) {
302     let SP = listOfSP.filter(sp => sp.edge == edgeId)
303
304     if (SP) {
305       return SP
306     } else {
307       null
308     }
309   }
310
311   function getNodeLength(nodeSId, nodeEId) {
312     let nodeS = getNode(nodes, nodeSId)
313     let nodeE = getNode(nodes, nodeEId)
314
315     let startX = nodeS[1]
316     let startY = nodeS[2]
317
318     let selisihX = nodeE[1] - nodeS[1]
319     let selisihY = nodeE[2] - nodeS[2]
320
321     let length =
322       Math.sqrt(selisihX*selisihX + selisihY*selisihY)
323
324     return length
325   }
326
327   function getEdgeByNodeId(nodeSId, nodeEId) {
328     return edges.find(e => {
329       return (
330         (e[1] == nodeSId && e[2] == nodeEId) ||
331         (e[2] == nodeSId && e[1] == nodeEId)
332       );
333     });
334   }
335
336   svg.append("circle")
337     .attr("cx", 10)
338     .attr("cy", 10)
339     .attr("r", 5)
340     .attr('id', 'query');
341
342   function traverseQueryPoint(selector, nodeIds) {
343     let node = getNode(nodes, nodeIds[0])
344
345     let svqq = svg.select(selector)
346       .attr('cx', xScale(node[1]))
347       .attr('cy', yScale(node[2]))

```

```

$48
$49 let SPs = []
$50 let TPs = []
$51 let reverses = []
$52
$53 for (let i = 1; i < nodeIds.length; i++) {
$54   let nodeId = nodeIds[i]
$55   let nodeIdPrev = nodeIds[i-1]
$56   let node = getNode(nodes, nodeId)
$57   let nodePrev = getNode(nodes, nodeIdPrev)
$58   let length = getNodeLength(nodeIdPrev, nodeId)
$59
$60   let nodePrevID = nodePrev[0]
$61   let nodePrevX = nodePrev[1]
$62   let nodePrevY = nodePrev[2]
$63
$64   let nodeNextID = node[0]
$65   let nodeNextX = node[1]
$66   let nodeNextY = node[2]
$67
$68   let edge = getEdgeByNodeId(nodeIdPrev, nodeId)
$69   let edgeId = edge[0]
$70
$71   let tick = 0
$72   let currentIndex = i
$73
$74   svgg = svgg.transition()
$75     .ease(d3.easeLinear)
$76     .duration(length * 50)
$77     .attr('cx', xScale(node[1]))
$78     .attr('cy', yScale(node[2]))
$79     .tween("sides-effects", function() {
$80       return function() {
$81         if (tick % 50 == 0) {
$82           let realX = d3.select(this).attr('cx')
$83           let realY = d3.select(this).attr('cy')
$84
$85           let currentX = xScaleReverse(realX)
$86           let currentY = yScaleReverse(realY)
$87
$88           let diffX = Math.abs(currentX) - Math.abs(nodePrevX)
$89           let diffY = Math.abs(currentY) - Math.abs(nodePrevY)
$90
$91           let deltaX = Math.abs(nodeNextX) - Math.abs(nodePrevX)
$92           let deltaY = Math.abs(nodeNextY) - Math.abs(nodePrevY)
$93
$94           let diffXY = Math.sqrt(diffX*diffX + diffY*diffY)
$95           let deltaXY = Math.sqrt(deltaX*deltaX + deltaY*deltaY)
$96
$97           let currentPosition = diffXY / deltaXY
$98
$99           let position = 0
$100          if (nodePrevID == edge[1]) {
$101            position = currentPosition
$102          } else {
$103            position = 1 - currentPosition
$104          }
$105
$106          let realLength = length * position
$107
$108          let maybeSP = getSP(listOfSP, edgeId)
$109          let isSPFound = false
$110          if (maybeSP.length > 0) {

```

```

412         let SP = maybeSP[0].SP
413
414         for (let i = 0; i < SP.length; i++) {
415             let dStart = SP[i].s
416             let dEnd = SP[i].e
417
418             if (dStart < realLength && realLength < dEnd) {
419                 isSPFound = true
420                 console.log(SP[i].sp)
421                 break
422             }
423         }
424
425         if (!isSPFound) {
426             console.log([])
427         } else {
428             console.log([])
429         }
430     }
431 }
432
433     tick++
434 }
435 })
436 }
437
438 return svgq
439 }
440
441 function getData() {
442     $.ajax({
443         url: 'http://localhost:8080/data',
444     })
445     .done(function(result) {
446         let data = result.data
447         listOfSP = result.data
448
449         for (var i = 0 ; i < data.length; i++) {
450             // console.log("update edge " + data[i].edge)
451             removeEdgeData(data[i].edge)
452             insertToGrid(data[i])
453         }
454     })
455     .fail(function() {
456         console.log('failed')
457     })
458
459     setTimeout(function(){
460         getData()
461     }, 500);
462 }
463
464 function getObjects() {
465     $.ajax({
466         url: 'http://localhost:8080/objects',
467     })
468     .done(function(result) {
469         let objects = result.data
470
471         svg.selectAll('.objects')
472             .remove()
473
474         svg.selectAll('.object-label')
475             .remove()

```

