TESIS – IF 185401

CONTINUOUS MULTIQUERIES K-DOMINANT SKYLINE DI JARINGAN JALAN RAYA

SYUKRON RIFAIL MUTTAQI
05111850010021

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

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
2020

(*Halaman ini sengaja dikosongkan*)

THESIS – IF 185401

CONTINUOUS MULTIQUERIES K-DOMINANT SKYLINE DI JARINGAN JALAN RAYA

SYUKRON RIFAIL MUTTAQI
05111850010021

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

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
2020

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN TESIS

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M. Kom)
di
Institut Teknologi Sepuluh Nopember
Oleh:
SYUKRON RIFA'IL MUTTAQI
NRP: 05111850010021

Tanggal Ujian: 14 Juli 2020
Periode Wisuda: September 2020

Disetujui oleh:
Pembimbing:

1. Bagus Jati Santoso, S. Kom., Ph. D. ……
   NIP. 19770824 200604 1 001

Penguji:

1. Waskitho Wibisono, S.Kom., M.Eng., Ph.D. ……
   NIP. 197410222000031001
2. Dr. Eng. Radityo Anggoro, S. Kom., M. Sc. ……
   NIP. 19841016 200812 1 002
3. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D. ……
   NIP. 19810620 200501 1 003

Kepala Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas

Dr. Eng. Chastine Fatichah, S.Kom., M.Kom
NIP. 19751220 200112 2 002

Halaman ini sengaja dikosongkan

CONTINUOUS MULTIQUERIES k-DOMINANT SKYLINE DI JARINGAN JALAN RAYA

Nama mahasiswa	:	Syukron Rifa'il Muttaqi
NRP	:	0511185001000021
Pembimbing	:	Bagus Jati Santoso, S.Kom., Ph.D.

ABSTRAK

Meningkatnya penggunaan perangkat *mobile* menjadikan data spasial patut untuk dipertimbangkan. Untuk mendapatkan hasil yang maksimal, pengguna seringkali mencari yang terbaik dari sekumpulan objek. Diantara algoritma yang dapat digunakan adalah *skyline query*. Algoritma tersebut mencari semua objek yang tidak didominasi oleh objek lain pada semua atributnya. Namun, data yang memiliki banyak atribut menjadikan query tersebut mengeluarkan hasil objek yang banyak sehingga kurang bermanfaat bagi pengguna. k-dominant *skyline query* dapat menjadi solusi untuk mengurangi objek yang menjadi hasil. Diantara hal yang menjadi tantangan adalah penggunaan *skyline query* dengan data spasial dan banyaknya preferensi pengguna dalam mencari objek terbaik.

Penelitian ini mengusulkan IKSR, algoritma dan struktur data yang digunakan untuk memproses *k-dominant skyline* di jaringan jalan raya. Algoritma ini bersifat *continuous* sehingga setiap jalan raya dapat diketahui *k-dominant skyline*-nya. Algoritma bersifat *multiquery*, yaitu dapat memproses banyak *query* dalam sekali komputasi pada dimensi yang sama. IKSR dapat mempercepat komputasi dua hingga tiga kali lipat dibandingkan algoritma *naive*.

Kata kunci: skyline, k-dominant, jaringan jalan raya

Halaman ini sengaja dikosongkan

CONTINUOUS MULTIQUERIES *k*-DOMINANT SKYLINE ON ROAD NETWORK

By	:	Syukron Rifa'il Muttaqi
Student Identity Number	:	0511185001000021
Supervisor(s)	:	Bagus Jati Santoso, S.Kom., Ph.D.

ABSTRACT

The increasing use of mobile devices makes spatial data worthy of consideration. To get maximum results, users often look for the best from a collection of objects. Among the algorithms that can be used is the skyline query. Skyline query looks for all objects that are not dominated by other objects in all of its attributes. However, data that has many attributes makes the query produce a lot of objects so it is less useful for the user. *k*-dominant skyline queries can be a solution to reduce the object that results. Among the challenges is the use of skyline queries with spatial data and the many preferences of users in finding the best object.

This thesis proposes IKSR, an algorithm and its data structure used to process *k*-dominant skyline on the road network. IKDSR compute continuously so that *k*-dominant skyline of every road is computed. The algorithm is multiquery, which can process multiple queries in one computation on the same dimension. IKDSR speeds up computing two to three times compared to the naive algorithm.

Key words : skyline, *k*-dominant, road network

Halaman ini sengaja dikosongkan

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 :

***CONTINUOUS MULTIQUERIES K-DOMINANT SKYLINE* DI 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. Ibu, Bapak, Adik, dan segenap keluarga yang telah mendukung dan memotivasi seumur hidup penulis.
2. Bapak Bagus Jati Santoso, S.Kom., Ph.D. sebagai pembimbing selama pendidikan S1 dan S2, dari persiapan hingga selesainya penulisan buku ini.
3. Teman-teman Lab Arsitektur dan Jaringan Komputer yang telah menemani penulis pagi siang malam selama menempuh S1 dan S2 di Surabaya.
4. Teman-teman S2 Informatika angkatan 2018 serta teman-teman S1 Informatika 2014.
5. Teman-teman kantor.
6. Dan teman-teman yang telah membantu dan menemani yang tidak bisa saya sebutkan satu per satu namanya.

Surabaya, Juli 2020

Syukron Rifa'il Muttaqi

Halaman ini sengaja dikosongkan

DAFTAR ISI

HALAMAN JUDUL i

COVER PAGE iii

LEMBAR PENGESAHAN v

ABSTRAK vii

ABSTRACT ix

KATA PENGANTAR xi

DAFTAR ISI xiii

DAFTAR GAMBAR xv

DAFTAR TABEL xvii

1 PENDAHULUAN 1

1.1 Latar Belakang 1

1.2 Rumusan Masalah 3

1.3 Tujuan 3

1.4 Manfaat 3

1.5 Kontribusi 3

1.6 Batasan Masalah 4

2 KAJIAN PUSTAKA 7

2.1 *Skyline* 7

2.2 *Skyline query* pada Jaringan Jalan Raya 8

2.3 *Continuous Skyline Queries* pada Jaringan Jalan Raya 9

2.4 *k-Dominant Skyline Query* 11

2.5 *Multiqueries k-dominant Skyline* 12

2.6 Diagram Voronoi 13

2.7 *k-Dominant Skyline* pada Lingkungan Paralel 14

3 METODE PENELITIAN 17

3.1 Metodologi Penelitian 17

3.2 Algoritma dan Struktur Data 18
3.2.1 Definisi Permasalahan 18
3.3 *Index-Based k-Dominant Skyline Query* di Jaringan Jalan raya (IKSR) 20
3.3.1 Analisis 21
3.3.2 Struktur Data dan Algoritma 26
3.4 Pengujian dan Evaluasi 38
3.5 Algoritma *Naive* 39

4 HASIL DAN PEMBAHASAN 43
4.1 Lingkungan Pengujian 43
4.2 Jenis Data 43
4.2.1 Independent 44
4.2.2 Correlated 44
4.2.3 Anti-Correlated 44
4.3 Analisis Hasil 45
4.3.1 Pengaruh Jumlah Objek Terhadap Waktu Komputasi 45
4.3.2 Pengaruh Jarak Maksimal d_ε Terhadap Waktu Komputasi 46
4.3.3 Pengaruh Jumlah Atribut Terhadap Waktu Komputasi 47
4.3.4 Pengaruh Jenis Data Terhadap Waktu Komputasi 48
4.3.5 Kesimpulan Analisis 48

5 KESIMPULAN DAN SARAN 51
5.1 Kesimpulan 51
5.2 Saran 51

DAFTAR PUSTAKA 53

DAFTAR GAMBAR

Gambar 2.1 *Skyline* hotel 7
Gambar 2.2 Contoh *skyline* pada jaringan jalan raya 9
Gambar 2.3 Diagram voronoi dengan 20 *centroid*. 13
Gambar 2.4 Lima *controid* pada voronoi di jaringan jalan raya. 14

Gambar 3.1 Penggabungan *query* q_1 dan q_2 21
Gambar 3.2 DSS dari objek o_2 dari $k = 4$ ke $k = 5$ 24
Gambar 3.3 Visualisasi BTree pada *edge* 26
Gambar 3.4 Alur konstruksi *k-dominant skyline* 27
Gambar 3.5 Algoritma *construction*: Konstruksi awal *k-dominant skyline*. 28
Gambar 3.6 Alur keluarnya objek/*insertion* 31
Gambar 3.7 Algoritma *insertion*: masuknya objek o pada jaringan jalan raya. 32
Gambar 3.8 Alur keluarnya objek/*deletion* 35
Gambar 3.9 Algoritma *deletion*: objek o keluar dari sistem. 36
Gambar 3.10 Peta jaringan jalan raya California (Li et al., 2005) 38
Gambar 3.11 Algoritma *construction* untuk metode *naive*. 40
Gambar 3.12 Algoritma *insertion* untuk metode *naive* 41
Gambar 3.13 Algoritma *deletion*: objek o keluar dari sistem. 41

Gambar 4.1 Grafik pengaruh jumlah objek terhadap waktu komputasi dalam *miliseconds* 45
Gambar 4.2 Grafik pengaruh jarak maksimal/d_ε terhadap waktu komputasi. ... 46
Gambar 4.3 Grafik pengaruh jumlah atribut terhadap waktu komputasi. 47
Gambar 4.4 Grafik pengaruh jumlah atribut terhadap waktu komputasi. 48

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Tabel 3.1 Simbol dan keterangannya 18
Tabel 3.2 Contoh dataset **D**. 19
Tabel 3.3 Contoh dataset dengan *cyclic dominance relationship*. 20
Tabel 3.4 Skenario pengujian. 39

Tabel 4.1 Skenario pengujian. 43

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Meningkatnya perangkat yang *mobile* pada beberapa tahun terakhir menjadikan penggiat bisnis dan teknologi semakin memperhitungkan data spasial untuk memaksimalkan proses bisnis. Perangkat *mobile* tersebut dapat ditemukan setiap saat dan dimanapun, seperti ponsel cerdas dan kendaraan yang sudah mengintegrasikan lokasi pada peta. Dengan demikian, perangkat-perangkat tersebut akan semakin banyak memproduksi dan menggunakan data spasial. Tidak hanya itu, secara umum data yang diproduksi dan yang digunakan oleh komputer di dunia juga semakin meningkat. Diantara tantangan yang perlu diselesaikan oleh ilmuwan adalah meningkatkan kecepatan proses ekstraksi informasi dari suatu data.

