ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN SKYLINE QUERY PADA UNCERTAIN DATA STREAMING OLEH TITIK BERGERAK DAN OBJEK TIDAK BERGERAK PADA JARINGAN JALAN RAYA

SYUKRON RIFA'IL MUTTAQI
NRP 5114 100 093

Dosen Pembimbing 1
Royyana M Ijtihadie, S.Kom, M.Kom, Ph.D.

Dosen Pembimbing 2
Bagus Jati Santoso, S.Kom., Ph.D.

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

Halaman ini sengaja dikosongkan

TUGAS AKHIR - KI141502

DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN SKYLINE QUERY PADA UNCERTAIN DATA STREAMING OLEH TITIK BERGERAK DAN OBJEK TIDAK BERGERAK PADA JARINGAN JALAN RAYA

SYUKRON RIFA'IL MUTTAQI
NRP 5114 100 093

Dosen Pembimbing 1
Royyana M Ijtihadie, S.Kom, M.Kom, Ph.D.

Dosen Pembimbing 2
Bagus Jati Santoso, S.Kom., Ph.D.

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

Halaman ini sengaja dikosongkan

UNDERGRADUATE THESES - KI141502

DESIGN AND IMPLEMENTATION OF SKYLINE QUERY PROCESSING ON UNCERTAIN DATA STREAMING BY MOVING QUERY POINTS AND STATIC OBJECTS ON ROAD NETWORK

SYUKRON RIFA'IL MUTTAQI
NRP 5114 100 093

Supervisor 1
Royyana M Ijtihadie, S.Kom, M.Kom, Ph.D.

Supervisor 2
Bagus Jati Santoso, S.Kom., Ph.D.

DEPARTMENT OF INFORMATICS
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN SKYLINE QUERY PADA UNCERTAIN DATA STREAMING OLEH TITIK BERGERAK DAN OBJEK TIDAK BERGERAK PADA JARINGAN JALAN RAYA

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:
Syukron Rifa'il Muttaqi
NRP: 5114 100 093

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

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

SURABAYA
JUNI 2018

Halaman ini sengaja dikosongkan

DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN SKYLINE QUERY PADA UNCERTAIN DATA STREAMING OLEH TITIK BERGERAK DAN OBJEK TIDAK BERGERAK PADA JARINGAN JALAN RAYA

Nama	: SYUKRON RIFA'IL MUTTAQI
NRP	: 5114 100 093
Departemen	: Informatika FTIK-ITS
Pembimbing I	: Royyana M Ijtihadie, S.Kom, M.Kom, Ph.D.
Pembimbing II	: Bagus Jati Santoso, S.Kom., Ph.D.

Abstrak

Penelitian ini bertujuan untuk memaparkan desain dan implementasi aplikasi pengolahan skyline query pada uncertain data streaming oleh titik bergerak dan objek tidak bergerak pada jaringan jalan raya. Pada kasus tertentu, pengguna teknologi membutuhkan pengambilan dan pemrosesan data terbaik dengan cepat dan tepat. Diantara permasalahan mengenai pemrosesan data yaitu pencarian objek yang paling unggul pada data spasial. Jaringan jalan raya road network adalah salah satu bentuk data yang bersifat spasial. Pencarian data pada jaringan jalan raya bersifat relatif terhadap titik query tertentu. Jika titik berpindah, maka perlu pemrosesan ulang data yang unggul sesuai titik terakhir. Metode ini kurang efisien karena biaya komputasi sangat tergantung pada titik query. Jika titik query selalu berpindah, maka dibutuhkan pemrosesan tersendiri yang bersifat kontinu sehingga tidak diperlukan banyak komputasi ketika titik query mengalami perpindahan.

Penggunaan skyline query pada kasus demikian menjadi kompleks ketika bersinggungan dengan uncertain data streaming dan jaringan jalan raya. Tugas akhir ini memaparkan algoritma CSd_{ε}, yaitu metode pencarian data yang dominan dengan menggunakan

struktur data Grid dan R-Tree. Hasil yang didapatkan berupa jalan beserta titik-titik skyline yang terdapat pada jalan tersebut. Data dapat masuk dan kadaluarsa/expire. Data streaming yang masuk/kadaluarsa pada sistem diproses satu per satu dan hasil dari pemrosesan/skyline points didapatkan pada setiap data yang masuk/kadaluarsa secara langsung. Dengan demikian, titik query yang banyak tidak mengurangi performa dari pencarian data. Pada proses pengujian, waktu komputasi algoritma CSd_{ε} lebih cepat 600 kali lebih cepat dan penggunaan memori 1500 lebih cepat kali lebih hemat daripada algoritma naive.

Kata Kunci: ***skyline query*, *uncertain data streaming*, *jaringan jalan raya***

DESIGN AND IMPLEMENTATION OF SKYLINE QUERY PROCESSING ON UNCERTAIN DATA STREAMING BY MOVING QUERY POINTS AND STATIC OBJECTS ON ROAD NETWORK

Name	: SYUKRON RIFA'IL MUTTAQI
NRP	: 5114 100 093
Major	: Department of Informatics Faculty of ICT-ITS
Supervisor I	: Royyana M Ijtihadie, S.Kom, M.Kom, Ph.D.
Supervisor II	: Bagus Jati Santoso, S.Kom., Ph.D.

Abstract

This study aims to describe the design and implementation of skyline query processing applications on uncertain data streaming by moving points and immovable objects on the road network. In some cases, users need the best and fastest data retrieval and processing. Among the problems of data processing is the search for the most superior objects in spatial data. Highway network road network is one form of data that is spatial. The search for data on a highway network is relative to a particular query point. If the query point moves, it will need to reprocess the data. This method is less efficient because the cost of computing is highly dependent on the query point. If the query point always moves, it requires continuous processing that is not required for much computation when the query point changes.

The use of skyline queries in such cases becomes complex when in contact with uncertain streaming data and highway networks. This final project describes algorithm CSd_e, the dominant data search method using Grid and R-Tree data structures. The results obtained in the form of roads along with the skyline points contained on the road. Data can enter and expire / expire. Incoming / expired stream data streaming on the system is processed one at a time and the result of skyline processing is obtained on each incoming

/ outdated data directly. Thus, many query points do not decrease the performance of data processing. In the testing process, CSd_{ε} algorithm computation time is 600 times faster and memory usage is 1500 times more efficient than the naive algorithm.

Keywords: ***skyline query*****,** ***uncertain data streaming*****,** ***road network***

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah Subhanahu wa Ta'ala, *rabb* semesta alam, atas pertolongan, taufiq, dan petunjuk-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

DESAIN DAN IMPLEMENTASI APLIKASI PENGOLAHAN SKYLINE QUERY PADA UNCERTAIN DATA STREAMING OLEH TITIK BERGERAK DAN OBJEK TIDAK BERGERAK PADA JARINGAN JALAN RAYA.

Banyak pengalaman berharga telah penulis dapatkan selama proses penulisan tugas akhir ini. Tentunya hal tersebut tidak lepas dari dukungan berbagai pihak. Penulis ucapkan terima kasih banyak kepada:

1. Ibuk, Bapak, Fariq, dan segenap keluarga yang telah mendukung dan memotivasi seumur hidup penulis.
2. Bapak Bagus Jati Santoso, S.Kom., Ph.D dan Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D sebagai pembimbing selama persiapan hingga selesainya penulisan buku ini.
3. Bapak Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Jurusan Teknik Informatika ITS pada masa pengerjaan Tugas Akhir, Bapak Radityo Anggoro, S.Kom., M.Sc., selaku koordinator TA, dan segenap dosen Informatika yang telah memberikan ilmu dan pengalamannya.
4. Teman-teman Lab AJK yang telah menemani dan membantu penulis selama mengerjakan tugas akhir di Lab: Oing, Fatih, Ambon, Thoni, Vivi, Bebet, Nahda, Fuad, Awan, Penyok,

Daus, Satria, Didin, Hana, Raldo, Khawari, dan Aguel.

5. Teman seperjuangan dan seperbimbingan: Ovan, Wira, Sekbay, Dwika, Tosca, dan Cahya.
6. Dan teman-teman yang telah membantu dan menemani yang tidak bisa saya sebutkan satu per satu namanya.

Surabaya, Januari 2018

Syukron Rifa'il Muttaqi

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

DAFTAR KODE SUMBER

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

Cepatnya pertumbuhan dan perkembangan teknologi memicu banyaknya data yang diproduksi. Banyaknya data yang berpindah mendorong proses pengolahan data menjadi lebih cepat. Bertambahnya pengguna teknologi dan internet juga mendorong frekuensi pengambilan data agar menjadi lebih efektif dan efisien.

Pada kasus tertentu, pengguna teknologi membutuhkan pengambilan dan pemrosesan data terbaik dengan cepat dan tepat. Diantara permasalahan mengenai pemrosesan data yaitu pencarian objek yang paling unggul pada data spasial. Jaringan jalan raya *road network* adalah salah satu bentuk data yang bersifat spasial. Pencarian data pada jaringan jalan raya bersifat relatif terhadap titik *query* tertentu. Jika titik berpindah, maka perlu pemrosesan ulang data yang unggul sesuai titik terakhir. Metode ini kurang efisien karena biaya komputasi sangat tergantung pada titik *query*. Jika titik *query* selalu berpindah, maka dibutuhkan pemrosesan tersendiri yang bersifat kontinu sehingga tidak diperlukan banyak komputasi ketika titik *query* mengalami perpindahan.

Banyaknya teknologi yang digunakan sekarang membuat *uncertain data* semakin banyak. *Uncertain data* adalah dan tersebar

pada interval tertentu. Beberapa contoh dari *uncertain data* yaitu [5]:

1. Banyak instrumen pengukuran yang memiliki tingkat ketelitian yang kurang baik.
2. Pada kasus ini, tingkat ketidakpastian dapat didapat dari kesalahan pada instrumen.
3. Banyak perangkat keras yang menghasilkan data yang tidak pasti, seperti sensor.

Query data terunggul dari *uncertain data* memerlukan metode tersendiri untuk menyelesaikannya.

Diantara algoritme pencarian data paling unggul adalah metode *skyline*, termasuk pada jaringan jalan raya. Metode ini dapat diterapkan pada jaringan jalan raya dengan menggunakan *uncertain data*. Struktur data khusus diperlukan agar proses pencarian data/*query* dapat dilakukan secara *real time*.

1.2 Rumusan Masalah

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

1. Bagaimana struktur data dan algoritma untuk desain dan implementasi aplikasi pengolahan *skyline query* pada *uncertain data streaming* oleh titik bergerak dan objek tidak bergerak pada jaringan jalan raya?
2. Bagaimana cara mengetahui biaya komputasi dan penyimpanan pada desain dan implementasi aplikasi pengolahan *skyline query* pada *uncertain data streaming* oleh titik bergerak dan objek tidak bergerak pada jaringan jalan raya?

1.3 Batasan Masalah

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

1. Implementasi algoritma dan pemrosesan menggunakan bahasa pemrograman Scala.
2. Data set yang digunakan adalah jaringan jalan raya asli dan sintetis.
3. *Uncertain data* menggunakan jenis *tuple-level uncertainty* dan dimodelkan dalam *probabilistic model*.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mendesain struktur data dan algoritma untuk desain dan implementasi aplikasi pengolahan *skyline query* pada *uncertain data streaming* oleh titik bergerak dan objek tidak bergerak pada jaringan jalan raya.
2. Menemukan metode untuk mengetahui biaya komputasi dan penyimpanan pada desain dan implementasi aplikasi pengolahan *skyline query* pada *uncertain data streaming* oleh titik bergerak dan objek tidak bergerak pada jaringan jalan raya.

1.5 Manfaat

Manfaat Tugas Akhir ini adalah untuk mengetahui struktur data dan algoritma yang tepat untuk pengolahan *skyline query* pada *uncertain data streaming* oleh titik bergerak pada jaringan jalan raya secara optimal dan efisien.

Apabila dilihat dari sisi praktis, algoritme ini dapat digunakan untuk berbagai hal. Sebagai contoh, sebuah perusahaan periklanan ingin menampilkan iklan dari suatu *point of interest* yang

disesuaikan dengan lokasi target pengguna. Dengan algoritme ini, sistem dapat memberikan *point of interest* yang relevan dengan jarak target pengguna serta *point of interest* yang paling dominan.

BAB II
DASAR TEORI

2.1 Pengantar

Bab ini menjelaskan mengenai dasar dari metode yang diusulkan pada Bab III. Berikut disajikan tabel mengenai simbol yang digunakan

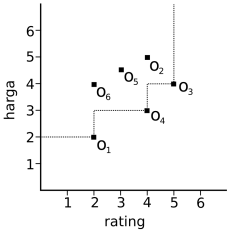
Tabel 2.1 Simbol dan deskripsi pada bab ini

Simbol	Deskripsi
d	Dimensi data
o	Objek
$o \prec o'$	Tuple o mendominasi o'
x	*Uncertain data tuple*
d	Dimensi data
SP	*Skyline Point*, himpunan objek yang menjadi *skyline*

2.2 Skyline

Pada himpunan titik pada multidimensi d, *skyline* dari data tersebut adalah titik yang tidak didominasi oleh titik lain. Titik o mendominasi titik o', dinotasikan sebagai $o \prec o'$, apabila setiap nilai dari atribut pada o lebih baik atau sama dengan yang terdapat pada o' dan setidaknya terdapat satu atribut o yang lebih baik daripada yang terdapat pada o'. Titik titik yang mendominasi tersebut disebut dengan *skyline points* (SP).

Sebagai contoh, seseorang ingin mencari hotel terbaik dari sekumpulan hotel. Sebuah hotel dikatakan lebih baik apabila memiliki rating yang lebih tinggi dan/atau harga yang lebih rendah.



hotel	rating	harga
o_1	2	2
o_2	4	5
o_3	5	4
o_4	4	3
o_5	3	4,5
o_6	2	4

Gambar 2.1 Contoh *skyline* sederhana

SP pada Gambar 2.1 adalah hotel yang memiliki ranking yang tinggi dan harga yang rendah. Objek yang menjadi bagian dari *SP* dari adalah hotel o_1, o_3 dan o_4. Ketiga objek tersebut tidak didominasi oleh objek yang lain. Hotel o_6 didominasi oleh o_4 pada atribut rating dan harga, didominasi oleh o_1 pada atribut harga, dan didominasi oleh o_3 pada atribut rating. Demikian juga untuk objek o_2 dan o_6, masing-masing didominasi oleh objek lain. Dengan demikian, hanya o_1, o_3, dan o_4 yang tidak didominasi objek lain[1].

2.3 *Skyline Query* pada Jaringan Jalan Raya

Skyline query tradisional seperti pada Gambar 2.1 hanya menggunakan atribut statis sebagai penentuan *SP*, yaitu atribut ranking dan hotel. Jika *skyline query* ditempatkan pada jaringan jalan raya seperti pada Gambar 2.3, maka perlu menambahkan jarak sebagai salah satu atribut dalam menentukan *SP*.

Pada Gambar 2.1, hotel yang menjadi *SP* adalah o_1, o_2 dan o_3. Ketika hotel o_1 hingga o_5 dimasukkan dalam jaringan jalan raya seperti pada Gambar 2.2, o_5 tidak lagi didominasi oleh o_4 sehingga o_5 bergabung menjadi bagian dari *SP*. Hal tersebut dikarenakana jarak antara titik *query* q dengan o_5 tidak didominasi oleh o_1, o_2 maupun o_3. Atribut jarak ini akan berubah-ubah sesuai dengan jarak titik *query* q dengan masing-masing objek. Oleh karena itu, atribut

jarak disebut sebagai atribut dinamis [1].