```

$76
$77     let objectsEnter = svg.selectAll('objects')
$78         .data(objects)
$79         .enter()
$80
$81     objectsEnter
$82         .append("circle")
$83             .attr('cx', function(d){ return xScale(findLocation(d[1], d[2])[0]); })
$84             .attr('cy', function(d){ return yScale(findLocation(d[1], d[2])[1]); })
$85             .attr('r', 8)
$86             .attr("class", "objects")
$87
$88     objectsEnter.append("text")
$89         .attr('dx', function(d){ return xScale(findLocation(d[1], d[2])[0]); })
$90         .attr('dy', function(d){ return yScale(findLocation(d[1], d[2])[1]) + 3; })
$91         .attr('text-anchor', 'middle')
$92         .attr('class', 'object-label')
$93         .append("tspan")
$94             .text(function(d) { return "O"; })
$95         .append("tspan")
$96             .attr("baseline-shift", "sub")
$97             .text(function(d) { return d[0]; });
$98     })
$99     .fail(function() {
$100         console.log('failed')
$101     })
$102
$103     setTimeout(function() {
$104         getObjects()
$105     }, 500);
$106 }
$107
$108     getObjects()
$109
$110     getData()
$111
$112     function findSP() {
$113         let srcID = document.getElementById('src').value
$114         let dstID = document.getElementById('dst').value
$115
$116         let data = {
$117             src: srcID,
$118             dst: dstID
$119         }
$120
$121         $.ajax({
$122             url: 'http://localhost:8080/getPath?src=' + srcID + '&dst=' + dstID,
$123         })
$124         .done(function(path) {
$125             traverseQueryPoint('#query', path)
$126         })
$127         .fail(function() {
$128             console.log('failed')
$129         })
$130     }

```

Kode Sumber 1.11 Kode sumber visualisasi utama dengan JavaScript

```

1   body {
2     padding: 50px;
3     font: 14px "Lucida Grande", Helvetica, sans-serif;

```

```
4  }
5  a {
6    color: #00B7FF;
7  }
8
9  .gridline {
10   stroke: green;
11   stroke-width: 1;
12 }
13 }
14
15 .nodes {
16   fill: gray;
17 }
18
19 .node-label {
20   display: hidden;
21   font: bold 10px Arial;
22 }
23
24 .edges {
25   stroke: black;
26   stroke-width: 2;
27 }
28
29 .objects {
30   fill: aqua;
31   stroke: black;
32 }
33
34 .object-label {
35   font: bold 10px Arial;
36 }
37
38 .edge-label {
39   font: Bold 10px Arial;
40   fill: brown;
41 }
42
43 #query {
44   fill: orange
45 }
46
47 .hidden {
48   visibility: hidden;
49 }
```

Kode Sumber 1.12 Kode sumber visualisasi dengan CSS

BIODATA PENULIS



Hafara Firdausi, lahir di Surabaya pada tanggal 19 September 1998. Penulis merupakan anak pertama dari 3 bersaudara. Penulis telah menempuh pendidikan formal di SD Negeri Gelam 1 Candi (2004-2010), SMP Negeri 1 Sidoarjo (2010-2013), dan SMA Negeri 1 Sidoarjo (2013-2015). Kemudian, penulis melanjutkan studi kuliah program sarjana di Departemen Informatika, Institut Teknologi Sepuluh Nopember Surabaya.

Selama menempuh pendidikan di Informatika ITS, penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Jaringan Komputer (2017-2018) dan Sistem Operasi (2019), serta menjadi Administrator Laboratorium Arsitektur dan Jaringan Komputer (AJK). Selain itu, penulis juga aktif dalam kegiatan organisasi dan kepanitiaan, antara lain menjadi Badan Pengurus Harian I 3D (Desain, Dekorasi, dan Dokumentasi) Schematics ITS 2017, staff dan staff ahli Departemen Media Informasi Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS, dan Panitia Gemastik ke-11 Bidang Keamanan Jaringan dan Sistem Informasi (KJSI). Penulis dapat dihubungi melalui surel di hafarafirdausi@gmail.com.