Seringkali seseorang perlu mencari objek terunggul dari sekumpulan data. Diantara algoritma yang dapat digunakan adalah *skyline query*. *Skyline query* merupakan metode pencarian yang menghasilkan objek yang tidak didominasi oleh objek lain. Dalam kata lain, objek-objek *skyline* merupakan objek yang lebih baik dari objek lain dan tidak lebih buruk dibandingkan objek yang lain pada semua atributnya. Sebagai contoh, seseorang ingin mencari hotel dengan kriteria rating yang paling tinggi dan harga yang paling rendah. Seringkali hasil dari *skyline query* lebih dari satu karena memungkinkan antar-objek tidak saling mendominasi. Sebagai contoh, hotel A memiliki rating 4,5 dengan biaya Rp400.000,00 dan hotel B memiliki rating 3 dengan biaya Rp200.000,00. Dalam hal ini, hotel A dan B tidak saling mendominasi karena tiap objek memiliki keunggulan sendiri-sendiri. Jika hotel A memiliki rating 2, maka hotel B mendominasi hotel A karena hotel B memiliki rating yang lebih tinggi dengan harga yang lebih rendah.

Dalam kehidupan sehari-hari, banyak kriteria yang dipertimbangkan untuk menentukan objek yang paling unggul. Sebagai contoh, untuk menentukan hotel yang unggul, seseorang menggunakan kriteria harga, biaya, kenyamanan, dan

keamanan. Semakin banyak atribut yang terdapat pada objek maka semakin banyak kemungkinan objek yang menjadi bagian dari *skyline*. Hal tersebut dikarenakan probabilitas satu objek mendominasi objek lain semakin kecil karena semakin banyaknya atribut yang menjadi kriteria.

Diantara metode untuk mengurangi jumlah objek yang menjadi *skyline* adalah *k-dominant skyline query*. *k-dominant skyline* merupakan pencarian objek *skyline* dengan kriteria suatu objek dapat menjadi *skyline* apabila terdapat *k* atribut yang sama dengan dengan *k* atribut yang sama pada semua objek lain dan terdapat minimal satu atribut yang lebih baik dari yang lain.

Di abad ke-21 ini, banyak informasi yang mengandalkan data spasial. Berbagai layanan melibatkan data spasial secara *real-time*. Sebagai contoh, layanan ojek daring dan aplikasi reservasi penginapan sangat mengandalkan informasi pengguna dan mitra berdasarkan letak geografisnya. Secara alami, permasalahan tersebut membutuhkan metode yang dapat mengkomputasi secara *real-time*. Di sisi lain, proses pengkomputasian *continuous* dapat menjadi solusi dari permasalahan tersebut yang pada umumnya banyak pengguna yang mengakses layanan umum.

Sebagaimana *skyline query* pada umumnya, *k-dominant query* dapat menggunakan jaringan jalan raya sebagai salah satu pertimbangan dalam mencari objek *skyline*. Diantara permasalahan yang terdapat pada jaringan jalan raya adalah pengguna atau titik *query* mungkin terdapat di berbagai tempat sehingga membutuhkan pemrosesan data yang dinamis sesuai lokasi dari pengguna. Hal tersebut menjadikan komputasi semakin berat karena terdapat banyak titik *query*. Pemrosesan secara *continuous* dapat menjadi solusi agar tidak terjadi proses komputasi yang berulang.

Secara umum, perilaku pengguna seringkali berbeda-beda. Pengguna memiliki preferensi sendiri dalam menentukan kriteria apa yang dapat dijadikan pertimbangan dalam mencari objek yang dianggap unggul. Dengan demikian, pemrosesan yang dapat mengkomputasi berbagai *query* dalam satu waktu diperlukan agar lebih efisien. Tesis ini mengusulkan metode *multiqueries*, yaitu

pemrosesan dapat menggunakan beberapa atribut sesuai preferensi tanpa harus melibatkan semua atribut dalam pemrosesannya.

1.2 Rumusan Masalah

Permasalahan yang akan diselesaikan pada tesis ini adalah sebagai berikut:

1. Bagaimana struktur data dan algoritma untuk *continuous multiqueries k-dominant skyline* pada jaringan jalan raya?
2. Bagaimana cara mengetahui biaya komputasi *continuous multiqueries k-dominant skyline* pada jaringan jalan raya?

1.3 Tujuan

Tujuan tesis ini adalah sebagai berikut:

1. Mendesain struktur data dan algoritma untuk *continuous multiqueries k-dominant skyline* pada jaringan jalan raya.
2. Mengetahui biaya komputasi dari *continuous multiqueries k-dominant skyline* pada jaringan jalan raya.

1.4 Manfaat

Metode pada tesis ini dapat digunakan untuk mencari objek yang *menarik* pada suatu set data yang besar. Objek yang dimaksud adalah objek yang lebih baik dari objek yang lain dalam *k* atribut tertentu. Metode ini dapat memperkecil jumlah objek yang dihasilkan *query skyline* pada umumnya sehingga mempermudah pengguna dalam mencari objek yang diinginkan.

1.5 Kontribusi

Hingga tesis ini ditulis, belum terdapat penelitian mengenai pemrosesan kueri *k-dominant skyline* yang mengkhususkan jaringan jalan raya dengan kapabilitas *multiqueries*. Secara garis besar, kontribusi penulis pada tesis ini adalah:

1. Penulis mengusulkan algoritma IKSR untuk mencari objek yang dominan dengan metode *k-dominant* pada jaringan jalan raya dengan kapabilitas *multiqueries* dan bersifat *continuous*.

2. Penulis menyusun struktur data yang sesuai untuk menyimpan dan memelihara data.

Proses komputasi pada algoritma ini bersifat *continuous*, artinya setiap ada perubahan pada data set berupa masuknya atau keluarnya objek pada jaringan jalan raya mengakibatkan semua *edge* yang terkena pengaruh dihitung *skyline*-nya. Algoritma ini tidak memerlukan *input* berupa titik *query* karena algoritma menghitung setiap titik di jaringan jalan raya. Dengan demikian, banyak sedikitnya titik *query* yang diminta pengguna tidak mempengaruhi kinerja algoritma.

1.6 Batasan Masalah

Masalah yang akan diselesaikan memiliki batasan-batasan berikut:

1. Atribut pada objek memiliki jumlah dan nilai yang tetap atau tidak berubah-ubah.

2. Setiap objek berada suatu titik di *edge* atau jalan raya.

3. Tidak ada objek yang berada di tempat/koordinat yang sama persis.

4. Setiap *edge* memiliki ujung *node* n_s dan n_t.

5. Objek dapat berpindah, masuk, atau keluar dari sistem sewaktu-waktu tanpa diketahui oleh sistem sebelumnya. Artinya, data bersifat *streaming*.

6. Terdapat banyak titik *query* dan dapat berpindah sewaktu-waktu.

7. Algoritma dapat melakukan komputasi satu atau banyak *query* (*multiquery*) dan tiap *query* dapat memiliki himpunan dimensi **D'** serta nilai *k* yang berbeda.

8. Jarak maksimal d_ε didefinisikan untuk membatasi jarak terjauh sebuah objek menjadi *skyline*.

Adanya jarak maksimal/d_ε perlu didefinisikan agar proses komputasi tidak menggunakan semua jaringan jalan raya karena pada kasus nyata seringkali objek menjadi semakin tidak menarik apabila berada di luar area yang dapat dijangkau oleh pengguna.

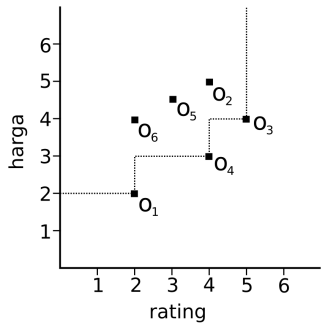
Halaman ini sengaja dikosongkan

BAB 2

KAJIAN PUSTAKA

2.1 *Skyline*

Operator *Skyline* kali pertama dicetuskan oleh Börzsönyi dkk. (Börzsönyi et al., 2001). Operator *Skyline* memfilter objek-objek yang menarik dari data set yang besar. Dalam definisi lain, Operator *Skyline* mengeluarkan semua objek yang tidak didominasi oleh objek lainnya. Sebagai ilustrasi, seseorang bepergian ke Surabaya dan ingin mencari hotel yang menarik, dalam artian, hotel tersebut memiliki harga yang paling rendah dengan rating yang paling bagus. Hasil dari pencarian orang tersebut tentunya memiliki banyak hasil, terdapat hotel yang ratingnya tinggi namun harganya juga tinggi dan terdapat hotel yang ratingnya rendah namun harganya murah. Sebagai contoh, pada Gambar 2.1, *skyline* dari hotel-hotel tersebut adalah o_1, o_3, dan o_4 karena tidak terdapat hotel yang lebih baik dari ketiga hotel tersebut.



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 *Skyline* hotel

Hotel o_1 mendominasi o_6 karena memiliki rating dan harga yang lebih baik. Hotel o_1 dikatakan tidak didominasi dan tidak mendominasi o_4 karena o_1 memiliki harga yang lebih baik dan o_4 memiliki rating yang lebih baik. Demikian juga o_1 tidak didominasi dan tidak mendominasi o_3 karena alasan yang sama.