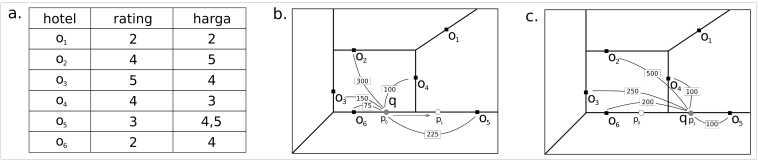
Skyline *query* yang hanya menggunakan atribut statis dapat dicari *SP*-nya dengan sekali proses *query*, atau disebut dengan *snapshot skyline query*. Pada kasus jaringan jalan raya, titik *query* dapat bergerak dengan leluasa. Metode naif yang paling mudah diimplementasikan yaitu melakukan proses *query* ketika titik *query* bergerak. Namun hal ini tentunya sangat tidak efektif mengingat banyaknya komputasi yang dilakukan apabila terdapat banyak titik *query* dan titik *query* tersebut sering bergerak.

Pada kasus tertentu objek *SP* memiliki jarak yang sangat jauh dari titik *query*. Objek tersebut seringkali tidak diinginkan karena kita mencari titik *query* yang dekat karena untuk mencapai objek yang jauh membutuhkan biaya lebih besar dalam hal transportasi.

Y.-K. Huang et al. telah mengusulkan metode untuk menyelesaikan permasalahan ini dengan metode *continuous* d_ε*-skyline query* ($Cd_\varepsilon - SQ$) [1]. Metode ini menggunakan jarak d_ε sebagai batas maksimal jarak antara titik *query* dengan objek.

$Cd_\varepsilon - SQ$ didefinisikan sebagai: Diketahui jalan P_q sebagai jalan tempat titik *query* q bergerak diatasnya, himpunan objek S_o, dan jarak d_ε. Kueri $Cd_\varepsilon - SQ$ menghasilkan himpunan skyline point, SP_p pada setiap titik p pada P_q, yang memiliki jarak antara $o \in SP_p$ ke titik p kurang dari sama dengan d_ε. SP yang memenuhi $Cd_\varepsilon - SQ$ dinotasikan sebagai $d_\varepsilon - SP$.

Gambar 2.2 mengilustrasikan pemrosesan *skyline* pada jaringan jalan raya. Gambar 2.3(a) menampilkan atribut statis dari keempat objek. Contoh tersebut mensimulasikan pencarian $d_\varepsilon - SP$ pada titik *query q* yang bergerak dari titik p_1 ke p_2. Ketika *q* berada pada koordinat p_1(lihat Gambar 2.3(b)), anggota dari $d_\varepsilon - SP$ hanyalah o_2. o_4 tidak dapat menjadi $d_\varepsilon - SP$ karena didominasi oleh



hotel	rating	harga
o_1	2	2
o_2	4	5
o_3	5	4
o_4	4	3
o_5	3	4,5
o_6	2	4

Gambar 2.2 Contoh *skyline* pada jaringan jalan raya

o_2 dalam hal atribut statis(harga dan rating) maupun dinamis(jarak). o_1 dan o_3 tidak tergabung dalam $d_\varepsilon - SP$ karena memiliki jarak lebih dari d_ε. Selanjutnya, seperti yang terlihat pada Gambar 2.3(c), titik *query* q bergerak ke kiri sampai tepat di titik p_2 sehingga jarak antara titik *query* q dengan o_2 dan o_4 menjadi sama. Ketika titik *query* q bergerak ke kiri, o_4 menjadi lebih dekat kepada q dibandingkan o_2. Dengan demikian, o_2 tergabung menjadi $d_\varepsilon - SP$ karena jarak o_2 tidak lagi didominasi oleh o_4. Selanjutnya $d_\varepsilon - SP$-nya adalah o_2 dan o_4.

Dalam menentukan $d_\varepsilon - SP$, yang pertama dilakukan adalah menghitung jarak terdekat antar dua objek. Penghitungan tersebut dapat dilakukan dengan algoritme Djikstra atau A*. Perlu digarisbawahi, penghitungan jarak terdekat membutuhkan sumber daya yang besar apabila terdapat banyak jalan(*edge*) dan persimpangan(*node*). Diantara hal yang menjadi tantangan adalah titik *query* seringkali berpindah-pindah sehingga diperlukan metode khusus agar komputasi tidak dilakukan berulang kali dan menggunakan sumber daya yang banyak [1].

2.4 *Uncertain Data*

Pemrosesan *uncertain data* mendapat banyak perhatian pada beberapa tahun terakhir. *Uncertain data* dapat ditemukan pada berbagai bidang, seperti jaringan sensor, jaringan RFID, sistem pelacakan lokasi menggunakan GPS, dan sosial media [4]. Beberapa perangkat memang menghasilkan data yang cenderung

tidak pasti (*uncertain*). Sebagai contoh, data yang dihasilkan oleh sensor cenderung tidak pasti karena hilangnya data ketika transmisi, galat pada perangkat sensor itu sendiri atau karena lingkungan yang berubah-ubah [2].

Seringkali data pada lingkungan bersifat dinamis dan kontinu. Sebagai contoh, pada jaringan sensor, *gateway sensor* mengirim hasil secara kontinu, pemantauan cuaca mendapatkan data secara kontinu dengan komputasi waktu nyata. Hal tersebut menjadi tantangan tersendiri, yaitu pemrosesan *uncertain data streaming* dengan efektif dan efisien sehingga hasil didapatkan di waktu itu juga [2].

2.5 *Sliding Window*

Pengolahan data statis yang banyak pada memori sekunder membutuhkan waktu yang lama. Kini, paradigma sudah banyak beralih kepada data stream. *Sliding window* adalah salah satu metode pemrosesan data *stream* yang efisien. *Sliding window* hanya memproses W data *tuple* terakhir yang masuk. Tuple t hanya *hidup* selama masa hidupnya (*lifespan*) $[r.t_{arr}, r.t_{exp}]$, $r.t_{arr}$ adalah waktu datang dari *tuple* tersebut dan $r.t_{exp}$ adalah waktu kadaluarsanya. Sebagai contoh, jika sensor mengirim data terus menerus dalam waktu 10 menit dan ukuran $W = 5$ menit, maka *sliding window* hanya mengolah data yang tiba pada 5 menit terakhir.

Pada tugas akhir ini, *sliding window* digunakan untuk memproses *uncertain data streaming* yang ditampung pada *node*/persimpangan pada data spasial. Pemrosesan *uncertain data* pada *sliding window* ini menggunakan struktur data R-Tree agar pemrosesan semakin cepat.

2.6 Scala

Scala adalah bahasa pemrograman tingkat tinggi yang menggabungkan paradigma pemrograman berbasis objek dengan pemrograman fungsional. Scala adalah bahasa pemrograman berbasis objek karena setiap nilai adalah objek. Tipe dan *behaviour* dari objek didapat dari Class dan Trait. Class pada bahasa ini telah dikembangkan sehingga terdapat fitur *multiple inheritance*. Scala adalah bahasa pemrograman fungsional karena setiap fungsi adalah nilai. Sebagaimana bahasa fungsional lain, Scala memiliki mendukung fungsi anonim/*lambda*, *currying*, *higher-order function*, *nested function*, dan *pattern matching*.

Scala adalah bahasa yang *statically typed*. Bahasa ini dioperasikan pada Java Runtime Environment (JRE). Oleh karena itu, bahasa ini dapat langsung terintegrasi dengan Java. Meskipun *static-typing*, bahasa ini memiliki fitur *local type inference*. Dengan fitur tersebut, *programmer* tidak perlu menuliskan tipe data pada variabel [7].

Pada tugas akhir ini, Scala digunakan untuk mengimplementasikan struktur data dan algoritme. Bahasa ini juga digunakan untuk membuat layanan *webserver* yang berhubungan dengan peramban/*client* menggunakan kumpulan pustaka Akka.

2.7 JavaScript

JavaScript adalah bahasa pemrograman yang ringan dan *interpreted*. JavaScript adalah bahasa yang multiparadigma dan dinamis yang mendukung pemrograman berbasis objek, imperatif, dan deklaratif. JavaScript banyak digunakan pada peramban/*client-side*. Meskipun demikian, JavaScript juga banyak ditemukan pada aplikasi *server*/*server-side* menggunakan *Node.js* [9].

Pada tugas akhir ini, JavaScript digunakan sebagai bahasa pada *client-side* untuk proses simulasi dari implementasi algoritme yang diusulkan.

2.8 D3.js

D3.js adalah pustaka pada lingkungan bahasa pemrograman JavaScript yang digunakan untuk memanipulasi data. D3 membantu pengguna untuk memvisualisasikan data yang diinginkan pada lingkungan peramban menggunakan HTML, SVG, dan CSS. D3 dilengkapi dengan berbagai fungsi untuk memanipulasi dan merepresentasikan data. D3.js digunakan untuk membuat banyak pustaka berbasis grafik pada peramban, seperti neo4jd3, plotly.js, dan billboard.js. Jaringan jalan raya juga dapat diimplementasikan dengan pustaka ini.

Halaman ini sengaja dikosongkan

BAB III
DESAIN

3.1 Daftar Istilah

Bab ini dan selanjutnya akan menggunakan istilah-istilah sebagai berikut:

Tabel 3.1 Simbol dan deskripsi pada bab ini

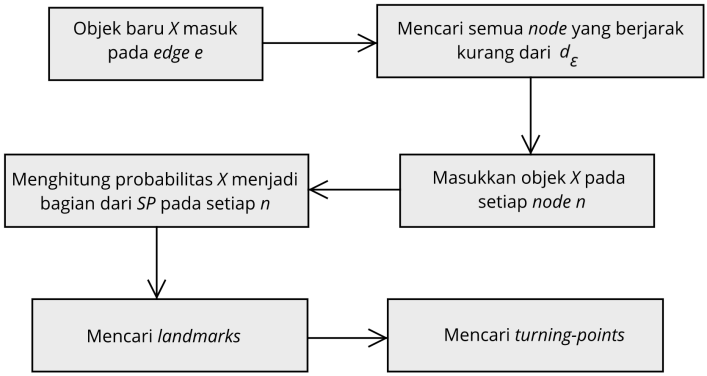
Simbol	**Deskripsi**
d	Dimensi data
$x[i]$	Nilai dari *tuple* x pada dimensi ke-i
X	Objek *uncertain data*, $x \in X$
U	*Uncertain data streaming*, $X \in U$
$Y \prec x$	Objek Y mendominasi *instance* x
$X \prec Y$	Objek X mendominasi objek Y
SP	*Skyline Point*, himpunan objek yang menjadi *skyline*
$Pr(x)$	Probabilitas tuple x
$SkyPr(x)$	Probabilitas kejadian x menjadi anggota dari *skyline*
L	*Landmark*

3.2 Pengantar

Tugas akhir ini mengusulkan metode *Continuous Streaming distance-based Skyline Query* ($CSd_\varepsilon - SQ$). Pencarian titik *skyline* pada jaringan jalan raya menggunakan algoritme $Cd_\varepsilon - SQ$[1]. Untuk mencari *skyline point*, SP dari suatu titik *query*, komputasi dilakukan untuk mencari semua objek yang memiliki jarak kurang dari sama dengan d_ε dari titik *query* tersebut. Dari objek-objek tersebut, metode ini mencari objek-objek yang tidak didominasi

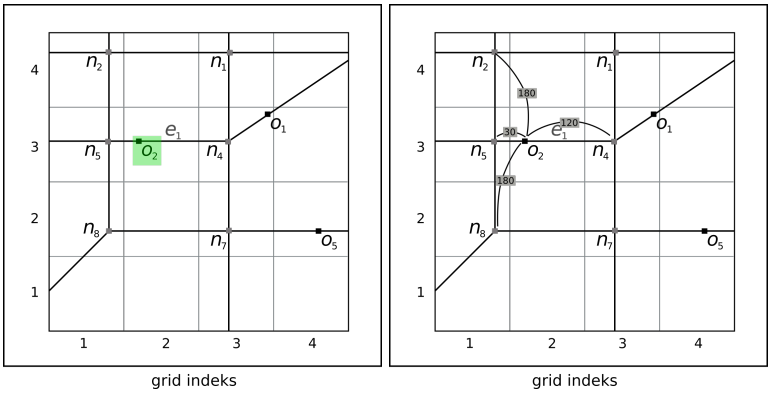
oleh objek lain, objek-objek tersebut dinamai *GSP*(*Global Skyline Points*).

Untuk menentukan *GSP* pada *uncertain data streaming*, algoritme yang digunakan adalah algoritme EPSU[2]. Tugas akhir ini menggunakan struktur data grid indeks agar algoritme hanya memproses data yang dibutuhkan saja. Agar pemrosesan *uncertain data* dapat dilakukan secara efisien, tugas akhir ini menggunakan struktur data R-Tree dan disimpan pada setiap *node*.



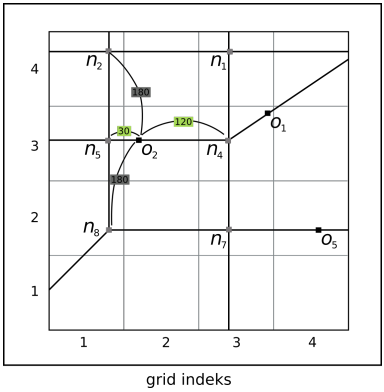
Gambar 3.1 Alur pemrosesan

Perhatikan Gambar 3.1 dan Gambar 3.2, saat objek X yang masuk dari *stream* pada suatu *edge* yang terdapat pada struktur Grid, algoritme BFS mencari semua *node* yang berjarak kurang dari d_ε dari objek X, $d_{X,n} \leq d_\varepsilon$.



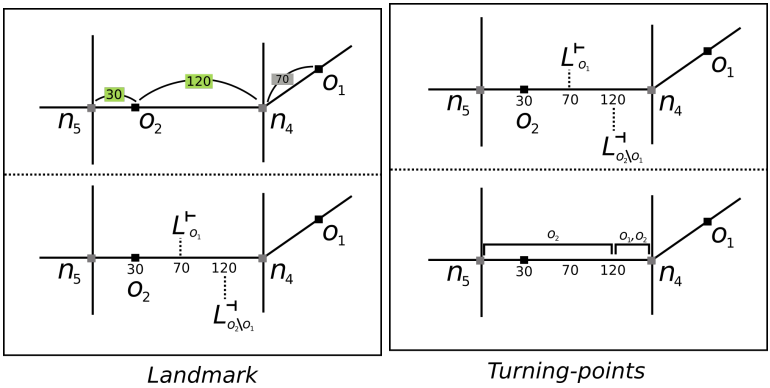
Gambar 3.2 Objek masuk pada grid indeks

Kemudian objek X dimasukkan pada struktur data R-Tree yang terdapat pada masing-masing *node* menggunakan algoritme *Insertion*. Setiap objek X bertahan pada Grid hanya dalam interval waktu yang sama *t*. Jika objek X sudah kadaluarsa, objek dikeluarkan dari *node* dengan algoritme *Deletion*.



Gambar 3.3 Dengan d_ε=150, o_2 hanya masuk pada *node* n_4 dan n_5

Setelah objek masuk pada semua *node* n $d_{X,n} \leq d_{\varepsilon}$, proses pencarian *landmark* dilakukan sebagai dasar pencarian *turning-point*. Masukan dari proses pencarian *Landmark* yaitu *GSP*. *GSP* adalah objek-objek yang menjadi SP^{ε} yang terdapat pada n_s dan n_e dan objek-objek yang terdapat pada edge e. Terakhir, penentuan *turning-point* dilakukan pada setiap *edge* yang berhubungan dengan n. Hasil akhir dari proses ini adalah *turning-point*.



Gambar 3.4 Kiri: *landmark* yang didapat dari *edge*. Kanan: *turning-point* didapat dari *landmark*

.

Suatu edge e pada jaringan jalan raya memiliki dua ujung *node*, yaitu *node* ujung awal n_s dan *node* ujung akhir n_e. Setiap *node* memiliki objek dan terhubung dengan *edge*. Dari proses *Skyline edge e* direpresentasikan dalam bentuk interval beserta *SP* yang terdapat pada masing-masing interval. Antara interval satu dengan interval lain terdapat pergantian objek yang menjadi *SP*. Titik pergantian *SP* ini diistilahkan dengan *turning-point*.