2.2 *Skyline query* pada Jaringan Jalan Raya

Skyline query tradisional seperti pada Gambar 2.1 hanya menggunakan atribut statis sebagai penentuan *Skyline Points*(*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 *skyline points*. Atribut jarak tersebut adalah atribut spasial yang dinamis yang nilainya berbeda-beda tergantung pada letak dari titik *query*.

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(dist(q, o_5)) tidak didominasi oleh o_1(dist(q, o_1)), o_2(dist(q, o_2)) maupun o_3(dist(q, o_3)).

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

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.

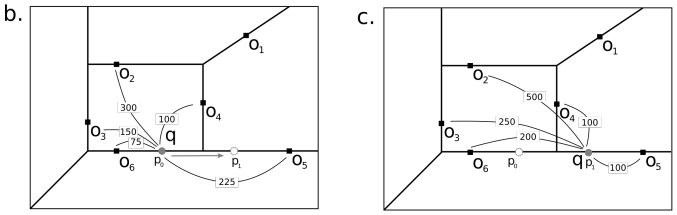
Huang dkk. (Huang et al., 2012) telah mengusulkan metode untuk menyelesaikan permasalahan ini dengan metode *continuous* d_ε*-skyline query* ($Cd_\varepsilon - SQ$). Metode ini menggunakan jarak d_ε sebagai batas maksimal jarak antara titik *query* dengan objek.

$Cd_\varepsilon - SQ$ 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 $d_\varepsilon - SP$.

a.

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

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_ε-SP pada titik kueri *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_ε-SP hanyalah o_2. o_4 tidak dapat menjadi d_ε-SP karena didominasi oleh o_2 dalam hal atribut statis(harga dan rating) maupun dinamis(jarak). o_1 dan o_3 tidak tergabung dalam d_ε-SP karena memiliki jarak lebih dari d_ε. Selanjutnya, seperti yang terlihat pada Gambar 2.3(c), titik kueri q bergerak ke kiri sampai tepat di titik p_2 sehingga jarak antara titik kueri q dengan o_2 dan o_4 menjadi sama. Ketika titik kueri 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_ε-SP-nya adalah o_2 dan o_4.

Dalam menentukan d_ε-SP, yang pertama dilakukan adalah menghitung jarak terdekat antar dua objek. Penghitungan tersebut dapat dilakukan dengan algoritma Dijkstra atau A*. Perlu digarisbawahi bahwa 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 (Huang et al., 2012).

2.3 *Continuous Skyline Queries* pada Jaringan Jalan Raya

Pada domain *data stream*, data dapat masuk dan keluar dari dalam sistem sewaktu-waktu sehingga algoritma yang bersifat *continuous* diperlukan agar

proses komputasi tidak perlu dilakukan berulang-ulang dan menggunakan banyak sumber daya. Dengan algoritma yang *continuous*, sistem dapat melaporkan hasil pemrosesan secara langsung dengan waktu yang cepat dan menggunakan sumber daya yang minimal. Beberapa algoritma telah diusulkan untuk memproses *data stream* pada jaringan jalan raya.

Huang dkk. (Huang et al., 2012) mengusulkan algoritma $Cd_\varepsilon - SQ$ sebagaimana pada Sub Bab 2.2 dan algoritma $Cknn - SQ$. Berbeda dari $Cd_\varepsilon - SQ$ yang mengeluarkan objek dengan jarak sebagai *threshold*, $Cknn - SQ$ mengeluarkan objek minimal sejumlah n objek terdekat. Kedua algoritma tersebut dapat menggunakan *eucledian* atau *road distance* untuk proses penghitungan jarak antara titik *query* dengan objek. Grid indeks juga digunakan untuk mengurangi komputasi objek yang tidak diperlukan dengan membagi area menjadi grid dengan ukuran $m \times n$. Sehingga proses komputasi *insertion* atau *deletion* hanya melibatkan objek dari grid yang diperlukan. Untuk menentukan *skyline points* yang berada pada suatu *edge*, konsep *landmark* digunakan untuk menentukan batas maksimal suatu objek menjadi *skyline*.

Pada (Fu et al., 2017), Fu dkk. mengusulkan algoritma *landmark-based* dan *index-based*. Kedua algoritma tersebut digunakan untuk menguji objek yang bergerak secara dinamis. Algoritma *landmark-based* yang terpengaruh dari (Huang et al., 2012) menggunakan konsep *Global Skyline Point*(*GSP*) pada setiap *edge* dan mencari *landmark* dari *skyline point* yang merupakan anggota dari *GSP*. Algoritma *index-based* menggunakan konsep *skyline scope* dengan memanfaatkan ide voronoi diagram. *Index-based* juga menggunakan B+-Tree untuk penyimpanan *skyline points* pada setiap *edge*.

Jika diterapkan pada objek yang bergerak, algoritma *landmark-based* membutuhkan komputasi yang besar karena harus melakukan komputasi ulang semua *edge* yang terkena pengaruh serta objek-objek yang berada di dalamnya. Berbeda dari *landmark-based*, *index-based* hanya menghitung ulang objek lain yang terdominasi saja ketika terdapat perubahan pada objek. Hal tersebut dapat dilakukan karena pada dasarnya objek yang awalnya terdominasi berpotensi

menjadi *skyline* jika terdapat perubahan atribut pada objek yang mendominasinya. Di sisi lain, objek lain yang sudah menjadi *skyline* tidak perlu dilibatkan dalam komputasi karena dia tetap menjadi *skyline* selama tidak ada yang mendominasi. Hal tersebut menjadikan algoritma *index-based* lebih superior dibandingkan *landmark-based* dalam objek yang dinamis. Metode pada tesis ini mengadaptasi *index-based* agar proses komputasi lebih efisien ketika berhubungan dengan *data stream*.

2.4 *k-Dominant Skyline Query*

Pada konsep dominansi *skyline* normal, sebuah objek mendominasi objek lain jika semua atributnya sama dan terdapat minimal satu atribut yang lebih baik. Semakin banyak atribut yang digunakan maka semakin kecil kemungkinan objek satu mendominasi objek lain dan mengakibatkan objek yang menjadi *skyline point* semakin banyak (Börzsönyi et al., 2001). Hal tersebut menjadikan keluaran dari *skyline query* kurang bermanfaat bagi pengguna jika data set memiliki atribut yang banyak.

Untuk mendapatkan objek yang lebih berguna, algoritma perlu mengeluarkan hasil yang lebih sedikit agar pengguna mudah dalam memilih objek yang diinginkan. Chan dkk. (Chan et al., 2006) mengusulkan algoritma *k-dominant skyline query* yang mengeluarkan objek yang dominan pada k atribut, dengan $k < d$, sehingga kemungkinan objek mendominasi objek lain semakin besar. Dengan demikian, algoritma tersebut mengeluarkan hasil objek yang lebih sedikit dibandingkan *skyline query*.

Chan dkk. (Chan et al., 2006) mengusulkan tiga algoritma, yaitu OSA, TSA, dan SRA. OSA menyimpan *non-k-dominant skyline points* agar tidak dapat *false-positive* karena bisa jadi terdapat objek di *k-dominant skyline* yang tidak didominasi oleh objek lain di *k-dominant skyline* itu sendiri. TSA tidak melakukan pemeliharaan *non-k-dominant skyline*, namun kandidat dari *k-dominant skyline* dibandingkan ulang dengan semua data untuk menghilangkan adanya *false-positive* sebagaimana permasalahan yang terdapat pada OSA. Sedangkan SRA menggunakan pendekatan stuktur data yang mengurutkan masing-masing

atribut sesuai nilai terbaiknya lalu memproses masing-masing atribut disertai *pruning* satu per satu objek hingga habis. SRA lebih stabil seiring dengan bertambahnya nilai *k* dibandingkan kedua algoritma yang lain karena proses *pruning* dilakukan lebih awal dan tidak memerlukan mekanisme pemeliharaan *non-k-dominant skyline* ataupun kandidat *k-dominant skyline*. Ketiga algoritma tersebut dapat diujikan pada data statis.

2.5 *Multiqueries k-dominant Skyline*

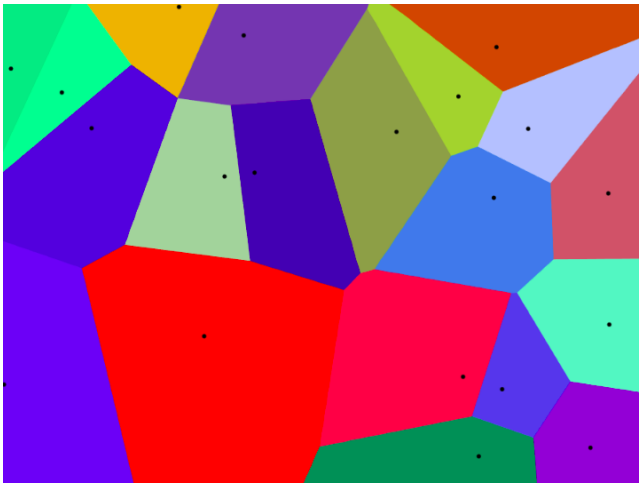
Kontaki dkk. pada (Kontaki et al., 2008) mengusulkan sebuah metode untuk memproses *k-dominant skyline* dengan kapabilitas *multiqueries*. *Multiqueries* pada penelitian tersebut adalah variasi dari nilai *k* dan penggunaan dimensi-dimensi sesuai keinginan pengguna. Sebagai contoh, pengguna A menggunakan nilai k = 4 dan dimensi $\{a, b, c, d, e, f\}$ dan pengguna B menggunakan nilai k = 3 dengan dimensi $\{b, c, d, f, g, h\}$. Kedua *query* tersebut dapat dijalankan dengan satu proses.