3.3 Struktur Data

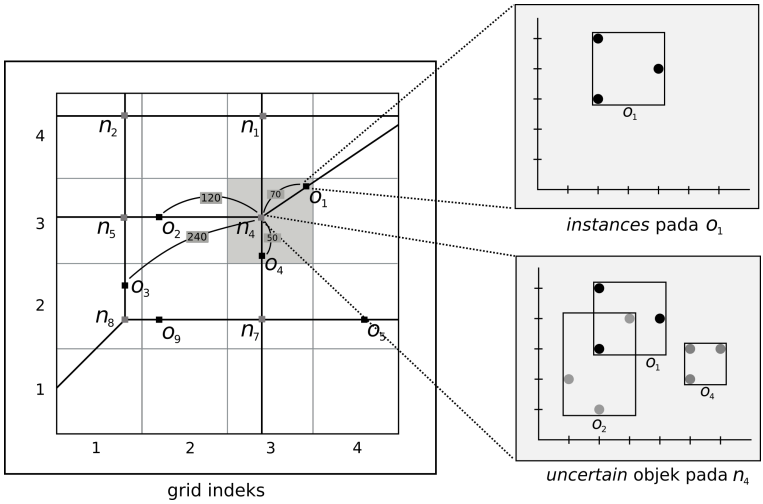
Jaringan jalan raya menggunakan jalan dan persimpangan sebagai objek utamanya. Hal tersebut dapat tersebut dapat dimodelkan dengan *undirected weighted graph* yang terdiri dari *node* dan *edge*. *Node* sebagai persimpangan dan *edge* sebagai jalan. Struktur ini efisien untuk pencarian data pada peta.

Struktur graf sederhana menjadi sangat berat untuk diolah apabila terdapat terlalu banyak *node*, *edge*, dan objek di dalamnya. Penulis menggunakan struktur data grid indeks pemrosesan data. **Grid indeks** adalah struktur graf yang terbagi-bagi menjadi kotak-kotak dengan ukuran $N \times N$. Setiap *edge* dan *node* menempati grid pada indeks m dan n.

Struktur grid menampung tiga tabel, yaitu tabel objek, *node*, dan *edge*. Perhatikan tabel objek pada Tabel 3.2, setiap objek koordinat x, y, dan *instances*. Atribut *instances* berisi *tuple-tuple* kejadian/titik dari objek. Dalam model matematis, setiap objek X memiliki *tuple-tuple* x, $x \in X$. Setiap tuple memiliki probabilitas yang dinotasikan dengan $Pr(x)$. Jumlah probabilitas pada semua *tuple* adalah 1, artinya $\sum_{x \in X} Pr(x) = 1$. Sebagai contoh, beberapa *instance* di satu objek pada bidang 2 dimensi dapat ditulis dengan $[(4, 5, 0.1), (5, 6, 0.5), (5, 7, 0.4)]$.

Tabel 3.2 Atribut dari objek

Atribut	**Deskripsi**
id	ID *objek*
x	Koordinat X
y	Koordinat Y
e	Edge tempat objek berada
$instances$	Semua instance dari objek



Gambar 3.5 Struktur data grid indeks

Perhatikan struktur tabel *node* pada Tabel 3.3. Setiap *node* yang terdapat pada grid menyimpan koordinat x, koordinat y, struktur R-Tree *SW-Tree*. Struktur R-Tree digunakan untuk menyimpan dan mengolah objek-objek *uncertain data* secara efisien. *SW-Tree* hanya menampung objek-objek yang berjarak kurang dari sama dengan d_ε. Jarak yang dimaksud adalah total panjang *edge*/jalan dari *node* menuju objek. Pada dasarnya, *SW-Tree* adalah struktur data R-Tree yang dimodifikasi agar dapat memproses *uncertain data* dalam bentuk SW(*sliding-window*). Terakhir, objek yang disimpan pada setiap *node* tersebut diberi informasi tambahan (*metadata*) agar algoritme tidak melakukan proses yang sama berulang-ulang.

Tabel 3.3 Atribut dari *node*

Atribut	**Deskripsi**
id	ID *node*
x	Koordinat X
y	Koordinat Y
SW-Tree	Struktur RTree untuk menyimpan dan mengolah *uncertain data*
M	Tabel *metadata* dari objek-objek yang disimpan

Perhatikan tabel *edge* pada Tabel 3.5. Tabel *edge* menyimpan n_i sebagai ID dari salah satu node, n_j sebagai ID dari *node* lainnya, len sebagai panjang *edge* tersebut, dan *objects*. Atribut *objects* pada *edge* adalah semua objek yang berada pada *edge* tersebut.

Tabel 3.4 Atribut dari *metadata* dari objek pada node n

Atribut	**Deskripsi**
id	ID dari objek
d	Jarak objek dari *node n*
$skyProb$	Probabilitas objek menjadi bagian dari SP
$isImpossible$	Tanda jika objek tidak dapat menjadi bagian dari SP

Tabel 3.5 Atribut dari *edge*

Atribut	Deskripsi
id	ID *edge*
n_i	ID *node* salah satu ujung
n_j	ID *node* ujung yang lain
len	Panjang *edge*
$objects$	Kandidat *skyline point*, yaitu semua objek yang berada pada *edge* tersebut

Tabel 3.4 menyimpan data yang melekat pada objek ketika objek sudah masuk pada *edge* e. Tabel tersebut menyimpan jarak *d*, yaitu jarak antara *node* dengan objek. Dengan adanya *d*, algoritme yang diusulkan tidak menggunakan komputasi *shortest-path*. *skyProb* menyimpan probabilitas objek menjadi bagian dari *SP* dan *skyProb* bernilai $0 \leq skyProb \leq 1$. Terakhir, *isImpossible* adalah *flag* yang menjadi tanda apabila objek tidak lagi dapat menjadi bagian dari *SP*.

3.4 Probabilitas *Skyline* pada *Uncertain Data*

Sub-bagian ini menjabarkan mengenai proses penghitungan Probabilitas *Skyline* dari suatu objek. Bidang d dimensi memiliki dataset U. Data set U memiliki objek *uncertain data* X sedemikian hingga $X \in U$. Setiap objek X memiliki *instances* x, $x \in X$. Jika objek X digambarkan pada bidang d dimensi, maka sebuah *minimum bounding rectangle*, MBR(X) secara tepat menampung semua *instance* dari X. X_{min} dan X_{max} adalah titik pojok terkecil dan terbesar dari X dengan ketentuan $X_{min} = min_{x \in X} x[i]$ dan $X_{min} = max_{x \in X} x[i]$, $i = 1, ..., d$. Gambar 3.6 menampilkan tiga objek *uncertain data* pada bidang dua dimensi beserta masing-masing MBR-nya.

Semua *instance*/*tuple* dari objek *uncertain data* X memiliki

probabilitas $Pr(x)$, dengan ketentuan $\sum_{x \in X} Pr(x) = 1$. Jika terdapat objek *uncertain data* yang lain Y, maka probabilitas Y mendominasi x adalah $Pr[Y \prec x] = \sum_{y \in Y, y \prec x} Pr(y)$. Demikian sebaliknya, probabilitas Y tidak mendominasi x adalah $Pr[Y \not\prec x] = \sum_{y \in Y, y \not\prec x} Pr(y)$. Untuk *instance* $x \in X$, SkyPr(x) adalah probabilitas dari *instance* x yang menjadi komponen untuk objek X agar objek X menjadi bagian dari *skyline*. Dalam kasus ini, probabilitas skyline dari x adalah hasil dari semua objek Y pada U yang tidak mendominasi x, $SkyPr(x) = \prod_{Y \in U, Y \neq X} Pr[Y \not\prec x]$. Dengan demikian, probabilitas objek X menjadi bagian dari skyline adalah $SkyPr(X) = \sum_{x \in X} Pr(x) * SkyPr(x)$ [2].

Sebagai contoh, perhatikan Gambar 3.6. Untuk menghitung $SkyPr(Z)$, kita perlu menghitung $SkyPr(z_1)$, $SkyPr(z_2)$, dan $SkyPr(z_3)$ terlebih dahulu. Pada kasus ini, objek lebih dominan apabila nilai x dan y lebih kecil. Penulis akan menjabarkan penghitungan $SkyPr(z_1)$. $SkyPr(z_2)$ dan $SkyPr(z_3)$ dapat dicari dengan cara yang sama. Titik z_1 didominasi oleh x_2, x_3, dan y_3, sehingga $Pr[Y \prec z_1]$ dan $Pr[X \prec z_1]$ perlu dihitung. $Pr[X \prec z_1] = Pr[x_2 \prec z_1] + Pr[x_3 \prec z_1] = 0.2 + 0.3 = 0.5$ dan $Pr[Y \prec z_1] = Pr[y_3 \prec z_1] = 0.3$. Maka $SkyPr(z_1) = Pr[X \not\prec z_1] \times Pr[Y \not\prec z_1] = (1 - Pr[X \prec z_1] \times Pr[Y \prec z_1]) = 1 - 0.5 * 0.3 = 0.85$. Dengan metode penghitungan yang sama untuk z_2 dan z_3, maka $SkyPr(Z) = 0.85 \times 0.1 + 0.85 \times 0.4 + 0.5 \times 0.5 = 0.675$.

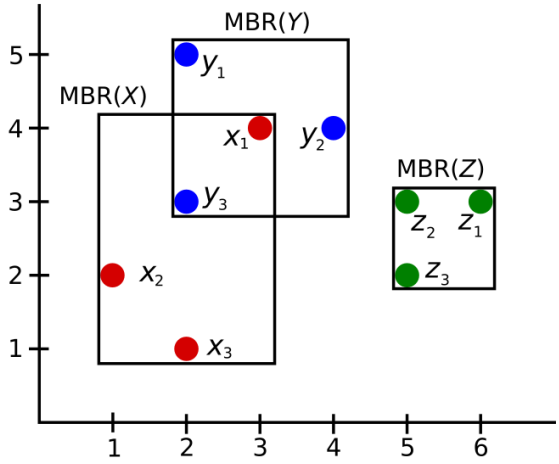
3.5 Pemrosesan *Uncertain Data Streaming*

Tugas akhir ini menggunakan metode EPSU [2] untuk menentukan *skyline*. *Uncertain data streaming* didefinisikan sebagai data yang terus menerus muncul dan akan kadaluarsa dalam periode tertentu. Setiap data yang masuk memiliki *timestamp* yang menunjukkan waktu data tersebut masuk dalam sistem.

Pemrosesan *uncertain data streaming* pada tugas akhir

Tabel 3.6 Contoh *uncertain data* beserta probabilitas *skyline*-nya

Objek	Instance	$Pr()$	$SkyPr()$	$SkyPr(object)$
X	$x_1(3,4)$ $x_2(1,2)$ $x_3(2,1)$	0.5 0.2 0.3	0.7 1 1	0.85
Y	$y_1(2,5)$ $y_2(4,4)$ $y_3(2,3)$	0.4 0.3 0.3	0.5 0 0.5	0.35
Z	$z_1(6,3)$ $z_2(5,3)$ $z_3(5,2)$	0.1 0.4 0.5	0.85 0.85 0.5	0.675

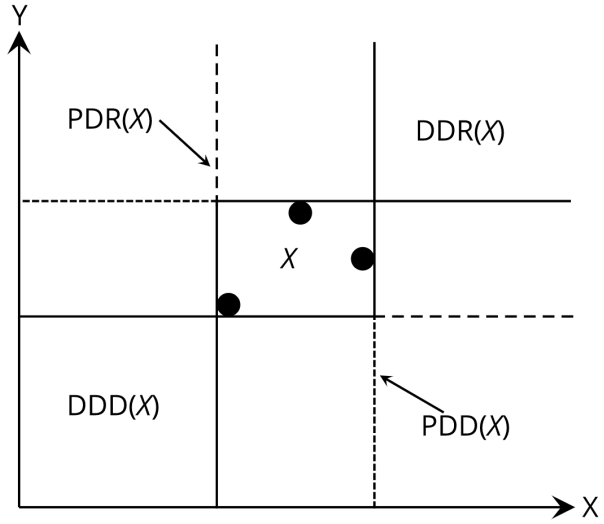


Gambar 3.6 X_1, X_2, dan X_3 beserta masing-masing MBR

ini menggunakan *sliding-window*, SW. SW memiliki ukuran $|W|$, artinya SW ini dapat menampung objek sejumlah $|W|$. EPSU menggabungkan struktur data R-Tree dan *sliding-window*,

SW-Tree. R-Tree digunakan karena struktur data ini efisien dalam pencarian data spasial menggunakan MBR (*Minimum Bounding Rectangle*). *Threshold* p juga perlu didefinisikan untuk proses *pruning* dan untuk menjadi acuan dari dominasi objek satu dengan objek lainnya.

EPSU menggunakan MBR untuk mengurangi komputasi yang berulang-ulang pada setiap instance dari objek *uncertain data*. Terdapat empat regional yang dimiliki setiap objek *uncertain data* X:



Gambar 3.7 Empat area pada bidang 2 dimensi

1. *Definitely Dominating Region X*, DDR(X), yaitu area yang didominasi penuh oleh X. Objek yang terdapat pada area ini tidak perlu dibandingkan dengan *instance* dari X. Objek Y yang terdapat pada DDR(X) berarti Y didominasi oleh X dan Y tidak dapat menjadi bagian dari *SP*.
2. *Probably Dominating Region X*, PDR(X), yaitu area yang

berkemungkinan dinominasi oleh X. Untuk mendapatkan probabilitas *skyline*, *instance* yang terdapat pada area ini perlu dibandingkan dengan *instance* yang terdapat pada X.

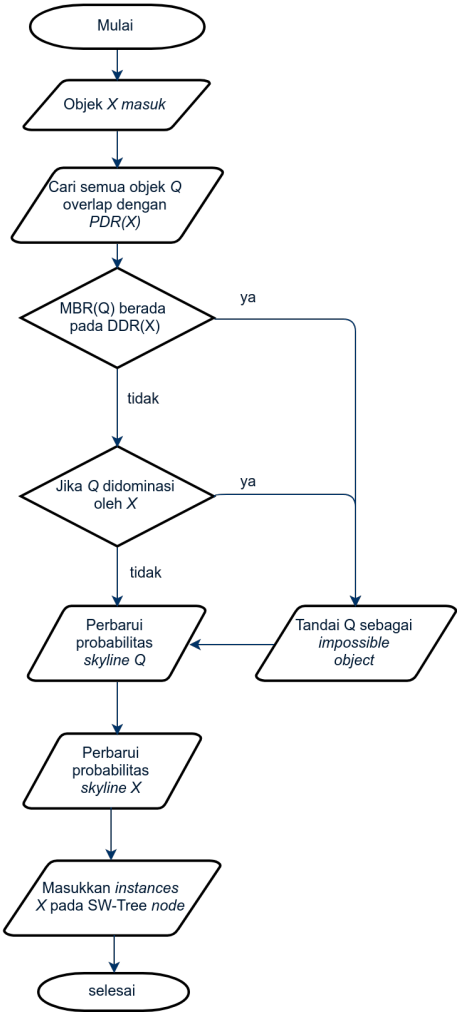
3. *Definitely DominateD X* , DDD(X), yaitu area yang mendominasi X secara penuh. Objek Y yang terdapat pada DDR(X) berarti X didominasi oleh Y dan X tidak dapat menjadi bagian dari *SP*.
4. *Probably DominateD X*, PDD(X), adalah area yang mungkin mendominasi *X*.

Jika S_{max} dan S_{min} adalah titik maksimal dan minimal pada bidang *d* dimensi, maka DDR(X), PDR(X), DDD(X), dan PDD(X) adalah [X_{max}, S_{max}], [X_{min}, S_{max}], [S_{min}, X_{min}], dan [S_{min}, X_{max}]. Perhatikan Gambar 3.7.