Ide dari Kontaki dkk. (Kontaki et al., 2008) adalah pembuatan struktur data yang akan digunakan pada setiap *query*. Penulis mengistilahkan hal tersebut dengan *reusable data*. Dengan demikian, setiap *query* tidak perlu menghitung ulang dari awal dan hanya perlu melanjutkan proses komputasi pada titik tertentu untuk mendapatkan hasil sesuai nilai *k* dan dimensi-dimensinya. Penulis mengadaptasi konsep *reusable data* tersebut untuk diterapkan pada jaringan jalan raya pada algoritma IKSR.

Secara detail, Kontaki dkk. (Kontaki et al., 2008) menggunakan banyak struktur data grid sebagai *reusable data*. Grid-grid tersebut diproses ketika terdapat objek yang berubah dalam artian objek masuk atau keluar dari data set. Untuk memproses satu *query*, algoritma hanya perlu mengambil grid yang dibutuhkan sesuai dimensi yang digunakan pada *query*. Semakin sedikit dimensinya, semakin kecil grid yang digunakan. Berdasarkan pengujian, proses penghitungan *multiqueries* tersebut dapat berjalan dengan lebih efisien dibandingkan TSA dan SRA (Chan et al., 2006) ketika *query* berjumlah 100.

2.6 Diagram Voronoi

Diagram voronoi merupakan pembagian suatu ruang berdasarkan objek *centroid* terdekat yang terdapat pada ruang tersebut. Diagram voronoi memiliki berbagai jenis dalam mendefinisikan objek terdekat. Sebagai contoh, Gambar 2.3 menampilkan daerah dengan warna sesuai dengan objek terdekat dalam jarak eucledian.

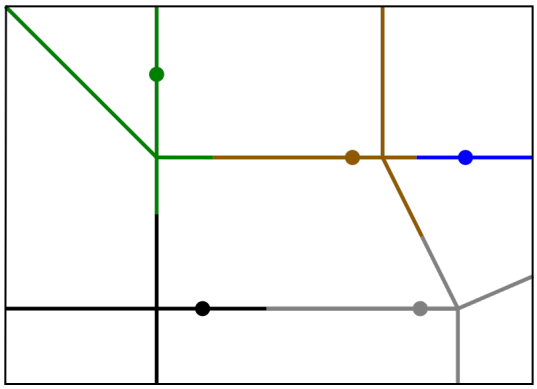


Gambar 2.3 Diagram voronoi dengan 20 *centroid*.

Pada geometri, diagram voronoi dapat menjawab pertanyaan *nearest neighbor query* yang meminta satu *centroid* terdekat dari suatu titik. Pada jaringan jalan raya, diagram voronoi juga dapat digunakan untuk mencari objek *centroid* terdekat dari suatu titik pada *edge*. Sebagai contoh, pada Gambar 2.4 menampilkan lima objek sebagai *centroid* serta area pada *edge* yang objek tersebut adalah objek terdekatnya. Karena jalan raya terdiri dari *edge* dan bukan merupakan ruang terbuka, jarak perlu dihitung dengan jarak jalan yang ditempuh, bukan dengan jarak eucledian.

Erwig pada (Erwig, 2000) mengusulkan algoritma modifikasi Dijkstra yang dinamai dengan Parallel Dijkstra. Pada umumnya algoritma Dijkstra hanya menggunakan satu titik awal untuk mencari *node-node* terdekat dengan membentuk *shortest-path tree*, sedangkan Parallel Dijkstra menggunakan objek-objek *centroid* yang terdapat pada ruang voronoi untuk mencari area dan *node* terdekat. *Shortest-path tree* tersebut tumbuh menyebar pada objek-objek

centroid secara paralel hingga semua *node* dan *edge* pada jaringan jalan raya dikunjungi salah satu *shortest-path tree*. Algoritma tersebut cukup efisien karena untuk hanya memerlukan *traverse* sekali untuk setiap *node*. Dengan demikian kompleksitasnya sama dengan kompleksitas algoritma Dijkstra, yaitu $O(|E| + |V \log|V|)$.



Gambar 2.4 Lima *controid* pada voronoi di jaringan jalan raya.

Graf voronoi digunakan pada (Fu et al., 2017) untuk mencari *Skyline Scope*/SS dari setiap objek. SS dari objek o merupakan area yang terdapat pada jaringan jalan raya yang objek o merupakan salah satu dari *skyline points* yang terdapat pada tempat tersebut. Komputasi menggunakan voronoi terbukti jauh lebih efisien dibandingkan menggunakan *landmark*.

2.7 *k-Dominant Skyline* pada Lingkungan Paralel

k-Dominant Skyline secara umum tidak didesain untuk komputasi dalam lingkungan paralel. Algoritma paralel untuk *k-dominant skyline* berbeda dengan *simple skyline query* karena adanya properti *cyclic dominance relationship*. Properti *cyclic dominance relationship* adalah sifat dari *k-dominant skyline* yang menjadikan objek yang telah didominasi dapat mendominasi objek yang lain hingga membentuk jaringan lingkaran sehingga objek yang masuk pada lingkaran tersebut tidak dapat menjadi *skyline* karena didominasi objek lain. Siddique pada (Siddique et al., 2015) mengusulkan algoritma *k-dominant skyline* yang beroperasi pada lingkungan *paralel* menggunakan metode MapReduce. Algoritma tersebut menggunakan struktur data OB-Tree, yaitu struktur data *tree* yang digunakan untuk

memisahkan data agar dapat dikomputasikan secara paralel. Peng dan Chen pada (Peng and Chen, 2019) menyelesaikan permasalahan komputasi *k-dominant skyline* dengan pendekatan yang berbeda, yaitu dengan memanfaatkan GPU dalam proses komputasinya. Algoritma tersebut menggunakan komputasi dalam bentuk *bitmap*.

Halaman ini sengaja dikosongkan

BAB 3
METODE PENELITIAN

3.1 Metodologi Penelitian

Untuk mencapai tujuan dari penelitian, penelitian ini perlu langkah-langkah yang sistematis agar dapat berjalan dengan baik. Studi literatur dilakukan terlebih dahulu dengan menganalisis penelitian yang sudah dilakukan dari berbagai sumber, khususnya jurnal dan artikel konferensi internasional. Penulis mencari permasalahan yang belum terdapat solusi dan mendefinisikannya. Selanjutnya penelitian terkait perlu dipelajari untuk mendapatkan solusi dalam menyusun struktur data dan algoritma. Setelah itu, struktur data dan algoritma disusun dengan tujuan utama mendapatkan hasil dengan komputasi dan sumber daya yang minimal. Setelah mengimplementasikan struktur data dan algoritma, penulis melakukan pengujian dan mengevaluasi hasil. Evaluasi dilakukan dengan membandingkan dengan algoritma yang sudah ada dan disesuaikan dengan kasus dari penelitian ini. Evaluasi dilakukan untuk mengetahui kelebihan dan kekurangan masing-masing algoritma.

Pada bab ini, pertama-tama penulis mendefinisikan dasar teori serta permasalahan dalam definisi formal. Kemudian penulis menjabarkan metode yang diusulkan disertai struktur data dan algoritmanya. Untuk memperjelas, Tabel 3.1 menjelaskan definisi setiap simbol yang digunakan pada penelitian ini.

Simbol	Keterangan
$G(V, E)$	Jaringan jalan raya dengan V sebagai himpunan *node* dan E sebagai himpunan jalan
$\mathbf{O}, O$	Himpunan objek dan jumlahnya
o, o_i	Sebuah objek dan objek ke-i
$\mathbf{D}, D$	Himpunan dimensi dan jumlah dimensinya
$\mathbf{Q}, Q$	Himpunan *query* dan jumlahnya
q, q_i	Sebuah *query* dan *query* ke-i
$q_i.k$	Nilai k dari *query* ke-i
$q_i.\mathbf{D}, q.D$	Himpunan dimensi dan jumlahnya di *query* ke-*i*
Dom(o)	Himpunan objek yang mendominasi o
Dominate(o)	Himpunan objek yang didominasi oleh o
kSQ$(k, \mathbf{D'})$	*k-dominant skyline* di jaringan jalan raya
PDSQ(p, k)	*Point-based Dominant Skyline Query*
DSS(o, k)	*Dominant Skyline Scope*
$o_i \prec o_j$	Objek o_i mendominasi o_j pada data nonspasial
$o_i \prec_k o_j$	Objek o_i me-*k-dominate* o_j
$o_i \prec_q o_j$	Objek o_i mendominasi o_j pada data spasial relatif terhadap titik *query* q
$o_i \prec_{kq} o_j$	Objek o_i mendominasi o_j pada data spasial dan nonspasial

Tabel 3.1 Simbol dan keterangannya

3.2 Algoritma dan Struktur Data

3.2.1 Definisi Permasalahan

Ruang dimensi $\mathbf{D} = \{d_1, d_2, ..., d_D\}$ memiliki himpunan objek $\mathbf{O} = \{o_1, o_2, ..., o_O\}$. Objek-objek tersebut berada di jaringan jalan raya $G(V, E)$ dengan E sebagai *edge* dan V sebagai *node*. Setiap objek memiliki beberapa atribut nonspasial (contoh, kualitas dan harga) dan satu atribut spasial (yaitu, jarak).

Definisi 3.2.1. (Dominansi). Jika terdapat dua objek, o_i dan o_j, o_i mendominasi o_j jika semua atribut nonspasial dari o_i tidak lebih buruk dibandingkan o_j dan terdapat minimal satu atribut nonspasial pada o_i yang lebih baik dibandingkan atribut pada o_j. Secara formal, objek o_i mendominasi o_j, dinotasikan $o_i \prec o_j$, jika dan hanya jika $\exists\, d_m \in \mathbf{D},\ o_{i,m} \geq o_{j,m}$, dan $\exists\, d_n \in \mathbf{D},\ o_{i,n} > o_{j,n}$.

Definisi 3.2.2. (Spasial dominansi). Titik *query* q, objek o_i, dan objek o_j berada pada jaringan jalan raya. Objek o_i dikatakan mendominasi spasial o_j jika i) o_i mendominasi o_j secara nonspasial dan ii) o_i mendominasi o_j secara spasial. Dalam konteks ini, o_i mendominasi o_j secara spasial jika jarak o_i titik *query* q lebih dekat dibandingkan jarak o_j terhadap q. Secara formal, objek o_i mendominasi o_j relatif terhadap titik *query* q, dinotasikan $o_i \prec_q o_j$, jika $o_i \prec o_j$ dan dist$(o_i, q) <$ dist(o_j, q).

Definisi 3.2.3. (*k-dominate*). Objek o_i *k-dominate* objek o_j apabila terdapat k atribut dengan $k < D$ di o_i yang tidak lebih buruk daripada k atribut di o_j dan terdapat satu atribut di k dari objek o_i yang lebih baik dibandingkan yang di o_j. Secara formal, objek o_i *k-dominate* objek o_j, dinotasikan $o_i \prec_k o_j$ apabila $\exists$ **D'** $\subseteq$ **D**, $D' = k$, $\forall\, d_m \in$ **D'**, $o_{i,m} \geq o_{j,m}$, dan $\exists\, d_n \in$ **D'**, $o_{i,n} > o_{j,n}$.

Definisi 3.2.4. (*k-dominate* di jaringan jalan raya). Sebuah objek o_i *k-dominate* o_j, dinotasikan $o_i \prec_{kq} o_j$, di titik *query* q pada jaringan jalan raya apabila o_i mendominasi o_j dilihat dari atribut nonspasial($o_i \prec_k o_j$) dan spasial($o_i \prec_q o_j$).

Objek	d_1	d_2	d_3	d_4	d_5	d_6
o_1	5	5	5	3	3	3
o_2	3	3	3	5	5	5
o_3	4	4	4	2	4	4
o_4	2	2	2	6	2	2
o_5	3	3	3	4	4	4

Tabel 3.2 Contoh dataset **D**.

Sebagai contoh, Tabel 3.2 menampilkan lima objek dalam enam dimensi/atribut pada dataset **D**. **D** memiliki empat *simple skyline*, yaitu o_1, o_2, o_3, dan o_4. Dari keempat objek tersebut, hanya terdapat tiga *5-dominant skyline*, yaitu o_1, o_2, dan o_3. Objek o_4 di-*5-dominate* oleh o_2.

k-dominant skyline query memungkinkan terdapatnya *cyclic dominance relationship* hingga mengakibatkan tidak ada objek yang menjadi *k-dominant skyline*. Dalam artian, objek saling mendominasi dan membentuk hubungan dominansi lingkaran. Perhatikan Tabel 3.3, tabel tersebut memperlihatkan *cyclic*

Objek	d_1	d_2	d_3	d_4
o_1	4	4	4	4
o_2	8	3	3	3
o_3	7	8	2	2
o_4	6	7	8	1

Tabel 3.3 Contoh dataset dengan *cyclic dominance relationship*.

dominance relationship ketika nilai $k = 3$. Secara spesifik, o_1 *k-dominate* o_2, o_2 *k-dominate* o_3, o_3 *k-dominate* o_4, dan o_4 *k-dominate* o_1.

Definisi 3.2.5. (*k-dominant skyline* di jaringan jalan raya(kSQ(k, **D'**))). Sebuah objek o_i dikatakan anggota dari *k-dominant skyline* di titik query q apabila objek tersebut tidak didominasi(*k-dominate*) oleh objek lain o_j dari atribut spasial maupun nonspasial. Secara formal, kSQ(k, **D'**) $= \{$**O'** | **O'** $\subseteq$ **O**, $\forall\, o' \in$ **O'**, $\forall\, o \in$ **O**, $o \not\prec_{kq} o'$, $o \neq o'\}$.

Secara formal, permasalahan pada penelitian ini dapat didefinisikan sebagai berikut:

Definisi 3.2.6. (*Continuous multiqueries k-dominant skyline* di jaringan jalan raya). Himpunan objek **O** yang terdapat pada jaringan jalan raya $G(V, E)$ dengan himpunan dimensi **D** serta himpunan *query* **Q**. Untuk setiap *query* $\forall\, q_i \in$ **Q** dengan dimensi q_i.**D**, q_i.**D** $\subseteq$ **D** dan $q_i.k, q_i.k < D$, **hitung** *k-dominant skyline* di setiap titik p di E, $\forall\, p \in E$.

3.3 *Index-Based k-Dominant Skyline Query* di Jaringan Jalan raya (IKSR)

Ide pokok dari algoritma ini adalah memaksimalkan penggunaan voronoi diagram untuk menghitung beberapa *query* dalam satu komputasi. Algoritma ini dapat menjalankan *k-dominant skyline query* yang memiliki nilai *k* yang berbeda. Namun *query-query* tersebut harus berada dalam himpunan dimensi **D** yang sama dan **O** yang sama. Sebagai contoh, perhatikan Gambar 3.1, q_1, q_2, dan q_3 dibuat dari himpunan dimensi **D** dan himpunan *query* **Q**. Setiap *query* memiliki *subspace* **D'** $\subseteq$ **D** dan nilai k. Dari ketiga *query* tersebut, q_1 dan q_2 dapat dikomputasikan dalam sekali komputasi voronoi karena memiliki **D'** yang sama.

$$\mathbf{D} = \{d_1, d_2, d_3, d_4, d_5\}$$
$$\mathbf{Q} = \{q_1, q_2, q_3\}$$
$$\mathbf{D'} \subseteq \mathbf{D}$$

$$q_1 \begin{cases} \mathbf{D'} = \{d_1, d_2, d_3, d_4\} \\ k = 4 \end{cases}$$

$$q_2 \begin{cases} \mathbf{D'} = \{d_1, d_2, d_3, d_4\} \\ k = 3 \end{cases}$$

$$q_3 \begin{cases} \mathbf{D'} = \{d_1, d_2, d_3\} \\ k = 2 \end{cases}$$

Gambar 3.1 Penggabungan *query* q_1 dan q_2.

Pada subbab ini, penulis menyampaikan analisis, struktur data, dan algoritma yang digunakan untuk mengkomputasi *k-dominant skyline* di jaringan jalan raya. Sebelum membahas lebih jauh, beberapa konsep dan definisi yang digunakan pada subbab selanjutnya perlu dijelaskan.

Definisi 3.3.1. *Point-based Dominant Skyline Query*(PDSQ(p, k)). Himpunan objek **O** serta titik query p pada himpunan dimensi **D** terdapat pada jaringan jalan raya $G(V, E)$. PSQ(p, k) mengeluarkan semua objek, o' yang tidak di-*k-dominate* oleh objek lain o baik dari dimensi nonspasial maupun spasial. Secara formal, pada ruang dimensi **D**, PSQ(p, k) = $\{\mathbf{O'} | \mathbf{O'} \subseteq O, \forall o' \in \mathbf{O'}, \forall o \in \mathbf{O}, o \not\prec_k o', o \neq o', \text{dist}(p, o') < \text{dist}(p, o)\}$.

Definisi 3.3.2. *Dominant Skyline Scope* (DSS(o, k)). *Dominant Skyline Scope* dari objek o dengan di himpunan dimensi **D** pada jaringan jalan raya $G(V, E)$ adalah semua area pada E, $E' \subseteq E$, yang o menjadi anggota dari *skyline points* dari area E' tersebut dengan jarak maksimal d_ε. Secara formal, DSS$(o, k) = \{E' | E' \subseteq E, \forall\, p \in E', o \in \text{PDSQ}(p, k), \text{dist}(p, o) <= d_\varepsilon\}$.

3.3.1 Analisis

Pertama penulis menyampaikan mengenai ciri dan sifat dari voronoi yang digunakan untuk mengkomputasi *simple skyline query* pada jaringan jalan raya. Fu dkk. (Fu et al., 2017) mengusulkan algoritma *Index Based Algorithm*(IBA) yang menggunakan *voronoi cell* untuk menyelesaikannya. *Voronoi cell* merupakan area dari suatu objek o_i yang objek tersebut merupakan objek terdekat. Objek o_i dapat dinyatakan sebagai *centroid* karena ia berada di tengah *voronoi cell* tersebut. Fu

dkk. (Fu et al., 2017) menjadikan *voronoi cell* tersebut sebagai SS/*Skyline Scope*. Pada tesis ini, penulis memodifikasi SS menjadi DSS/Dominant Skyline Scope. Perbedaanya adalah metode ini tidak menggunakan operasi *k-dominate* untuk membandingkan dominansi antarobjek, sedangkan metode sebelumnya menggunakan operasi *simple dominate skyline*. Selanjutnya, penulis menggunakan istilah *voronoi cell* yang juga berarti DSS untuk memudahkan penjelasan.

Suatu objek o_i yang terdapat pada E dinyatakan sebagai *skyline* apabila o_i tidak didominasi oleh objek lain pada atribut spasial dan nonspasial. Untuk menghitung luas dari *voronoi cell* tersebut, algoritma perlu memastikan bahwa di area tersebut objek tidak didominasi objek lain. Sehingga, algoritma perlu mengetahui objek-objek lain di sekitar yang mendominasinya. Dari penjelasan tersebut, dapat disimpulkan dengan Proposisi 1.

Proposisi 1. Semakin banyak objek yang mendominasi, maka semakin banyak *voronoi cell* yang digunakan untuk proses komputasi.

Sesuai tujuannya, *k-dominant skyline* digunakan untuk memperkecil *output* dari *skyline query* dengan mempertimbangkan *k* atribut untuk perbandingan dominansinya. Hal tersebut diungkapkan pada teorema dan *corollary* yang telah disampaikan oleh Chan dkk. (Chan et al., 2006) yang menyatakan bahwa:

Teorema 1. Jika titik p_i merupakan *k-dominant skyline*, $p_i \in DSP(k, \mathbf{D}, \mathbf{S})$, maka p_i merupakan $p_i \in DSP(k+1, \mathbf{D}, \mathbf{S})$.