Pada tugas akhir ini, terdapat dua proses yang terjadi pada *sliding window*, yaitu *Insertion* dan *Deletion*.

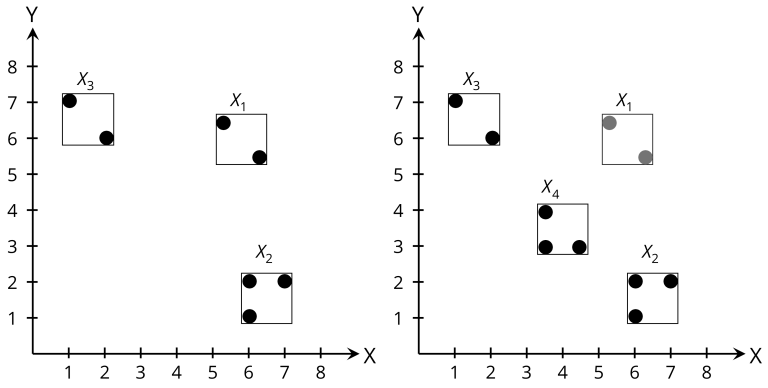
3.6 Proses *Insertion*

Proses *Insertion* yaitu proses masuknya objek *uncertain* pada *SW-Tree*. Hasil dari proses ini yaitu berupa *node* yang telah diperbarui. Perbaruan yang dilakukan adalah penambahan objek pada *SW-Tree* dan pembaruan probabilitas *skyline* dari objek-objek yang terkena pengaruh. Objek yang terkena pengaruh dari masuknya objek *uncertain* X adalah objek-objek yang *overlap* dengan PDR(X).



Gambar 3.8 Alur objek masuk pada *node*

Perhatikan Gambar 3.9, pada gambar sisi kiri, objek X_1, X_2, dan X_3 masing-masing memiliki probabilitas menjadi bagian dari

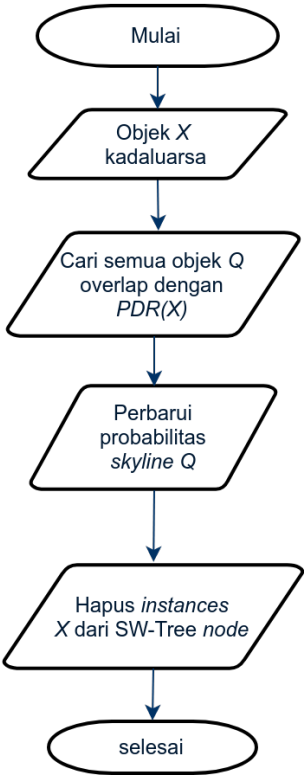


Gambar 3.9 Objek X_4 masuk ke SW-Tree, objek X_1 ditandai sebagai objek *impossible*.

skyline. Ketika objek X_4 masuk, objek X_1 dinyatakan sebagai objek *impossible*.

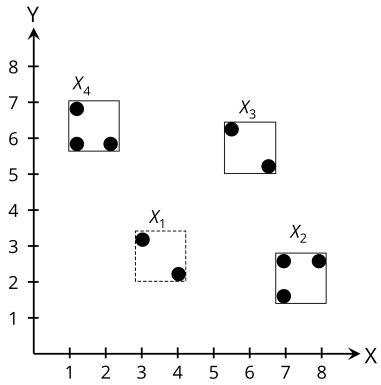
3.7 Proses *Deletion*

Proses *Deletion* adalah proses keluarnya objek dari *SW-Tree* karena objek telah mencapai waktu kadaluarsanya. Objek lama yang perlu dikomputasi ulang yaitu objek yang memiliki *MBR* yang *overlap* dengan PDR(X). Ketika objek X dihapus, penghitungan probabilitas *skyline* hanya dilakukan pada objek yang tidak ditandai.



Gambar 3.10 Alur proses *Deletion*

Perhatikan Gambar 3.11, ketika objek X_1 kadaluarsa dan keluar dari *SW-Tree*, objek yang *overlap* dengan PDR(X_1) (objek X_2 dan X_3) perlu dikomputasi ulang probabilitas *skyline*-nya. Objek X_4 tidak dikomputasi ulang karena tidak *overlap*.

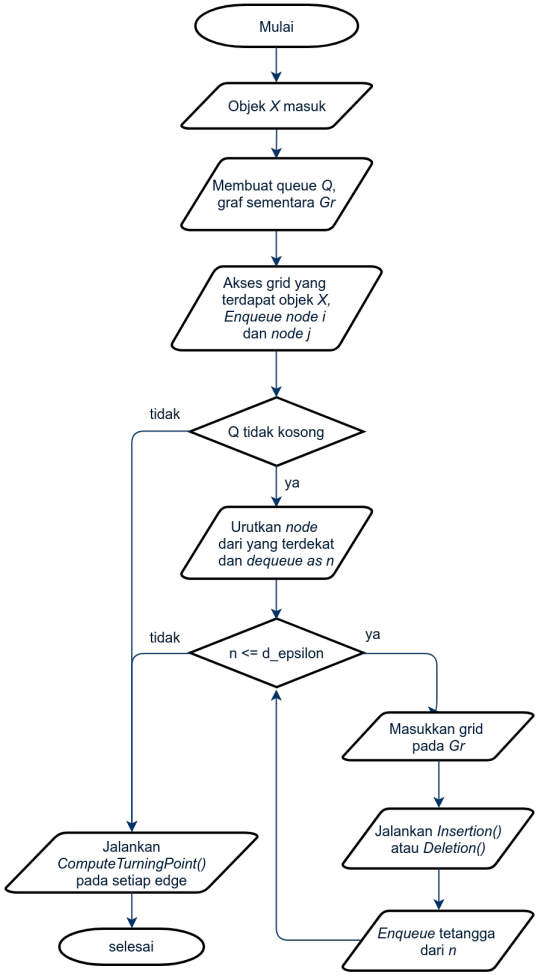


Gambar 3.11 Objek X_1 kadaluarsa dam keluar dari SW-Tree

3.8 Penentuan *GSP*(*Global Skyline Points*)

Setiap *edge* e pada graf memiliki dua ujung berupa *node*, yaitu n_s dan n_e. GSP_e adalah semua objek yang menjadi bagian dari *SP* pada *edge* e. Objek-objek yang menjadi bagian dari GSP_e adalah objek yang berada pada edge $e(S_e)$, objek yang menjadi SP pada $n_s(SP_{n_s})$, dan objek yang menjadi SP pada $n_e(SP_{n_e})$.

SP_{n_s}(dan SP_{n_e}) adalah SP dari semua objek yang berjarak kurang dari d_ε dari n_s(dan n_e). Setiap objek pada n_s(dan n_e) memiliki probabilitas untuk menjadi bagian dari SP, namun hanya objek yang tidak didominasi saja yang dapat menjadi bagian dari SP. Dominasi ini dapat dilihat dari dua jenis atribut, yaitu atribut statis(*uncertain data*) dan atribut dinamis(jarak) sebagaimana yang telah dijelaskan pada sub-bab 3.6. Secara matematis, *GSP* pada *edge* e adalah gabungan dari kandidat *skyline point* yang terdapat pada n_s, n_e, dan *edge* e itu sendiri, $SP_p^\varepsilon = SP_{n_s} \cup SP_{n_e} \cup SP_e$.



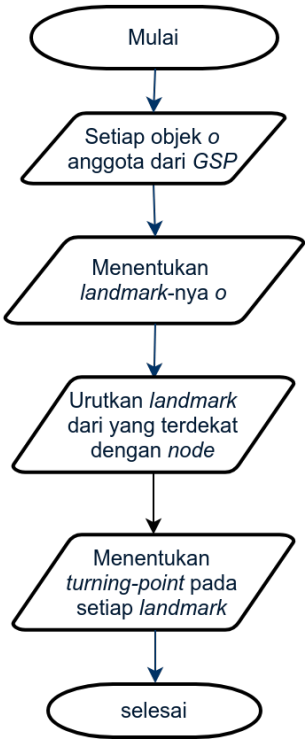
Gambar 3.12 Alur pencarian *GSP*

Dalam hal ini, objek bersifat dinamis dan *node* bersifat statis. Maksudnya, objek setiap waktu dapat masuk dan keluar dalam interval tertentu dan *node* tidak dapat berpindah. Ketika objek

masuk ke *Grid* pada waktu t, objek dimasukkan pada struktur RTree yang terdapat pada *node* yang berjarak kurang dari sama dengan d_ε. Pencarian SP_n^ε dilakukan ketika objek masuk dan keluar/kadaluarsa pada node n.

3.9 Proses *Compute Turning Point*

Proses *Compute Turning Point* dilakukan ketika semua objek telah masuk pada *node* n dengan $d_{n,X} \leq d_\varepsilon$. Objek yang ditandai sebagai *impossible* tidak dilakukan penghitungan probabilitas *skyline*.



Gambar 3.13 Alur pencarian *turning-point*

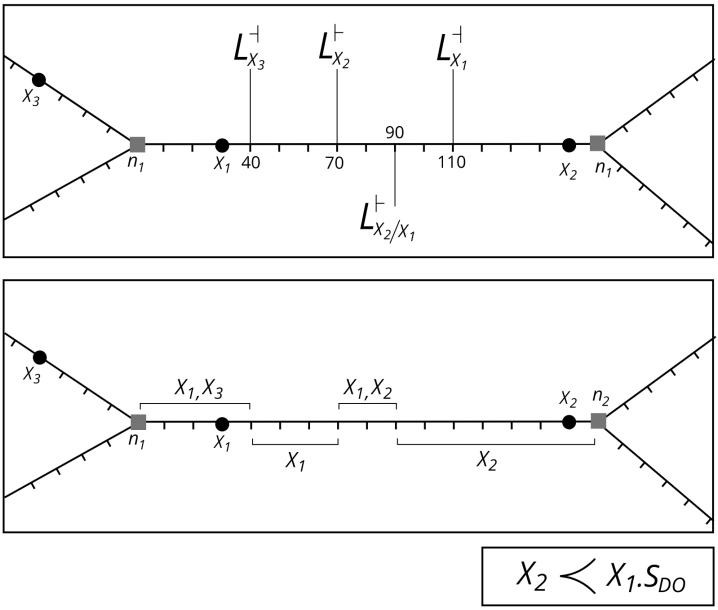
Langkah pertama dalam menentukan *skyline* adalah penentuan *landmark*. *Landmark* adalah area yang dimiliki objek sehingga objek memungkinkan menjadi bagian dari *SP* di area tersebut. Hanya anggota dari *GSP* yang dipakai untuk menentukan *landmark-landmark* berikut. Algoritma ini mencari empat *landmark* yang didefinisikan sebagai berikut:

1. $L_o^{\vdash}$: Asumsikan titik *query* q bergerak di edge e dari arah n_s menuju n_e dan jarak $d_{q,o}$ antara objek *o* dan titik *query* q lebih besar daripada d_ε serta titik *query* q bergerak menuju objek o. $L_o^{\vdash}$ adalah titik *landmark* yang terdapat pada edge e dan berjarak sejauh d_ε dari objek o. Objek *o* menjadi bagian dari $d_\varepsilon - SP$ ketika melewati *landmark* ini.
2. $L_o^{\dashv}$: Asumsikan titik *query* q bergerak menjauhi objek o menuju n_e. *Landmark* $L_o^{\dashv}$ adalah titik ketika jarak $d_{q,o}$ sama dengan d_ε dan titik ini berada pada edge e. Objek *o* tidak lagi dapat menjadi bagian dari $d_\varepsilon - SP$ setelah melewati *landmark* ini.
3. $L_{o\backslash o'}^{\vdash}$: Asumsikan objek *o* mendominasi objek lain, o' dalam hal atribut statis (*uncertain data*). *Landmark* ini berada pada titik *query q* ketika $d_{q,o} = d_{q,o'}$. Dengan *landmark* ini, ketika titik *query* q berada di sisi kiri dari *landmark* ini, objek o tidak dapat mendominasi objek o' dalam hal atribut dinamis (jarak $d_{q,o'} < d_{q,o}$). Ketika titik *query q* berada di sisi kanan *landmark*, objek *o* mendominasi objek o', baik dari sisi atribut dinamis(jarak) ataupun statis(*uncertain data*).
4. $L_{o\backslash o'}^{\dashv}$: Asumsikan objek *o* mendominasi objek lain, o' dalam hal atribut statis (*uncertain data*). Sebelum titik *query* q mencapai *landmark* ini, objek *o* dapat mendominasi o' dalam hal atribut statis dan dinamis. Ketika titik *query* q melewati *landmark* ini, objek o tidak dapat mendominasi objek o' karena $d_{d,o'} < d_{d,o}$.

Penentuan *turning-point* dilakukan pada *landmark* yang

lebih dekat dengan n_s terlebih dahulu. Berikut metode pencarian *turning-point* berdasarkan *landmark* yang sudah ada.

1. Dalam pemrosesan *landmark* $L_o^{\vdash}$, jika *landmark* $L_{'o\backslash o}^{\dashv}$ terdapat pada *queue* Q, maka $L_o^{\vdash}$ tidak menjadi *turning-point*. Hal tersebut dikarenakan, meskipun titik *query* q berada di $[L_o^{\vdash}, L_{o'\backslash o}^{\dashv}]$ sehingga $d_{q,o} < d_{\varepsilon}$, objek o masih didominasi o' hingga mencapai $L_{o'\backslash o}^{\dashv}$. Jika tidak terdapat $L_{o'\backslash o}^{\dashv}$, maka $L_o^{\vdash}$ dapat menjadi *turning-point*.
2. *Landmark* $L_o^{\dashv}$ menjadi *turning-point* apabila titik *query* q berada di kiri $L_o^{\dashv}$ dan objek o menjadi $d_{\varepsilon} - SP$. Ketika q melewati $L_o^{\dashv}$, $d_{q,o}$ menjadi lebih besar dibandingkan d_{ε} sehingga objek o dikeluarkan dari $d_{\varepsilon} - SP$. Dengan demikian, $L_o^{\dashv}$ mejadi *turning-point*.
3. *Landmark* $L_{o\backslash o'}^{\vdash}$ menjadi *turning-point* dengan kondisi berikut: ketika titik *query* q mencapai $L_{o\backslash o'}^{\vdash}$, objek o akan mendominasi o' dalam hal atribut statis maupun dinamis. Jika saat tersebut kedua objek sedang menjadi bagian dari $d_{\varepsilon} - SP$, objek o' tidak lagi menjadi bagian dari $d_{\varepsilon} - SP$ setelah melewati $L_{o\backslash o'}^{\vdash}$.
4. *Landmark* $L_{o\backslash o'}^{\dashv}$ menjadi *turning-point* dengan kondisi berikut: *landmark* $L_{o\backslash o'}^{\dashv}$ berarti objek o tidak dapat mendominasi objek o' setelah titik *query* q mencapai $L_{o\backslash o'}^{\dashv}$. Jika objek *o* ketika itu menjadi bagian dari $d_{\varepsilon} - SP$ dan tidak ada *landmark* $L_{o''\backslash o'}^{\dashv}$ yang terdapat pada Q, maka objek o' dapat menjadi bagian dari $d_{\varepsilon} - SP$ setelah q melewati $L_{o\backslash o'}^{\dashv}$. Dengan demikian, $L_{o\backslash o'}^{\dashv}$ menghasilkan *turning-point*.



Gambar 3.14 Contoh penentuan *landmark* dan *turning-point*

Perhatikan Gambar 3.14, terdapat 3 objek X_1, X_2, dan X_3. Objek X_2 mendominasi X_1 dalam hal atribut statis (*uncertain data*). Pada titik *node* n_1, nilai awal SP^{ε} adalah SP_{n_s}, yaitu X_1, X_3. Berikut penjelasan masing-masing *landmark* yang terdapat pada Gambar 3.14:

1. *Landmark* $L^{\dashv}_{X_3}$: Karena $d_{X_3,q} > d_{\varepsilon}$, objek X_3 dikeluarkan dari SP^{ε}. Sehingga *landmark* $L^{\dashv}_{X_3}$ menjadi *turning-point* dan SP^{ε} berganti dari $\{X_1, X_3\}$ menjadi $\{X_1\}$.
2. *Landmark* $L^{\vdash}_{X_2}$: Karena $d_{X_2,q} < d_{\varepsilon}$, objek X_2 menjadi bagian dari SP^{ε}. Objek X_2 tidak dapat mendominasi objek X_2 karena $d_{X_1,q} < d_{X_2,q}$. Dengan demikian, *landmark* $L^{\vdash}_{X_2}$ menjadi *turning-point* dan SP^{ε} berganti dari $\{X_1\}$ menjadi

$\{X_1, X_2\}$.

3. *Landmark* $L^{\vdash}_{X_2 \setminus X_1}$: Pada titik ini, objek X_2 mulai mendominasi X_1 karena $d_{X_1,q} > d_{X_2,q}$. Dengan demikian, objek X_1 keluar dari SP^{ε}. *Landmark* $L^{\vdash}_{X_2 \setminus X_1}$ menjadi *turning-point* dan SP^{ε} berubah dari $\{X_1, X_2\}$ menjadi $\{X_2\}$.
4. *Landmark* $L^{\dashv}_{X_1}$: Karena objek X_1 tidak menjadi bagian dari SP^{ε}, maka *landmark* ini tidak dapat menjadi *turning-point* dan SP^{ε} tetap berisi $\{X_2\}$.

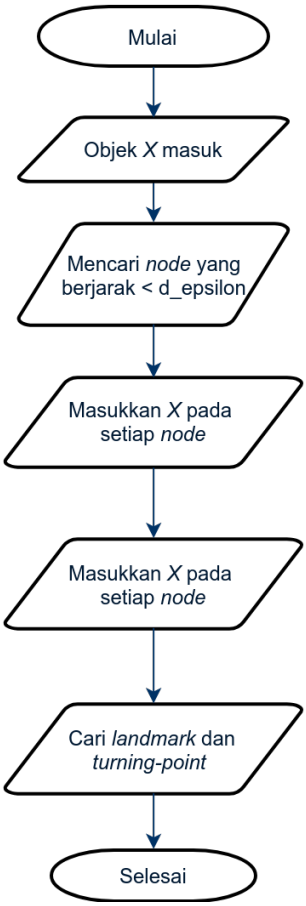
3.10 Algoritme *Naive*

Algoritme *naive* dibuat untuk membandingkan dengan algoritme CSd_{ε}. Algoritme ini tidak serta merta menggunakan metode *brute-force* dan tetap menggunakan algoritma yang lebih efisien seperti algoritme Dijkstra.

Metode ini menyimpan data *edge*, *node*, dan objek dalam bentuk *array* dan tidak menggunakan struktur data grid indeks. Metode ini juga menggunakan *array* untuk menyimpan objek yang terdapat pada *node*.

Objek X masuk pada *edge* dan mencari *node* yang berjarak kurang dari d_{ε}. Pencarian *node* dilakukan dengan cara mencari jarak terdekat antara objek dengan semua *node* yang ada pada graf. Penentuan jarak antara objek dengan *node* menggunakan algoritma Dijkstra. Setelah itu, *node* yang dimasuki objek yaitu *node* yang berjarak kurang dari sama dengan d_{ε}.

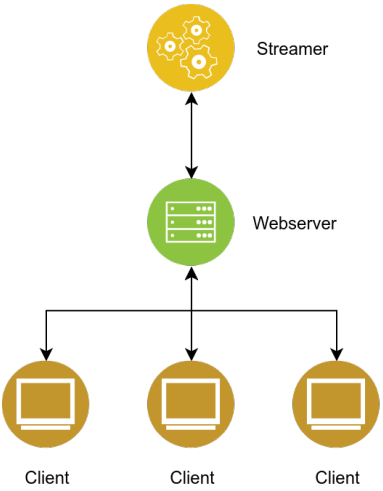
Objek yang masuk pada *node* disimpan pada *array*. Proses penghitungan *skyline* dilakukan dengan cara membandingkan objek satu dengan semua objek yang lain. Hal ini berbeda dengan algoritme CSd_{ε} yang hanya membandingkan objek dengan objek yang berada pada area tertentu saja.



Gambar 3.15 Alur metode *naive*

Langkah selanjutnya, untuk menentukan *landmark* dan *turning-point*, algoritme yang digunakan pada metode *Naive* sama dengan algoritme yang digunakan pada metode CSd_{ε}.

3.11 Perancangan Arsitektur Implementasi



Gambar 3.16 Arsitektur implementasi

Proses implementasi algoritma ini menggunakan antarmuka pengguna grafis dengan peramban. Terdapat tiga entitas, yaitu:

1. Streamer: Entitas ini digunakan sebagai pemrosesan data utama. Algoritma yang diusulkan pada tugas akhir ini diimplementasikan pada entitias ini. Implementasi bagian ini menggunakan bahasa Scala.
2. Client: Entitas ini digunakan untuk menampilkan antarmuka pengguna grafis menggunakan peramban dengan bahasa JavaScript. D3.js digunakan untuk merepresentasikan objek, *node*, dan *edge* dalam bentuk gambar.
3. Webserver: Entitas ini digunakan sebagai penghubung antara Streamer dengan Client. Entitas ini mendapatkan data dari Streamer. Data dikumpulkan dan dikonversi agar Client dapat

menerima data siap pakai.

Streamer hanya mengirimkan data kepada Webserver. Webserver menerima dan memproses data. Pemrosesan yang dilakukan berupa mengganti *turning-point* lama dengan yang terbaru. Client meminta data *turning-point* serta objek yang terdapat pada grid indeks secara kontinu dalam interval tertentu kepada Webserver. Data yang dikirim dari Webserver menuju Client berupa struktur *turning-point*.

Halaman ini sengaja dikosongkan

BAB IV
IMPLEMENTASI

4.1 Pengantar

Bab ini memaparkan metode yang diusulkan pada Bab 3 dalam bentuk *pseudocode*.

4.2 Implementasi Menentuan *GSP*

Metode ini menggunakan algoritme *Breadth First Search* untuk *traversing* pada graf. *Queue* Q digunakan untuk menyimpan antrian yang berisi *node* n beserta jarak n terhadap X ($d_{X,n}$). Untuk mengurangi komputasi yang berlebih dalam pencarian *node*, struktur graf sementara Gr dibuat untuk menampung data yang dibutuhkan. Setiap *node* yang dikenali dimasukkan kedalam *queue* Q. Proses *sorting* berdasarkan jarak dilakukan pada Q sebelum proses *dequeue*. Hanya objek yang berjarak kurang dari sama dengan d_ε saja yang diproses. Pemrosesan yang dilakukan tergantung dari jenis objek, jika objek baru maka dijalankan $Insertion()$ pada *node* n, jika objek kadaluarsa maka dijalankan $Deletion()$. Perlu diketahui bahwa, proses pencarian SP_{n_s} (atau SP_{n_e}) hanya dilakukan pada $Insertion()$ dan $Deletion()$. Setelah semua *node* $d_{X,n} < d_\varepsilon$ diperbarui, *GSP* dicari dari setiap *edge* yang memiliki *node* yang diperbarui dan dilanjutkan dengan pencarian *turning-point* dari *GSP* tersebut.

Algorithm 1 DetermineGSP

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

Gambar 4.1 Algoritme *DetermineGSP*

4.3 Implementasi Fungsi *Insertion()*

Sebuah objek *uncertain* X mendominasi objek *uncertain* lain Y, $X \prec Y$, dilihat dari dua jenis atribut, yaitu atribut statis(*uncertain data*) dan atribut dinamis(jarak). Pertama, dilihat dari atribut statis, X mendominasi Y apabila jumlah probabilitas

dari setiap *instance* dari Y yang terdapat pada DDR(X) lebih dari $1 - p$. Secara matematis hal tersebut disimbolkan dengan $\sum_{q \text{ in MBR(Y)} \cap \text{DDR(X)}} Pr(q) > (1 - p)$. Kedua, X mendominasi Y apabila jarak X kurang dari Y.

Algorithm 2 *Insertion*

```
1:  Input: uncertain data object X, threshold p
2:  for each object Q overlapped with PDR(X) do
3:      if MBR(Q) is within DDR(X) then
4:          mark Q as impossible object
5:      else
6:          if Σ_{q in MBR(Q)∩DDR(X)} Pr(q) > (1 − p) then
7:              mark Q as as impossible object
8:          else
9:              update SkyPr(Q) with X
10: compute SkyPr(X) with objects overlapped with PDD(X)
11: insert X into SW-Tree
```

Gambar 4.2 Algoritme *Insertion*

Jika MBR(Q) berada pada DDR(X), maka objek tersebut ditandai sebagai objek *impossible* dan dinyatakan tidak dapat menjadi bagian dari *skyline*. Jika MBR(Q) berada pada PDR(X), maka objek tersebut perlu dicek apakah probabilitasnya diatas *threshold* p. Jika jumlah probabilitas dari *instance* Q yang berada pada DDR(X) lebih dari $1 - p$, maka objek tersebut ditandai dan dinyatakan tidak dapat menjadi bagian dari *skyline*. Selanjutnya, penghitungan probabilitas *skyline* hanya dilakukan pada objek yang tidak ditandai. Probabilitas *skyline* dari objek yang ditandai tidak perlu diperbarui karena sudah tidak mungkin menjadi bagian dari *skyline*.

4.4 Implemantasi Fungsi *Deletion*

Algorithm 3 *Deletion*

1: **Input:** expired uncertain data U, threshold p
2: **for** each object Q overlapped with *PDR(X) and not marked* **do**
3: update SkyPr(Q) with removal of X
4: Remove X from *SW-Tree*

Gambar 4.3 Algoritme *Deletion*

Objek *uncertain* X yang keluar dari *node* karena kadaluarsa dapat mempengaruhi probabilitas *skyline* dari objek yang lain. Pada setiap objek yang *overlap* dengan X, penghitungan probabilitas *skyline* dilakukan untuk memastikan objek tersebut memiliki kemungkinan yang terbaru.

4.5 Implementasi Penentuan *Turning-point*

Algoritma dibawah mengilustrasikan penentuan *landmark* pada *edge* e yang memiliki *node* awal n_s dan *node* lainnya n_e. Queue Q dibuat untuk menampung *landmark*. Penentuan landmark ini berdasarkan pada objek-objek yang menjadi anggota dari *GSP*. Setelah mendapatkan semua landmark, *queue* Q diurutkan berdasarkan jarak terdekat dari *node* n_s.

Algorithm 4 ComputeTurningPoint 1

1: **Input:** threshold p, a distance d_ε, a set *GSP*, an edge *e* connecting two nodes n_s and n_e
2: **Output:** A set of tuples in form of $< [n_i, n_j], SP^\varepsilon >$ where SP^ε is the $d_\varepsilon - SP$ set between $[n_i, n_j]$
3: create an empty queue Q
4: **for** object $o \in GSP$ **do**
5: determine o's landmarks $L_o^\vdash$ and $L_o^\dashv$
6: **if** $L_o^\vdash$ is on *e* **then** insert $L_o^\vdash$ into Q
7: **if** $L_o^\dashv$ is on *e* **then** insert $L_o^\dashv$ into Q
8: **for** object $o' \in (GSP - \{o\}) \cap \sum_{q \text{ in } MBR(o') \cap DDR(o)} Pr(q) > (1-p)$ **do**
9: determine o's landmarks $L_o^\vdash$ or $L_o^\dashv$
10: **if** $L_{o\backslash o'}^\vdash$ is on e and $L_o^\vdash$ is closer to n_s than $L_{o\backslash o'}^\vdash$ **then**
11: insert $L_{o\backslash o'}^\vdash$ into Q
12: **if** $L_{o\backslash o'}^\dashv$ is on e and $L_{o\backslash o'}^\dashv$ is closer to n_s than $L_o^\dashv$ **then**
13: insert $L_{o\backslash o'}^\dashv$ into Q
14: sort landmarks in Q in ascending order of their distances to n_s

Gambar 4.4 Algoritme *Compute Turning Point* bagian 1

Algoritma ini memiliki masukan p, jarak d_ε, *GSP*, e, n_s, dan n_e. Pemrosesan pertama adalah pencarian *landmark* untuk mendapatkan batas-batas dari setiap objek agar dapat menjadi bagian dari SP^ε. Pada setiap objek yang terdapat pada *GSP*, pencarian *landmark* dimulai dari pencarian landmark $L_o^\vdash$ dan $L_o^\dashv$ (baris 4-7). Kedua *landmark* tersebut digunakan untuk mencari *landmark* $L_{o\backslash o'}^\dashv$ dan $L_{o\backslash o'}^\vdash$. *Landmark* $L_{o\backslash o'}^\dashv$ dan $L_{o\backslash o'}^\vdash$ hanya dicari pada objek yang saling mendominasi (baris 8). Apabila *landmark*

$L^{\vdash}_{o\backslash o'}$ terdapat pada e dan *landmark* $L^{\vdash}_{o}$ lebih dengan kepada n_s dibandingkan $L^{\vdash}_{o\backslash o'}$, maka *landmark* tersebut dimasukkan pada *queue* dan demikian juga untuk *landmark* $L^{\dashv}_{o\backslash o'}$ (baris 10-13). Hasil dari *queue* Q diurutkan berdasarkan jarak yang lebih dekat dengan n_s terlebih dahulu.

Algorithm 5 ComputeTurningPoint 2

```
1:  /* determining the result turning points */
2:  while Q is not empty do
3:      dequeue o.L
4:      switch o.L do
5:          case L⊢_o
6:              if there is no L⊣_{o'\o} then
7:                  return < [n_i, L⊢_o], SP^ε >, n_i = L⊢_o,
8:                  and add o into SP^ε
9:          case L⊣_o
10:             if o ∈ SP^ε then
11:                 return < [n_i, L⊣_o], SP^ε >, n_i = L⊣_o,
12:                 and remove o from SP^ε
13:         case L⊢_{o\o'}
14:             if o ∈ SP^ε and o' ∈ SP^ε then
15:                 return < [n_i, L⊢_{o\o'}], SP^ε >, n_i = L⊢_{o\o'},
16:                 and remove o' from SP^ε
17:         otherwise
18:             if o ∈ SP^ε and there is no L⊣_{o''\o'} ∈ Q then
19:                 return < [n_i, L⊣_{o\o'}], SP^ε >,
20:                 n_i = L⊣_{o\o'}, and add o' into SP^ε
```

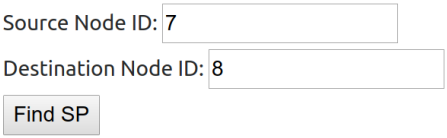
Gambar 4.5 Algoritme *Compute Turning Point* bagian 2

Setelah semua *landmark* didapatkan, pencarian *turning-point* dilakukan. *Turning-point* ini adalah hasil dari algoritma. *Turning-point* berupa sekumpulan *tuple* yang merepresentasikan interval awal n_i, interval akhir n_e, dan objek-objek yang menjadi SP pada interval tersebut SP^{ε}.