Corollary 1.1. $|DSP(k, \mathbf{D}, \mathbf{S})| \leq |DSP(k+1, \mathbf{D}, \mathbf{S})|$

Dengan demikian, semakin kecil nilai *k* yang digunakan pada *k-dominant skyline query*, maka semakin sedikit jumlah *output* yang dihasilkan. Hal tersebut dikarenakan pada *k* yang kecil, objek cenderung lebih mudah mendominasi jika hanya mempertimbangkan beberapa atribut dan kemungkinan terdapatnya *cyclic dominance relationship* lebih besar dibandingkan nilai k yang besar.

Kembali pada definisi *skyline point* pada jaringan jalan raya di (Fu et al., 2017), Fu dkk. menyatakan bahwa objek dikatakan *skyline* pada jaringan jalan raya jika tidak didominasi pada atribut nonspasial dan spasial oleh objek lain.

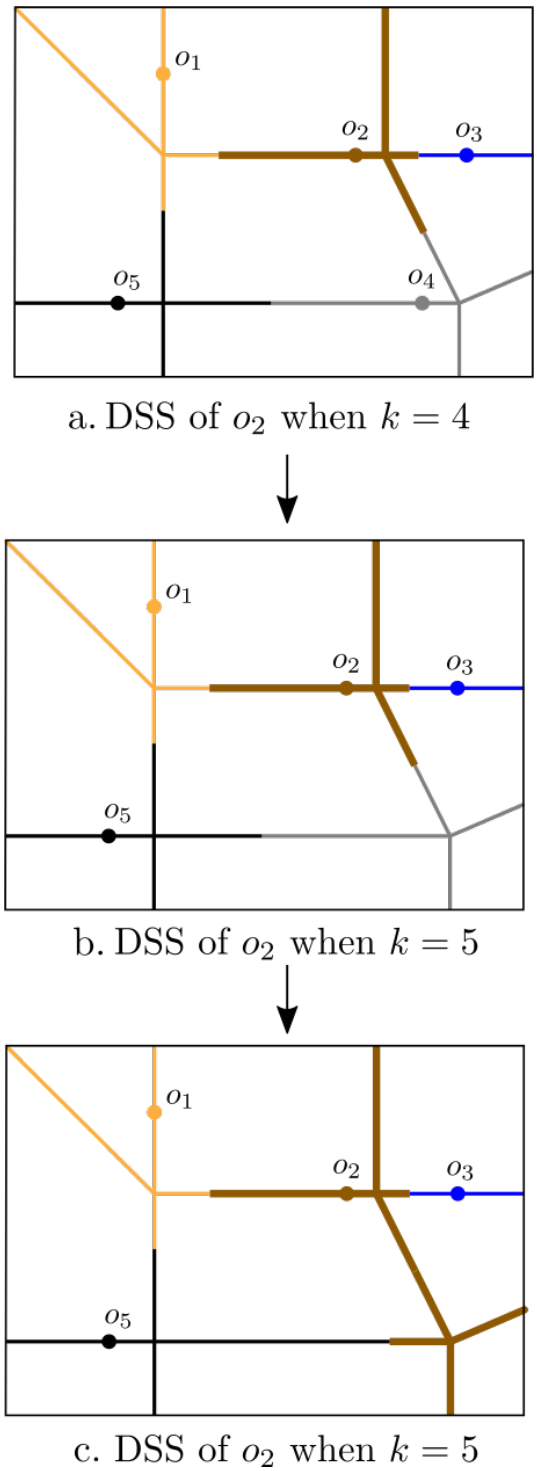
Secara spesifik, objek o dikatakan *skyline point* pada titik p apabila tidak didominasi oleh objek lain o', $o' \in$ **O** dan jaraknya tidak lebih dari jarak setiap objek yang mendominasinya secara nonspasial, $\forall\, o' \in \text{Dom}(o)$, $\text{dist}(p, o) < \text{dist}(p, o')$. Hal tersebut senada dengan definisi *k-dominant skyline point* pada tesis ini sebagaimana yang tertulis di Definisi 3.2.5. Perbedaan metode ini dengan (Fu et al., 2017) adalah proses perbandingan dominansi antarobjek pada atribut nonspasial. Untuk menyatakan dominansi, *simple skyline* menggunakan keseluruhan dimensi **D** dengan jumlah D sedangkan *k-dominant skyline* hanya menggunakan *k* dimensi tertentu, $k < D$. Dengan demikian, konsep pada Proposisi 1 yang bekerja untuk *simple skyline* juga dapat diaplikasikan pada kasus *k-dominant skyline*.

Pada Corollary 1.1, semakin besar nilai *k* berarti memungkinkan semakin besar juga jumlah *skyline points*-nya. Hal tersebut dikarenakan jumlah objek yang mendominasi dan didominasi semakin sedikit. Proposisi 1 juga dapat berlaku bahwa semakin sedikit objek yang mendominasi, maka semakin sedikit jumlah *voronoi cell* yang perlu dikomputasi. Dengan demikian, kesimpulan dapat diambil dengan pernyataan:

Proposisi 2. Jika semakin besar nilai *k*, maka memungkinkan adanya pengurangan *voronoi cell* pada graf voronoi.

Proposisi 3. Setiap objek/*voronoi cell* yang terdapat pada $k + 1$ graf voronoi, pasti terdapat pada k graf voronoi.

Terkait algoritma Parallel Dijkstra(Erwig, 2000), algoritma ini melakukan *traverse* dari semua *centroid* -atau objek dalam konteks tesis ini- sebagai awal *traverse* dan membuat area yang menjauh dari *centroid* untuk membuat *voronoi cell* untuk masing-masing *centroid* pada jaringan jalan raya. Berdasarkan Teorema 1, setiap objek yang merupakan *k-dominant skyline* di *k* akan menjadi $k + 1$-*dominant skyline*. Demikian juga yang terdapat pada komputasi graf voronoi, setiap objek menjadi *centroid* di $k + 1$ juga menjadi centroid di k graf voronoi.



Gambar 3.2 DSS dari objek o_2 dari $k = 4$ ke $k = 5$

Berdasarkan uraian diatas, untuk menghitung voronoi dari $k + 1$ graf voronoi, algoritma hanya perlu menghapus *voronoi cell* dari *centroid* yang tidak terdapat pada k graf voronoi kemudian menghitung area yang belum dikomputasi

saja. Dengan demikian, komputasi graf voronoi harus dimulai dari nilai k yang terkecil ke nilai k yang terbesar atau dengan urutan *descending*. Komputasi tersebut menjadi lebih efisien karena dapat memaksimalkan beberapa komputasi Parallel Dijkstra menjadi satu kali komputasi saja. Namun, komputasi tersebut hanya berlaku jika beberapa *k-dominant skyline query* berada pada *subspace* **D** dan himpunan objek **O** yang sama sebagaimana yang terdapat pada Teorema 1.

Sebagai contoh, perhatikan Gambar 3.2, gambar tersebut menunjukkan kasus proses penghitungan DSS dari objek o_2 ketika $k = 3$ dari data DSS ketika $k = 4$. Sebelumnya, Gambar 3.2 mengasumsikan bahwa objek o_2 didominasi oleh o_1, o_3, o_4, o_5 dituliskan dengan Dom(o_2) $= \{o_1, o_3, o_4, o_5\}$ ketika nilai $k = 4$. Ketika nilai $k = 3$, objek o_2 tidak lagi di-*k-dominate* oleh o_4 sehingga Dom(o_2) $= \{o_1, o_3, o_5\}$. Gambar 3.2.a menampilkan DSS/*voronoi cell* dari objek o_2 yang ditandai dengan garis yang tebal. Karena objek o_4 tidak lagi mendominasi o_2, o_4 tidak lagi dikomputasi ketika $k = 5$ sebagaimana Gambar 3.2.b. Dengan demikian, *edge* yang sebelumnya ditempati oleh DSS dari o_4 tidak lagi digunakan. Objek o_2 dan o_5 merupakan objek yang secara langsung bersinggungan dengan DSS dari o_4 sehingga DSS dari kedua objek tersebut meluas untuk menempati DSS dari o_4.

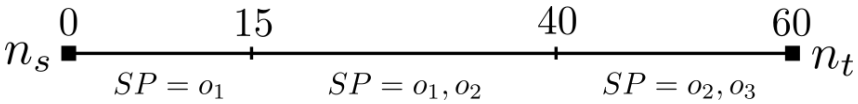
Terdapat satu ciri yang dimiliki oleh *k-dominant skyline*, yaitu kemungkinan adanya *cyclic dominance relationship*(Chan et al., 2006). Hal ini tidak terjadi di kasus yang terdapat pada tesis ini, yaitu *k-dominant skyline* di jaringan jalan raya. Sebagaimana yang tertulis pada Definisi 3.2.5, terdapat dua properti yang menjadikan sebuah objek bagian dari *k-dominant skyline*, yaitu objek tidak didominasi oleh objek lain dalam hal atribut spasial dan nonspasial. Jika metode ini hanya mempertimbangkan atribut nonspasial, maka ia memungkinkan terdapatnya *cyclic*. Namun, metode ini juga mempertimbangkan atribut spasial, yaitu jarak, sebagai faktor dominansi. Jarak setiap objek $\forall\, o \in$ **O** pada suatu titik p, dist$(o, p) = d_o$ tidak memungkinkan terjadinya *cyclic*. Sebagai contoh, tidak mungkin terdapat tiga objek yang jaraknya saling mendominasi seperti $d_a < d_b < d_c < d_a$. Hal tersebut tidak akan terjadi. Dengan demikian jika tidak dibatasi oleh jarak maksimal d_{max}, maka setiap titik p di jaringan jalan raya,

$\forall\, p \in E$, terdapat minimal satu objek yang menjadi *k-dominant skyline*.