4.6 Implementasi Antarmuka Simulasi

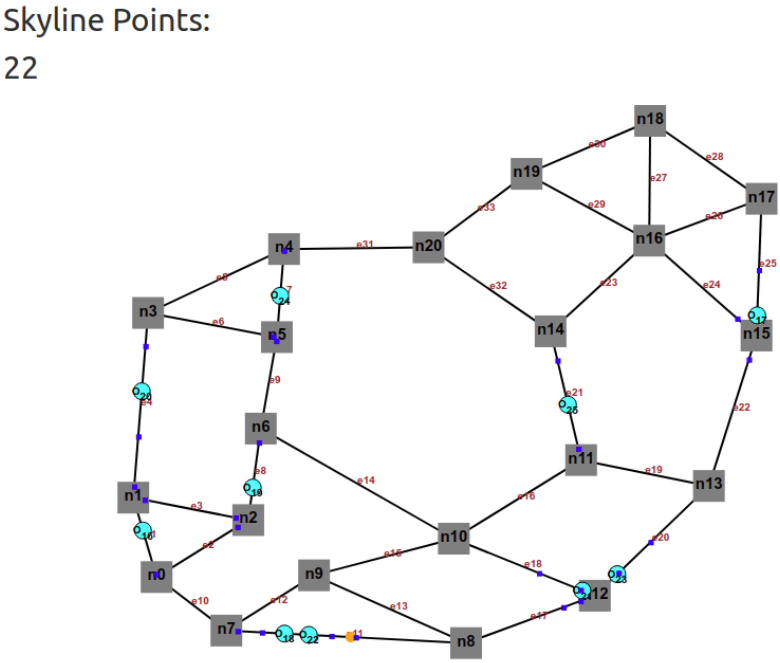
Implementasi antarmuka algoritma pada tugas akhir ini menggunakan pustaka D3.js. Pustaka tersebut mudah diimplementasikan karena memiliki banyak fungsi berkaitan dengan visualisasi. *Node*, *edge*, dan objek dapat digambarkan pada peramban dengan format SVG.

Antarmuka ini menampilkan *node*, *edge*, objek, *turning-point*, titik *query*, serta SP^{ε} yang dimiliki oleh titik *query*. *Node* direpresentasikan dengan kotak abu-abu sebagai persimpangan. *Edge* sebagai penghubung antar-*node*. Objek digambarkan dengan bulatan abu-abu. *Turning-point* digambarkan dengan kotak-kotak kecil yang berada pada *edge*. Titik *query* digambarkan dengan bulatan kuning yang bergerak.



Gambar 4.6 Antarmuka masukan *node* awal dan *node* tujuan

Perhatikan Gambar 4.6, pengguna dapat mencari rute yang diinginkan menggunakan dua masukan, yaitu *node* awal dan *node* tujuan. Setelah pengguna menekan *Find SP*, sistem akan mencari rute terdekat menggunakan algoritma Dijkstra.



Gambar 4.7 Objek yang menjadi SP pada momen diatas adalah objek 22.

Gambar 4.7 memperlihatkan titik *query* (kuning) yang bergerak dari n7(*node* 7) menuju n8 setelah pengguna menekan tombol *Find SP*. Sistem menampilkan objek-objek yang menjadi *skyline*/SP. Nomor-nomor objek tersebut tertulis pada pojok kiri atas. Ketika titik *query* berada di tengah-tengah *edge*, objek yang menjadi SP pada saat tersebut adalah objek 22 (o_{22}).

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 *instance* 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[8] dengan irisan garis lintang antara 32.0 hingga 37.0 dan garis bujur antara -120.0

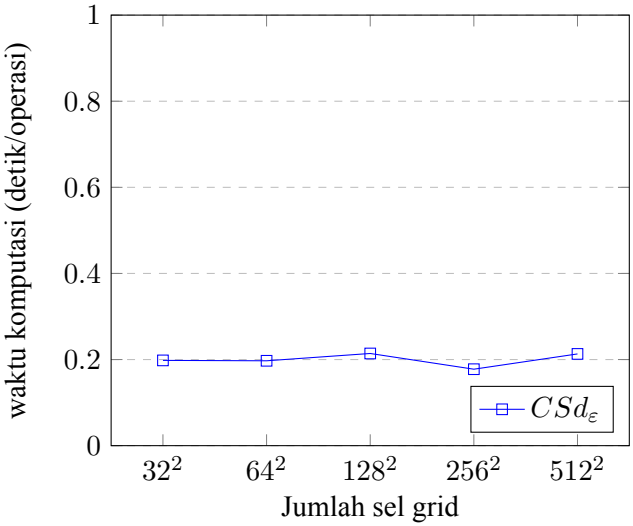
hingga -114.0. Pada koordinat tersebut didapat *node* sebanyak 8716 dan *edge* sebanyak 9077.

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.

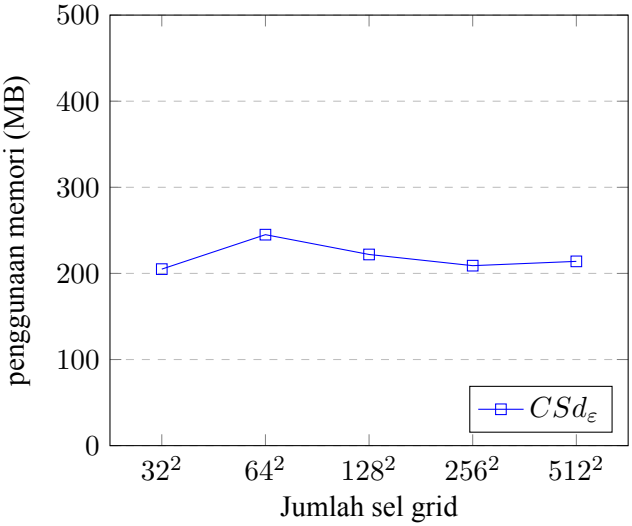
5.3.1.1 Skenario Uji Coba *Performance* Terhadap Perubahan Jumlah Sel

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

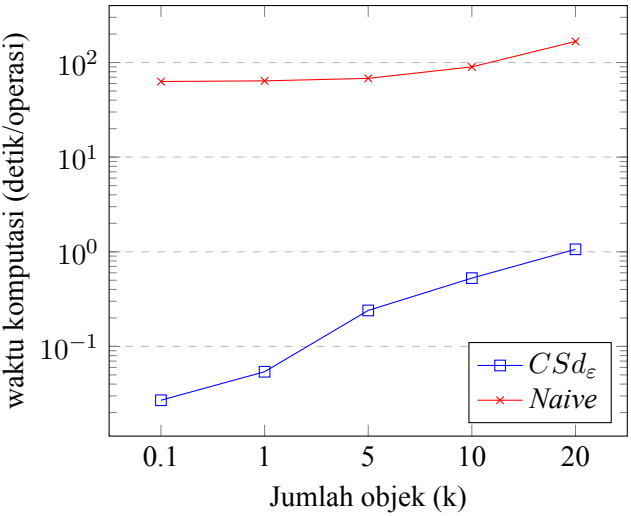
Jumlah sel tidak banyak mempengaruhi penggunaan memori dan waktu komputasi. Hal ini dikarenakan adanya *trade-off* antara proses memuat data dengan komputasi. Grid indeks yang memiliki sel sedikit menjadikan data yang dimuat lebih banyak sehingga menjadikan data yang diproses labih banyak. Tetapi di 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

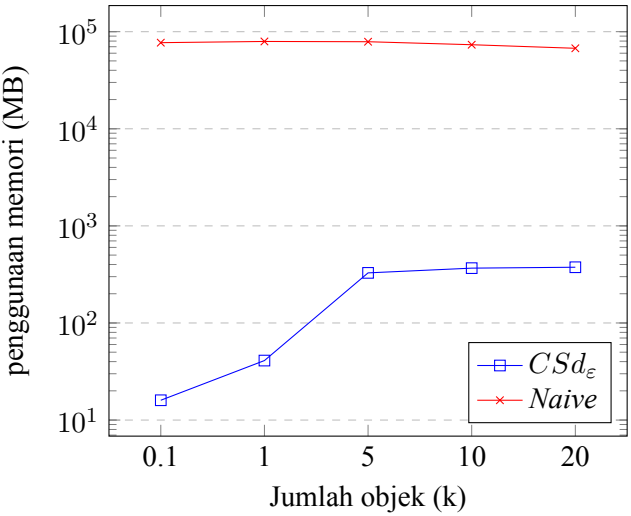
5.3.1.2 Skenario Uji Coba *Performance* Terhadap Perubahan Jumlah Objek

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*.

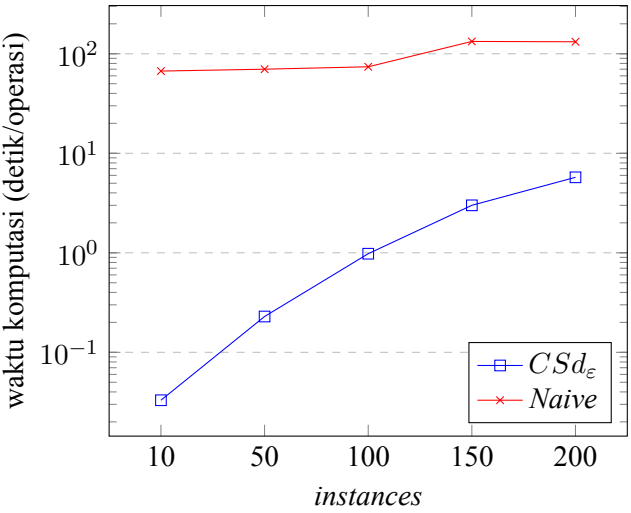


Gambar 5.4 Pengaruh jumlah objek terhadap penggunaan memori dalam satuan megabita

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

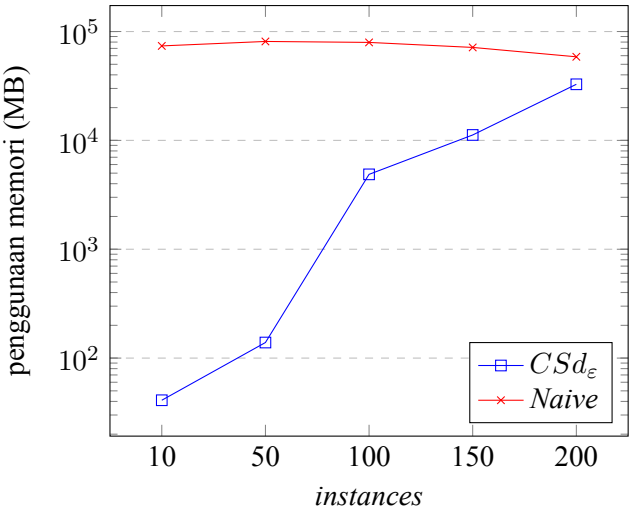
5.3.1.3 Skenario Uji Coba *Performance* Terhadap Perubahan Jumlah *Instance* pada Objek

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

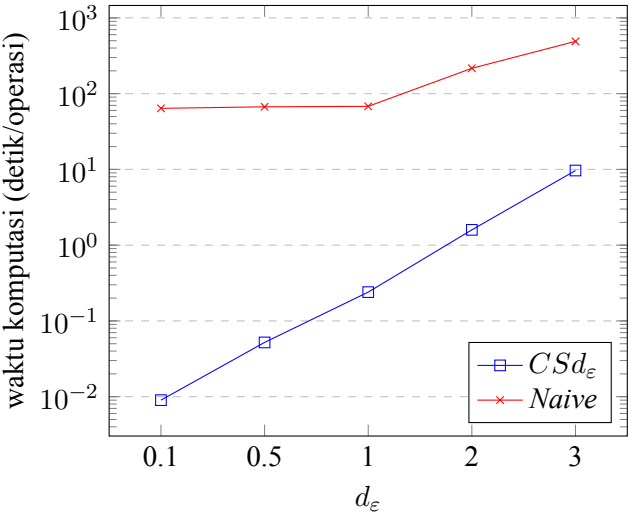
Pada metode *naive*, objek perubahan waktu komputasi terlihat ketika jumlah *instance* diatas 100. Hal ini dikarenakan waktu komputasi lebih banyak digunakan untuk penghitungan jarak terpendek dari setiap *node* ke objek, sehingga jumlah *instance* yang sedikit tidak berpengaruh banyak terhadap waktu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



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

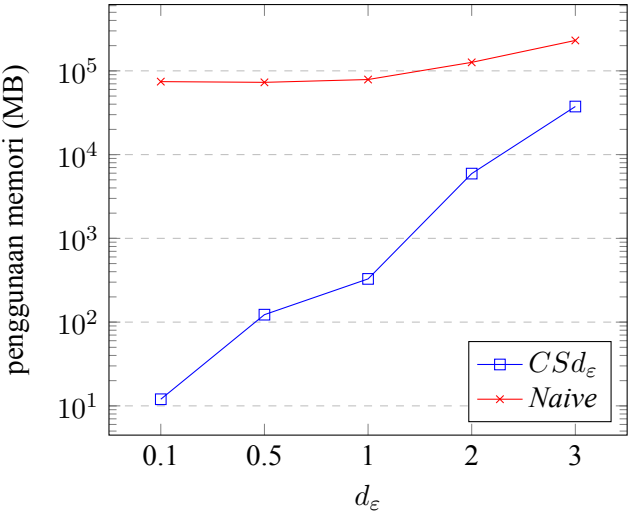
5.3.1.4 Skenario Uji Coba *Performance* Terhadap Perubahan Panjang Jarak Maksimal d_ε

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 penghiungan 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

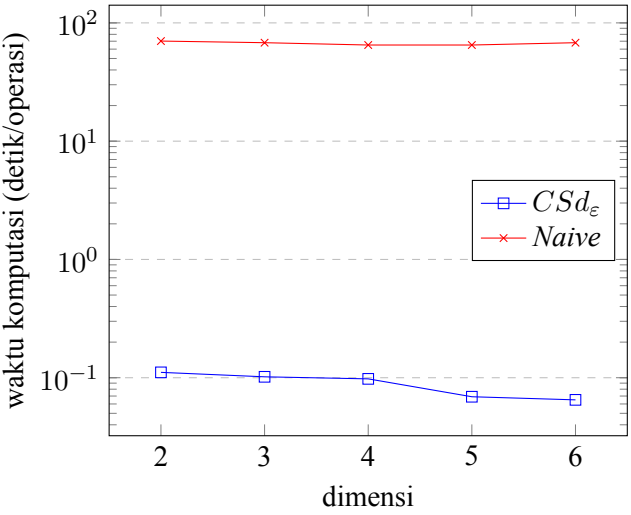
Penggunaan memori sangat tergantung dari jumlah objek yang diproses. Nilai d_ε yang besar menjadikan objek yang diproses semakin banyak karena setiap *node* memiliki jangkauan yang lebih jauh. Banyaknya objek yang diproses menjadikan penggunaan memori semakin besar.



Gambar 5.8 Pengaruh d_ε terhadap penggunaan memori dalam satuan megabita

5.3.1.5 Skenario Uji Coba *Performance* Terhadap Perubahan Dimensi Data

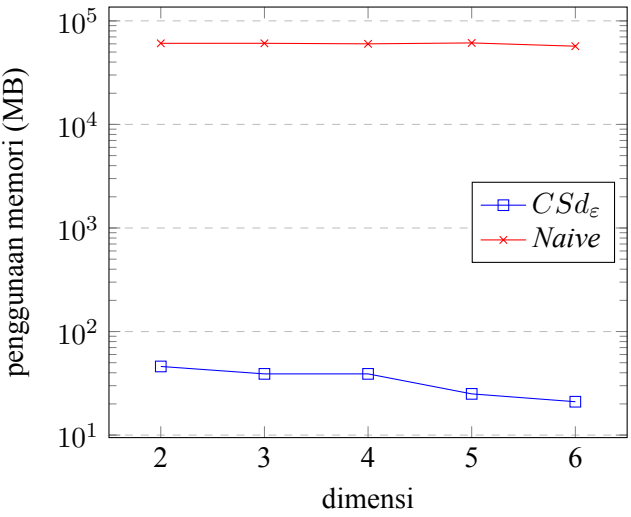
Dimensi dari *uncertian data* tidak banyak mempengaruhi *performance* dari algoritma. Pada Gambar 5.9, waktu komputasi pada algoritma CSd_ε cenderung turun. Hal tersebut dikarenakan antar objek tidak banyak yang mendominasi atau didominasi karena semakin banyaknya variabel yang perlu dibandingkan. Sebagai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

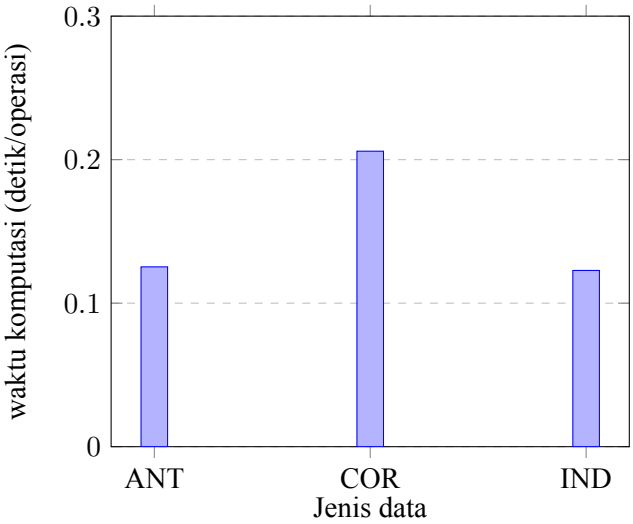
Pengalokasian memori berlaku pada objek-objek yang *overlap*. Semakin sedikit objek yang *overlap*, maka semakin sedikit memori yang dibutuhkan untuk pemrosesan probabilitas *skyline*.