3.3.2 Struktur Data dan Algoritma

Metode yang diusulkan pada tesis ini bersifat *continuous*, yaitu algoritma ini akan mengkomputasi hasil *k-dominant skyline* di semua area yang terdapat pada jaringan jalan raya dan tidak tergantung dari jumlah dan lokasi *query* yang diminta oleh pengguna. Metode ini melakukan *indexing* pada setiap *edge*/jalan di jaringan jalan raya sehingga ketika pengguna mencari informasi mengenai *skyline* di suatu titik $p \in E$, algoritma hanya perlu melakukan *lookup* pada hasil *indexing* tersebut.

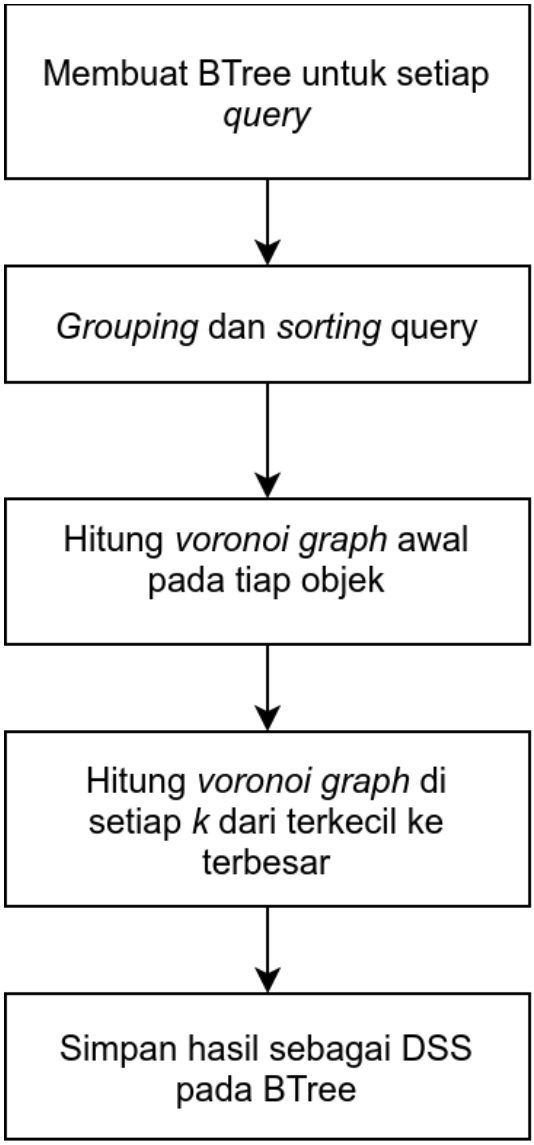
Hasil dari komputasi *k-dominant skyline* ini disimpan dalam struktur data BTree(Fu et al., 2017). Setiap *edge* di jalan raya memiliki struktur BTree untuk setiap *query* yang berupa <*key* , *SP*>. BTree melakukan *indexing* pada *key*-nya dan membentuk struktur *tree* dengan kompleksitas O(log *n*) untuk proses pencarian/*lookup*. *Key* pada struktur tersebut merupakan jarak maksimal *SP*(himpunan objek yang menjadi *skyline points*) menjadi *skyline*.



Gambar 3.3 Visualisasi BTree pada *edge*

Sebagai contoh visualisasi representasi *skyline points* pada BTree, perhatikan Gambar 3.3. Objek o_1 menjadi *skyline* di titik 0 atau n_s hingga titik 40, objek o_2 dari jarak 15 hingga jarak 60 atau n_t, dan objek o_3 menjadi *skyline* dari jarak 40 hingga akhir atau n_t. Struktur tersebut direpresentasikan dalam BTree sebagai berikut dengan setiap poin sebagai *node* yang terdapat pada BTree:

- $< 0, \{o_1\} >$
- $< 15, \{o_1, o_2\} >$
- $< 40, \{o_1, o_3\} >$



Gambar 3.4 Alur konstruksi *k-dominant skyline*

Dengan demikian, setiap *node* dari BTree hanya perlu menyimpan awal jarak dari SP dan SP tersebut tetap valid hingga ada *node* setelahnya. Jika *node* tersebut *node* terakhir dari BTree, maka *SP* tersebut valid hingga n_s/ujung akhir dari *edge*. BTree tersebut juga menyimpan metadata berupa *subspace* **D** dan nilai k dari *query* yang digunakan untuk menghasilkannya.

Secara umum, algoritma pembuatan *k-dominant skyline* dari jaringan jalan raya beserta objek yang sudah terdapat di dalamnya dapat digambarkan dalam alur yang terdapat pada Gambar 3.4. Pertama, BTree kosong dibuat pada setiap *edge* untuk setiap *query*. BTree digunakan karena memiliki kompleksitas

```
Input: the set of objects O in road network G(V, E), the set of queries Q
Output: k-dominant skyline BTree for each query in each edge
for e ∈ E do
    for q ∈ Q do
        create empty BTree for q in e;
    end
end
G = group each q ∈ Q based on it's dimentions/subspace;
for g ∈ G do
    sort q ∈ g based on q.k in ascending order;
end
for o ∈ O do
    for g ∈ G do
        for q ∈ g do
            if q.k is the smallest then
                V = compute initial voronoi graph;
            else
                V = delete voronoi cells in V;
                V = compute unused road network in V;
            end
            save voronoi V as DSS of o into BTree on each edge;
        end
    end
end
```

Gambar 3.5 Algoritma *construction*: Konstruksi awal *k-dominant skyline*.

yang ringan untuk operasi *insertion*, *deletion*, maupun *lookup*. Proses *insertion* dan *deletion* digunakan untuk memodifikasi *Dominant Skyline Scope* ketika terdapat objek masuk pada sistem maupun ketika objek keluar. Proses *lookup* pada BTree banyak digunakan untuk mencari *k-dominant skyline* sesuai permintaan pengguna berdasarkan lokasi yang terdapat pada *edge*. Selanjutnya, algoritma perlu melakukan *grouping* dan *sorting* dari *query* untuk menggabungkan *query* yang memiliki *subspace* yang sama. Selanjutnya, untuk setiap objek pada setiap *query* perlu dihitung *voronoi graph*-nya di awal komputasi dengan nilai *k* yang terkecil. Kemudian *voronoi graph* perlu dihitung untuk setiap *k* yang berukuran lebih besar secara berurutan sesuai dengan *query* yang ada. Setelah itu, *voronoi cell* dari objek *o* pada masing-masing *query* disimpan untuk masing-masing BTree yang ada. Secara umum, algoritma *construction* digunakan untuk mempercepat

komputasi pada objek yang sudah terdapat pada jaringan jalan raya tanpa harus mengkomputasi ulang dengan proses *insertion* satu per satu.

Gambar 3.5 menunjukkan algoritma pembuatan *k-dominant skyline* yang menghasilkan BTree dari setiap *query* pada setiap *edge*. BTree pada setiap *edge* tersebut menyimpan hasil komputasi *k-dominant skyline* pada setiap *edge* masing-masing. Pada baris 1-5, untuk setiap *edge* yang terdapat pada jaringan jalan raya, dibuatkan BTree untuk setiap *query* yang terdapat pada *Q*/*query* masukan. Struktur BTree diperlukan untuk setiap *query* agar proses pencarian data dapat berjalan lebih cepat dibandingkan menyimpan semua hasil *k-dominant skyline* pada semua *query* dalam satu struktur BTree.

Pada baris 6, setiap *query* dikelompokkan berdasarkan dimensinya q.**D**. Hal tersebut diperlukan karena komputasi *k-dominant skyline* yang banyak dala satu dimensi/*subspace* dapat dilakukan dalam sekali komputasi *voronoi*. Selanjutnya pada baris 7-9, setiap *query* yang terdapat pada masing-masing *group* perlu disortir dari nilai *k* yang terkecil hingga tertinggi karena komputasi pembentukan voronoi berjalan dari nilai *k* terkecil hingga ke yang terbesar.

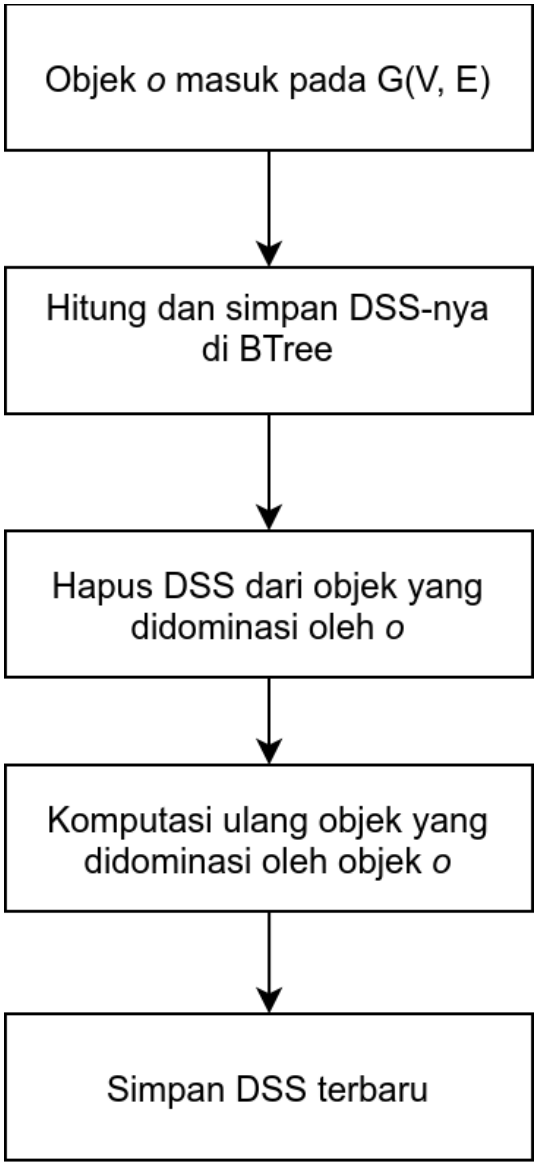
Selanjutnya untuk setiap objek yang terdapat pada jaringan jalan raya dikomputasikan *query* pada setiap *group*. Pada baris ke-13, jika q merupakan *query* pertama, maka ia menghitung *voronoi cell* dari objek o tersebut secara keseluruhan. Kemudian, pada baris ke-15, jika *query* bukan pertama, maka ia perlu menghapus voronoi cell yang tidak digunakan pada komputasi dengan nilai *k*. Hal ini perlu mengingat bahwa pada *(k + 1)-dominant skyline* terdapat kemungkinan bahwa objek tidak lagi mendominasi sebagaimana pada *k-dominant skyline*. Terakhir, simpan *voronoi cell* dari objek o sebagai DSS dengan metadata dari *query* q pada setiap BTree di *edge* yang sesuai.