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
PENUTUP

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 algoitma 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] Yuan-Ko Huang, Chia-heng Chang, Chiang Lee, ”Continuous distance-based skyline queries in road networks”, *Information Systems*, vol 37, no 7, pp. 611-633, 2012.

[2] Liu, Chuan-Ming, Tang, Syuan-Wei, ”An effective probabilistic skyline query process on uncetain data streams”, in *The 6th International Conference on Emerging Ubiquitous Systems and Persuasive Networks*, Londok, UK, 2016.

[3] Xiaofeng Ding, Xiang Lian, Lei Chan, Hai Jin, ”Continuous monitoring of skylines over uncertain data streams”, *Information Systems*, vol 184, no 1, pp. 196-214, 2012.

[4] Yijie Wang, Xiaoyong Li, Yuan Wang, ”A Survey of queries over uncertain data”, *Knowledge and Information Systems*, vol 37, no 3, pp 485-530, 2013.

[5] Wenjie Zhang, Xuemin Lin, Jian Pei, Ying Zhang, ”Managing Uncertain Data: Probabilistic Approaches”, *Web-Age Information Management*, 2008

[6] docs.python.org, ”General Python FAQ”, 2018. [Online]. Available: https://docs.python.org/3/faq/general.html, [Accessed: 25 March 2018].

[7] docs.scala-lang.org, ”Tour of Scala”, 2018. [Online]. Available: https://docs.scala-lang.org/tour/tour-of-scala.html, [Accessed: 28 May 2018].

[8] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, Shang-Hua Teng, ”On Trip Planning Queries in Spatial

Databases", *Advances in Spatial and Temporal Databases*, vol 3633, pp 273-290, 2005

[9] developer.mozilla.org, "What is JavaScript?", 2018. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript, [Accessed: 22 June 2018].

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.algorithm
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.addEdges(nodesNodeI, edgesNodeI)
51      addedGrid += gridNodeI
52
53      if (!addedGrid.contains(gridNodeJ)) {
54        val EdgesNodes(edgesNodeJ, nodesNodeJ) = grid.getDataGrid(gridNodeJ)
55        tempGraph.addEdges(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)
```

```
131           }
132       }
133
134       def deleteFromNode(grid: Grid, currentNode: Node, objectId: Int, rect: Rect2d): Node = {
135         val objectMaybe = currentNode.objects.find(_.id == objectId)
136
137         if (objectMaybe.isEmpty) {
138           return currentNode
139         }
140
141         val toBeDeletedPoints = objectMaybe.get.points(grid)
142
143         for (point <- toBeDeletedPoints) {
144           currentNode.tree.remove(point)
145         }
146
147         var overlappedObjects = findPDROverlappedObjects(currentNode, rect)
148
149         overlappedObjects = overlappedObjects.map {
150           case q@Object(_, _, _, true, _, _, _, _) => q
151           case q@Object(_, _, _, _, _, _, _, _) =>
152             val skyProb = SkyPrX(currentNode.tree, q.id)
153             q.updateSkyProb(skyProb)
154         }
155
156         var newObjects = currentNode.objects.map { o =>
157           val updatedObjectMaybe = overlappedObjects.find(_.id == o.id)
158
159           updatedObjectMaybe match {
160             case None => o
161             case Some(obj) => obj
162           }
163         }
164
165         newObjects = newObjects.filterNot(_.id == objectId)
166
167         Node(currentNode.id, currentNode.x, currentNode.y, currentNode.tree, newObjects)
168       }
169
170       def insertToNode(grid: Grid, node: Node,
171                        rawObject: RawObject,
172                        distance: Double,
173                        rect: Rect2d): Node = {
174
175         var overlappedObjects = findPDROverlappedObjects(node, rect)
176
177         rawObject.points.foreach(p => node.tree.add(p))
178
179         val updatedOverlappedObjects = overlappedObjects.map(q => {
180           val ddrRect = rect.getDDR.asInstanceOf[Rect2d]
181           val qRect = q.rect
182
183           if (ddrRect.contains(qRect) & distance < q.distance) {
184             q.asImpossible()
185           } else {
186             val objProb = getDominationProbability(node.tree, ddrRect, q.id)
187             if (objProb > (1 - P_THRESHOLD) & distance < q.distance) {
188               q.asImpossible()
189             } else {
190               val skyProb = SkyPrX(node.tree, q.id)
191               q.updateSkyProb(skyProb)
192             }
193           }
194         })
```

```
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 => PrYnotdominatex(Y, x))
241         .product
242     }
243
244     def PrYnotdominatex(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 = {
```

```
259       val result = new Array[Point2d](N_POINTS)
260
261       tree.search(ddrRect, result)
262
263       result
264           .filter(_.isInstanceOf[Point2d])
265           .filter(_.o == objectId)
266           .foldLeft(0.0)((acc, e) => acc + e.p)
267     }
268
269     def findPDROverlappedObjects(node: Node, rect: Rect2d): Set[Object] = {
270       val tree = node.tree
271       val PDRBox: Rect2d = rect.getPDR.asInstanceOf[Rect2d]
272       val overlappedPoints: Array[Point2d] = new Array[Point2d](N_POINTS)
273       tree.search(PDRBox, overlappedPoints)
274
275       val objectIds = overlappedPoints.toList
276                         .filter(_.isInstanceOf[Point2d]).map(_.o).toSet
277       val objects = objectIds.map(id => {
278         val a = node.objects.find(_.id == id)
279         a.get
280       })
281
282       objects
283     }
284
285     def findPDDOverlappedObjects(node: Node, rect: Rect2d, currentId: Int)
286       : Set[Object] = {
287       val tree = node.tree
288       val pddBox: Rect2d = rect.getPDD.asInstanceOf[Rect2d]
289       val overlappedPoints: Array[Point2d] = new Array[Point2d](N_POINTS)
290       tree.search(pddBox, overlappedPoints)
291
292       val objectIds = overlappedPoints.toList
293                         .filter(_.isInstanceOf[Point2d])
294                         .map(_.o).toSet - currentId
295
296       val objects = objectIds.map(id => {
297         val a = node.objects.find(_.id == id)
298         a.get
299       })
300
301       objects
302     }
303   }
```

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 spNodeS = 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 = spNodeS ++ 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      spNodeS
```

```
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       }
176
177       processLandmark(Q.toList, spNodeS, spNodeE, edge)
178     }
179
180     def findDistanceFromNodeS(obj: Object, edge: Edge, spNodeE: Set[Object]): Double = {
181       if (obj.edgeId == edge.id) {
182         edge.length * obj.position
183       } else {
184         val objInSpNodeEMaybe = spNodeE.find(_.id == obj.id)
185         if (objInSpNodeEMaybe.isDefined) {
186           if (objInSpNodeEMaybe.get.distance < obj.distance) {
187             obj.distance + edge.length
188           } else {
189             obj.distance * -1
190           }
191         } else {
192           obj.distance * -1
193         }
194       }
195     }
196
197     def findDistanceFromNodeE(obj: Object, edge: Edge, spNodeS: Set[Object]): Double = {
198       if (obj.edgeId == edge.id) {
199         edge.length * obj.position
200       } else {
201         val objInSpNodeSMaybe = spNodeS.find(_.id == obj.id)
202         if (objInSpNodeSMaybe.isDefined) {
203           if (objInSpNodeSMaybe.get.distance < obj.distance) {
204             obj.distance * -1
205           } else {
```

```
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       val 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
268                 turningPointList = turningPointList :+ TP(dStart, l.distance, SP)
269
```

```
270                 dStart = l.distance
271                 SP = SP + obj
272               }
273             }
274           case _: LandmarkRight =>
275             if (isObjectInSP(SP, l.objId)) {
276               val obj = SP.find(_.id == l.objId).get
277
278               turningPointList = turningPointList :+ TP(dStart, l.distance, SP)
279
280               dStart = l.distance
281               SP = SP - obj
282             }
283           case landmark: LandmarkLeftMid =>
284
285             if (isObjectInSP(SP, landmark.objId)
286               & isObjectInSP(SP, landmark.objDominatedId)) {
287               val objDominated = SP.find(_.id == landmark.objDominatedId).get
288
289               turningPointList = turningPointList :+ TP(dStart, l.distance, SP)
290
291               dStart = l.distance
292               SP = SP - objDominated
293             }
294           case _ =>
295             val landmark = l.asInstanceOf[LandmarkRightMid]
296
297             val isLandmarkRightMidExist = queue.exists {
298               case a: LandmarkRightMid =>
299                 a.objDominatedId == landmark.objDominatedId
300               case _ =>
301                 false
302             }
303
304             if (isObjectInSP(SP, l.objId) & !isLandmarkRightMidExist) {
305               val objDominated = objects.find(_.id == landmark.objDominatedId).get
306
307               turningPointList = turningPointList :+ TP(dStart, l.distance, SP)
308
309               dStart = l.distance
310               SP = SP + objDominated
311             }
312         }
313       }
314
315       turningPointList = turningPointList :+ TP(dStart, dEnd, SP)
316     }
317   }
```

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()
```

```
23
24      def createDataNaive = {
25        val filename = "n"+N_OBJECTS+"np"+N_POINTS+"g"+N_GRID_CELL+"d"+PERCENT_DISTANCE+"p"+P_THRESHO
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 + "␣" + ro.edgeId + "␣" + ro.position + "␣" + ro.points.size + "␣"
37          )
38          var strp = ""
39          ro.points.foreach { p =>
40            strp += p.x + "␣" + p.y + "␣" + p.p + "␣" + p.o + "␣"
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 + "␣" + n.objects.size + "␣"
57          )
58          var str = ""
59          n.objects.foreach { o =>
60            str += o.id + "␣" + o.distance + "␣" + o.skyProb + "␣" + o.edgeId + "␣" + o.position + "␣
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
79          node
80        }.toList.toSet
81      }
82
83      var tableGrid: Map[Int, Map[Int, Table]] = Map()
84
85      def getRawObject(objectId: Int): Option[RawObject] = rawObjects.par.find(_.id == objectId)
86      def addRawObject(rawObject: RawObject): Unit =
```

```
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
```

```
151             val dx = nodej.x - nodei.x
152             val dy = nodej.y - nodei.y
153             val length = Math.sqrt(dx*dx + dy*dy)
154             Edge(rawEdge.id, rawEdge.i, rawEdge.j, length, Set())
155           }
156       }).toList.toSet
157
158       addEdges(es)
159     }
160
161     // edge
162     def getEdge(edgeId: Int): Option[Edge] = this.edges.par.find(_.id == edgeId)
163
164     def addEdge(edge: Edge): Unit = {
165       this.edges = this.edges + edge
166     }
167
168     def addEdges(edges: Set[Edge]): Unit = {
169       edges.foreach((e: Edge) => {
170         addEdge(e)
171       })
172     }
173
174     def addObjectToEdge(rawObject: RawObject): Unit = {
175       val edgeMaybe = this.edges.par.find(_.id == rawObject.edgeId)
176
177       if (edgeMaybe.isEmpty) {
178         println(rawObject)
179       }
180
181       val edge = edgeMaybe.get
182
183       val newObject = Object(
184         rawObject.id,
185         rawObject.edgeId,
186         100,
187         isImpossible = false,
188         edge.i,
189         createRect(rawObject.points),
190         rawObject.position * edge.length,
191         rawObject.position)
192
193       addObjectToEdge(newObject)
194     }
195
196     def addObjectToEdge(obj: Object): Unit = {
197       val e = this.edges.par.find(_.id == obj.edgeId).get
198       val newEdge = Edge(e.id, e.i, e.j, e.length, e.objects + obj)
199
200       this.edges = this.edges - e + newEdge
201     }
202
203     def removeObjectFromEdge(obj: Object): Unit = {
204       val e = this.edges.par.find(_.id == obj.edgeId).get
205
206       this.edges = this.edges - e + Edge(e.id, e.i, e.j, e.length, e.objects - obj)
207     }
208
209     def removeObjectFromEdge(objectId: Int): Unit = {
210       val o = this.rawObjects.par.find(_.id == objectId).get
211       val e = this.edges.par.find(_.id == o.edgeId).get
212       val deletedObject = e.objects.find(_.id == objectId).get
213
214       this.edges = this.edges - e + Edge(e.id, e.i, e.j, e.length, e.objects - deletedObject)
```

```
215     }
216
217     def getEdges(nodes: List[Node]): List[Edge] = {
218       val nodeIds = nodes.map((n: Node) => n.id)
219
220       this.edges.par.filter((e: Edge) => {
221         nodeIds.contains(e.i) | nodeIds.contains(e.j)
222       }).toList
223     }
224
225     def getGridLocation(node: Node): GridLocation = {
226       val gx = math.floor(node.x / GRID_WIDTH).toInt
227       val gy = math.floor(node.y / GRID_HEIGHT).toInt
228
229       GridLocation(gx, gy)
230     }
231
232     def getGridLocation(x: Double, y: Double): GridLocation = {
233       val gx = math.floor(x / GRID_WIDTH).toInt
234       val gy = math.floor(y / GRID_HEIGHT).toInt
235
236       GridLocation(gx, gy)
237     }
238
239     def getDataGrid(g: GridLocation): EdgesNodes = {
240       val _nodes = getNodes(g)
241       val _nodeIds = _nodes.map(_.id)
242       val edges = getEdges(_nodes)
243       val nodeIds = edges
244         .flatMap(e => Set(e.i, e.j))
245         .filterNot(nid => _nodeIds.contains(nid))
246         .toSet
247
248       val nodes = getNodesFromId(nodeIds).toSet
249
250       EdgesNodes(edges.toSet, _nodes.toSet ++ nodes)
251     }
252   }
```

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   * #L%
```

```
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;
```

```
147             final double minY = Double.MIN_VALUE;
148             final double maxX = min.x;
149             final double maxY = min.y;
150 
151             return new Rect2d(minX, minY, maxX, maxY);
152         }
153 
154         @Override
155         public int getNDim() {
156             return 2;
157         }
158 
159         @Override
160         public HyperPoint getCentroid() {
161             final double dx = min.x + (max.x - min.x)/2.0;
162             final double dy = min.y + (max.y - min.y)/2.0;
163 
164             return new Point2d(dx, dy);
165         }
166 
167         @Override
168         public HyperPoint getMin() {
169             return min;
170         }
171 
172         @Override
173         public HyperPoint getMax() {
174             return max;
175         }
176 
177         @Override
178         public double getRange(final int d) {
179             if(d == 0) {
180                 return max.x - min.x;
181             } else if(d == 1) {
182                 return max.y - min.y;
183             } else {
184                 throw new IllegalArgumentException("Invalid␣dimension");
185             }
186         }
187 
188         @Override
189         public boolean contains(final HyperRect r) {
190             final Rect2d r2 = (Rect2d)r;
191 
192             return min.x <= r2.min.x &&
193                     max.x >= r2.max.x &&
194                     min.y <= r2.min.y &&
195                     max.y >= r2.max.y;
196         }
197 
198         @Override
199         public boolean intersects(final HyperRect r) {
200             final Rect2d r2 = (Rect2d)r;
201 
202             if(min.x > r2.max.x ||
203                     r2.min.x > max.x ||
204                     min.y > r2.max.y ||
205                     r2.min.y > max.y) {
206                 return false;
207             }
208 
209             return true;
210         }
```

```
211
212         @Override
213         public double cost() {
214             final double dx = max.x - min.x;
215             final double dy = max.y - min.y;
216             return Math.abs(dx*dy);
217         }
218
219         @Override
220         public double perimeter() {
221             double p = 0.0;
222             final int nD = this.getNDim();
223             for(int d = 0; d<nD; d++) {
224                 p += 2.0 * this.getRange(d);
225             }
226             return p;
227         }
228
229         @Override
230         public boolean equals(Object o) {
231             if (this == o) return true;
232             if (o == null || getClass() != o.getClass()) return false;
233
234             final Rect2d rect2D = (Rect2d) o;
235
236             return min.equals(rect2D.min) &&
237                     max.equals(rect2D.max);
238         }
239
240         @Override
241         public int hashCode() {
242             return min.hashCode() ^ 31*max.hashCode();
243         }
244
245         public String toString() {
246             final StringBuilder sb = new StringBuilder();
247             sb.append('(');
248             sb.append(Double.toString(min.x));
249             sb.append(',');
250             sb.append(Double.toString(min.y));
251             sb.append(')');
252             sb.append('␣');
253             sb.append('(');
254             sb.append(Double.toString(max.x));
255             sb.append(',');
256             sb.append(Double.toString(max.y));
257             sb.append(')');
258
259             return sb.toString();
260         }
261
262         public final static class Builder implements RectBuilder<Rect2d> {
263
264             @Override
265             public HyperRect getBBox(final Rect2d rect2D) {
266                 return rect2D;
267             }
268
269             @Override
270             public HyperRect getMbr(final HyperPoint p1, final HyperPoint p2) {
271                 return new Rect2d(p1.getCoord(0), p1.getCoord(1),
272                     p2.getCoord(0), p2.getCoord(1));
273             }
274         }
```

```
}
```

Kode Sumber 1.9 Kode sumber kotak dua dimensi

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simulasi</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <div>
      <label for="name">Source Node ID:</label>
      <input type="number" id="src" name="src">
    </div>
    <div>
      <label for="mail">Destination Node ID:</label>
      <input type="number" id="dst" name="dst">
    </div>
    <div class="button">
                <button onclick="findSP()">Find SP</button>
        </div>
  </body>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  <script src="https://d3js.org/d3.v3.min.js"  charset="utf-8"></script>
  <script src="https://d3js.org/d3-ease.v1.min.js"></script>
  <script src='/javascripts/main.js'></script>
</html>
```

Kode Sumber 1.10 Kode sumber visualisasi HMTL

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

let edges = [
  [1, 1, 0],
  [2, 2, 0],
```

```
    [3, 2, 1],
    [4, 3, 1],
    [5, 4, 3],
    [6, 5, 3],
    [7, 5, 4],
    [8, 6, 2],
    [9, 6, 5],
    [10, 7, 0],
    [11, 8, 7],
    [12, 9, 7],
    [13, 9, 8],
    [14, 10, 6],
    [15, 10, 9],
    [16, 11, 10],
    [17, 12, 8],
    [18, 12, 10],
    [19, 13, 11],
    [20, 13, 12],
    [21, 14, 11],
    [22, 15, 13],
    [23, 16, 14],
    [24, 16, 15],
    [25, 17, 15],
    [26, 17, 16],
    [27, 18, 16],
    [28, 18, 17],
    [29, 19, 16],
    [30, 19, 18],
    [31, 20, 4],
    [32, 20, 14],
    [33, 20, 19]
]

let svgWidth = 1100;
let svgHeight = 1100;

let viewWidth = 800;
let viewHeight = 800;

let domainX = [100, 1000]
let domainY = [-200, 500]

var xScale = d3.scale.linear()
                    .domain(domainX)
                    .range([0, viewWidth]);

var xScaleReverse = d3.scale.linear()
                    .domain([0, viewWidth])
                    .range(domainX);

var yScale = d3.scale.linear()
                    .domain(domainY)
                    .range([viewHeight, 0]);

var yScaleReverse = d3.scale.linear()
                    .domain([viewHeight, 0])
                    .range(domainY);

let svg = d3.select('body')
          .append('svg')
          .attr('width', svgWidth)
          .attr('height', svgHeight);

let svgGridX = svg.selectAll('gridline')
```

```
                .data([0, 5, 10, 15, 20])
                .enter()
                .append('line');

svgGridX.attr('x1', function(d) { return xScale(d); })
      .attr('y1', function(d) { return yScale(0); })
      .attr('x2', function(d) { return xScale(d); })
      .attr('y2', function(d) { return yScale(20); })
      .attr('class', 'gridline');

let svgGridY = svg.selectAll('gridline')
                .data([0 , 5, 10, 15, 20])
                .enter()
                .append('line');

svgGridY.attr('x1', function(d) { return xScale(0); })
      .attr('y1', function(d) { return yScale(d); })
      .attr('x2', function(d) { return xScale(20); })
      .attr('y2', function(d) { return yScale(d); })
      .attr('class', 'gridline')

let edgesEnter = svg.selectAll('edges')
                .data(edges)
                .enter();

function findNode(id) {
  for(let i = 0; i < nodes.length; i++) {
    if (nodes[i][0] == id) return nodes[i];
  }

  return null;
}

edgesEnter.append('line')
  .attr('x1', function(d) {
    return xScale(findNode(d[1])[1]); })
  .attr('y1', function(d) {
    return yScale(findNode(d[1])[2]); })
  .attr('x2', function(d) {
    return xScale(findNode(d[2])[1]); })
  .attr('y2', function(d) {
    return yScale(findNode(d[2])[2]); })
  .attr('class', 'edges')

edgesEnter.append('text')
  .attr('x', function(d){
        x = findMiddleEdge(findNode(d[1]),
          findNode(d[2]))[0]
        return xScale(x);
  })
  .attr('y', function(d){
        y = findMiddleEdge(findNode(d[1]),
          findNode(d[2]))[1]
        return yScale(y);
  })
  .attr('class', 'edge-label')
  .text(function(a, i) { return "e" + a[0]; });

function findLocation(edgeId, pos){
  let edge;
  for (let i = 0; i < edges.length; i++) {
    if (edges[i][0] == edgeId) {
      edge = edges[i]
    }
```

```
156     }
157
158     let node1 = findNode(edge[1])
159     let node2 = findNode(edge[2])
160
161     let x = (node2[1] - node1[1]) * pos + node1[1]
162     let y = (node2[2] - node1[2]) * pos + node1[2]
163
164     return [x, y]
165   }
166
167   let nodesEnter = svg.selectAll('nodes')
168                   .data(nodes)
169                   .enter()
170
171   nodesEnter.append('rect')
172         .attr('x', function(d){
173           return xScale(d[1]) - 7.5;
174         })
175         .attr('y', function(d){
176           return yScale(d[2]) - 7.5;
177         })
178         .attr('class', 'nodes')
179         .attr('width', 15)
180         .attr('height', 15);
181
182   nodesEnter.append('text').attr('x', function(d){
183       return xScale(d[1]);
184     })
185     .attr('y', function(d){
186       return yScale(d[2]) + 3; })
187     .attr('class', 'node-label')
188     .attr('text-anchor', 'middle')
189     .text(function(a, i) { return "n" + a[0]; });
190
191   function findMiddleEdge(node1, node2) {
192     x = (node2[1] + node1[1]) / 2;
193     y = (node2[2] + node1[2]) / 2;
194
195     return [x, y];
196   }
197
198   function getNode(i) {
199     return nodes[i].map(x => x * 10);
200   }
201
202   function getEdge(edges, edgeId) {
203     for (let i = 0; i < edges.length; i++) {
204       if (edges[i][0] == edgeId)
205         return edges[i]
206     }
207
208     return null
209   }
210
211   function getNode(nodes, nodeId) {
212     for (let i = 0; i < nodes.length; i++) {
213       if (nodes[i][0] == nodeId)
214         return nodes[i]
215     }
216
217     return null
218   }
219
```

```
220   function getEdgeLength(edgeId) {
221     let edge = getEdge(edges, edgeId)
222     let nodeSId = edge[1]
223     let nodeEId = edge[2]
224     let nodeS = getNode(nodes, nodeSId)
225     let nodeE = getNode(nodes, nodeEId)
226
227     let startX = nodeS[1]
228     let startY = nodeS[2]
229
230     let selisihX = nodeE[1] - nodeS[1]
231     let selisihY = nodeE[2] - nodeS[2]
232
233     return Math.sqrt(selisihX*selisihX + selisihY*selisihY)
234   }
235
236   function findTPCoordinatesByNodeId(nodeSId, nodeEId, TP) {
237     let nodeS = getNode(nodes, nodeSId)
238     let nodeE = getNode(nodes, nodeEId)
239
240     let startX = nodeS[1]
241     let startY = nodeS[2]
242
243     let selisihX = nodeE[1] - nodeS[1]
244     let selisihY = nodeE[2] - nodeS[2]
245
246     let length =
247       Math.sqrt(selisihX*selisihX + selisihY*selisihY)
248
249     return TP.map(function(tp) {
250       let ratio = tp / length
251
252       let posX = startX + selisihX * ratio
253       let x = xScale(posX) - 2
254
255       let posY = startY + selisihY * ratio
256       let y = yScale(posY) - 2
257
258       return {
259         x: x,
260         y: y
261       }
262     })
263   }
264
265   function findTPCoordinatesByEdgeId(edgeId, TP) {
266     let edge = getEdge(edges, edgeId)
267     let nodeSId = edge[1]
268     let nodeEId = edge[2]
269
270     return findTPCoordinatesByNodeId(nodeSId, nodeEId, TP)
271   }
272
273   function addTurningPointMarker(edgeId, TP) {
274     let tps = findTPCoordinatesByEdgeId(edgeId, TP)
275
276     let turningPointEnter = svg.selectAll('turning-point')
277       .data(tps)
278       .enter()
279
280     turningPointEnter.append('rect')
281       .attr('x', d => d.x)
282       .attr('y', d => d.y)
283       .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 svgq = svg.select(selector)
346         .attr('cx', xScale(node[1]))
347         .attr('cy', yScale(node[2]))
```

```
348
349     let SPs = []
350     let TPs = []
351     let reverses = []
352
353     for (let i = 1; i < nodeIds.length; i++) {
354       let nodeId = nodeIds[i]
355       let nodeIdPrev = nodeIds[i-1]
356       let node = getNode(nodes, nodeId)
357       let nodePrev = getNode(nodes, nodeIdPrev)
358       let length = getNodeLength(nodeIdPrev, nodeId)
359
360       let nodePrevID = nodePrev[0]
361       let nodePrevX = nodePrev[1]
362       let nodePrevY = nodePrev[2]
363
364       let nodeNextID = node[0]
365       let nodeNextX = node[1]
366       let nodeNextY = node[2]
367
368       let edge = getEdgeByNodeId(nodeIdPrev, nodeId)
369       let edgeId = edge[0]
370
371       let tick = 0
372       let currentIndex = i
373
374       svgq = svgq.transition()
375         .ease(d3.easeLinear)
376         .duration(length * 50)
377         .attr('cx', xScale(node[1]))
378         .attr('cy', yScale(node[2]))
379         .tween("side-effects", function() {
380           return function() {
381             if (tick % 50 == 0) {
382               let realX = d3.select(this).attr('cx')
383               let realY = d3.select(this).attr('cy')
384
385               let currentX = xScaleReverse(realX)
386               let currentY = yScaleReverse(realY)
387
388               let diffX = Math.abs(currentX) - Math.abs(nodePrevX)
389               let diffY = Math.abs(currentY) - Math.abs(nodePrevY)
390
391               let deltaX = Math.abs(nodeNextX) - Math.abs(nodePrevX)
392               let deltaY = Math.abs(nodeNextY) - Math.abs(nodePrevY)
393
394               let diffXY = Math.sqrt(diffX*diffX + diffY*diffY)
395              let deltaXY = Math.sqrt(deltaX*deltaX + deltaY*deltaY)
396
397               let currentPosition = diffXY / deltaXY
398
399               let position = 0
400               if (nodePrevID == edge[1]) {
401                 position = currentPosition
402               } else {
403                 position = 1 - currentPosition
404               }
405
406               let realLength = length * position
407
408               let maybeSP = getSP(listOfSP, edgeId)
409
410               let isSPFound = false
411               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                 }
428               } else {
429                 console.log([])
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()
```

```
476
477       let objectsEnter = svg.selectAll('objects')
478         .data(objects)
479         .enter()
480
481       objectsEnter
482         .append("circle")
483         .attr('cx', function(d){ return xScale(findLocation(d[1], d[2])[0]); })
484         .attr('cy', function(d){ return yScale(findLocation(d[1], d[2])[1]); })
485         .attr("r", 8)
486         .attr("class", "objects")
487
488       objectsEnter.append("text")
489         .attr('dx', function(d){ return xScale(findLocation(d[1], d[2])[0]); })
490         .attr('dy', function(d){ return yScale(findLocation(d[1], d[2])[1]) + 3; })
491         .attr('text-anchor', 'middle')
492         .attr('class', 'object-label')
493         .append("tspan")
494           .text(function(d) { return "O"; })
495         .append("tspan")
496           .attr("baseline-shift", "sub")
497           .text(function(d) { return d[0]; });
498     })
499     .fail(function() {
500       console.log('failed')
501     })
502
503     setTimeout(function(){
504       getObjects()
505     }, 500);
506   }
507
508   getObjects()
509
510   getData()
511
512   function findSP() {
513     let srcID = document.getElementById('src').value
514     let dstID = document.getElementById('dst').value
515
516     let data = {
517       src: srcID,
518       dst: dstID
519     }
520
521     $.ajax({
522       url: 'http://localhost:8080/getPath?src=' + srcID + '&dst='+ dstID,
523     })
524     .done(function(path) {
525       traverseQueryPoint('#query', path)
526     })
527     .fail(function() {
528       console.log('failed')
529     })
530   }
```

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
6   a {
7     color: #00B7FF;
8   }
9
10  .gridline {
11    stroke: green;
12    stroke-width: 1;
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

Syukron Rifail Muttaqi, lahir di Kabupaten Madiun tanggal 17 Februari 1996. Penulis merupakan anak pertama dari 2 bersaudara. Penulis menyelesaikan pendidikan SMA-nya di SMA Negeri 2 Kota Madiun (2011-2014). Selanjutnya, penulis berkesempatan mengenyam program studi kuliah sarjana di Departemen Informatika ITS.

Pada dua tahun pertama selama kuliah di ITS, penulis aktif menjadi anggota Unit Kegiatan Mahasiswa Merpati Putih dan Koperasi Mahasiswa dr. Angka. Pada tahun selanjutnya, penulis menjadi asisten mata kuliah Sistem Operasi dan Jaringan Komputer selama dua tahun berturut-turut. Selama menjalani pendidikan S1, penulis menjadi asisten laboratorium Arsitektur dan Jaringan Komputer. Selain berminat di bidang Jaringan, penulis juga menyukai bidang Rekayasa Perangkat Lunak. Penulis juga ikut bergabung di beberapa proyek perangkat lunak semasa kuliah. Penulis dapat dihubungi melalui surel di alamat *syukronrifai[di]gmail[titik]com*.