Pada baris 14, pengkomputasian voronoi diawali dari salah satu objek sebagai *centroid*. Algoritma melakukan *traverse* dari objek tersebut ke *node* dan *edge* sekitar hingga jarak $2 \times d_\varepsilon$ untuk mencari objek-objek yang memiliki kemungkinan dominansi. Setiap objek yang mendominasi pada masing-masing *k* dimensi disimpan. Untuk iterasi pertama, algoritma mencari semua objek dengan *k*

yang paling kecil kemudian mencari *voronoi cell* dari *centroid* dengan algoritma Parallel Dijkstra (Erwig, 2000). *Voronoi cell* dari *centroid* adalah DSS dari *centroid*, artinya setiap *query* yang terdapat pada *voronoi cell* tersebut mendapati objek *centroid* sebagai *k-dominant skyline*-nya. Proses *traverse* pada Parallel Dijkstra tersebut menggunakan jarak maksimal d_ε. Jarak $2 \times d_\varepsilon$ tidak digunakan dalam hal ini karena tujuan dari menggunakan Parallel Dijkstra adalah mencari DSS dan jarak maksimal dari DSS merupakan d_ε sebagaimana yang tertulis pada batasan masalah.

Pada iterasi selanjutnya yang menggunakan nilai *k*, baris 16 dijalankan. Setiap objek yang didominasi *centroid* pada *k - 1* dan tidak lagi mendominasi pada *k* dihapus *voronoi cell*-nya. Kemudian algoritma melakukan *traverse* dari setiap *neighbor* yang terdapat di sekitar *voronoi cell* yang dihapus untuk mengisi area yang kosong.

Selanjutnya penulis memaparkan algoritma untuk proses *insertion*, yaitu masuknya objek dalam sistem dan menghitung *k-dominant skyline* di setiap *edge* untuk setiap *query*.



Gambar 3.6 Alur keluarnya objek/*insertion*

Gambar 3.6 memperlihatkan alur komputasi ketika objek masuk pada sistem. Pertama, setelah objek masuk pada graf, algoritma menghitung DSS dari objek yang masuk dengan cara mengkomputasi *voronoi diagram* untuk nilai *k* yang terkecil kemudian mengiterasi pada semua nilai *k* pada tiap *group* di *query* dari nilai terkecil ke terbesar dan melakukan komputasi untuk mencari DSS-nya. Kemudian semua DSS tersebut disimpan pada BTree dari masing-masing *edge* pada masing-masing *query*. Selanjutnya algoritma mencari objek yang didominasi dari objek yang masuk kemudian menghapus DSS dari objek-objek tersebut yang

```
Input: object o in inserted to G(V, E)
Output: updated k-dominant skyline in G(V, E)
for g ∈ G do
    for q ∈ g do
        if q.k is the smallest then
            V = compute initial voronoi graph;
        else
            V = delete voronoi cells in V;
            V = compute unused road network in V;
        end
        save voronoi V as DSS of o into BTree on each edge;
    end
    for each object m ∈ Dominate(o) do
        Q' = ∅;
        for q ∈ G do
            if o (q.k)-dominate m then
                append q into Q';
                remove DSS of m from related BTree of query q;
            end
        end
        for q ∈ Q' do
            if q.k is the smallest then
                V = compute initial voronoi graph;
            else
                V = delete voronoi cells in V;
                V = compute unused road network in V;
            end
            save voronoi V as DSS of m into BTree on each edge;
        end
    end
end
```

Gambar 3.7 Algoritma *insertion*: masuknya objek o pada jaringan jalan raya.

terdapat pada BTree agar tidak terdapat redundansi data dan *false positive*. Proses *deletion* pada langkah ini memiliki kompleksitas O(log n) karena menggunakan struktur data BTree. Objek-objek yang didomniasi tersebut kemudian dihitung ulang DSS-nya dengan cara mengiterasi setiap *query* dengan mendahulukan nilai *k* yang paling kecil hingga nilai *k* yang paling besar. Pada nilai *k* yang paling kecil, algoritma mencari objek-objek di sekitar objek yang dimasukkan pada sistem dan didominasi oleh objek yang dimasukan tersebtu dan berjarak kurang dari $2 \times d_\varepsilon$. Jarak tersebut didefinisikan dua kali lipat karena DSS dari setiap objek dapat mencapai d_ε sehingga ada kemungkinan objek satu mendominasi objek yang lain di *edge* yang berada diantara kedua objek tersebut. Dari alasan tersebut, objek masih dapat mendominasi jika jarak kedua objek tersebut kurang dari $2 \times d_\varepsilon$. Di akhir, DSS baru dari objek yang didominasi tersebut disimpan pada BTree. Proses *insertion* ini berjalan dengan kompleksitas O(log n) karena menggunakan BTree.

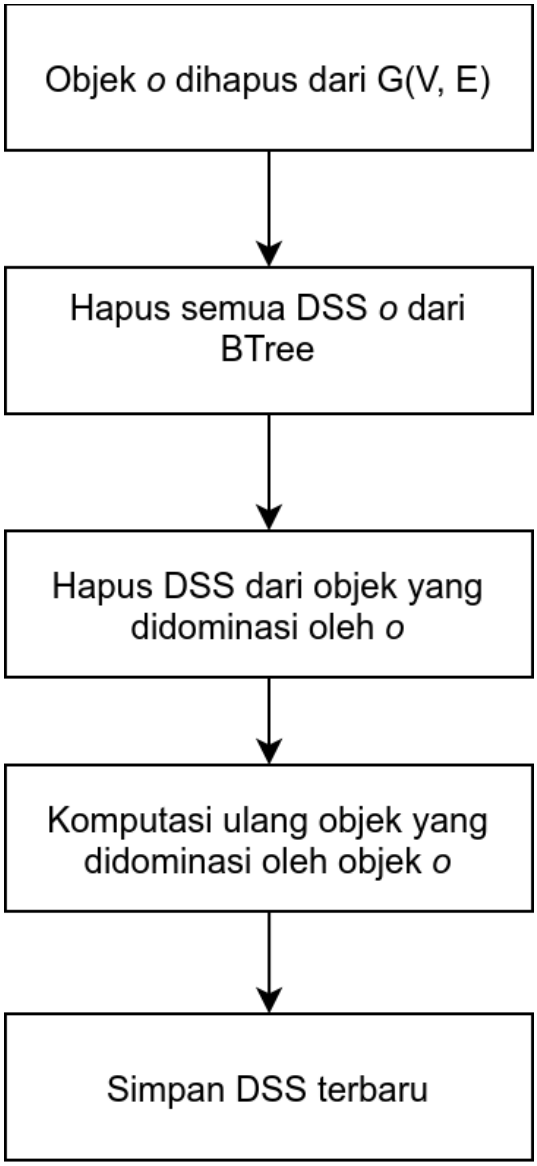
Algoritma *insertion* pada Gambar 3.7 mengasumsikan bahwa komputasi tersebut dilakukan setelah komputasi dengan algoritma *insertion* dilakukan. Dengan demikian tidak lagi diperlukan konstruksi BTree di setiap terdapat objek masuk pada sistem. Algoritma *insertion* dimulai dengan mengiterasi setiap *group*. Komputasi setiap *group* dijalankan tanpa ada ketergantungan satu sama lain.

Baris 2-10 digunakan untuk mencari DSS dari objek *o* yang dimasukkan pada sistem. Komputasinya dilakukan dengan cara yang sama sebagaimana yang terdapat pada algoritma *construction* baris 12-22. Voronoi dari objek o dihitung dengan cara mencari objek-objek di sekitar yang mendominasi o dengan jarak maksimal d_ε. Untuk iterasi pertama pada baris 4, algoritma menggunakan *k* yang terkecil untuk mencari objek yang mendominasi *o*. Pada iterasi *k* selanjutnya, objek yang tidak lagi mendominasi o dihapus *voronoi cell*-nya pada baris 6. Kemudian *voronoi cell* dari objek yang telah dihapus dihitung ulang dengan melakukan *traverse* dari *neighbor* sekitar hingga semua *edge* telah di-*traverse* dengan batasan maksimal d_ε.

Masuknya objek baru pada sistem ini mempengaruhi objek yang didominasi dalam hal DSS. Sistem perlu memperbarui semua objek yang

didominasi dengan menghitung ulang objek-objek tersebut sebagaimana yang tertulis pada algoritma *insertion* baris 11-28. Pada baris 11, algoritma mencari dan mengiterasi semua objek yang didominasi yang memiliki jarak $2 \times d_\varepsilon$ dari objek yang masuk o. Baris 12 menginisialisasi array **Q'** dari *queue*. **Q'** menyimpan *query-query* yang secara valid menyatakan bahwa objek o mendominasi m. Hal tersebut diperlukan dalam rangka optimasi untuk memastikan bahwa hanya *query* yang diperlukan saja yang perlu dihitung karena komputasi *voronoi* memiliki waktu komputasi yang signifikan. Pada baris 13-18, algoritma mengiterasi setiap *query* di *group* dan memasukkan setiap *query* yang o mendominasi m sesuai dengan $q.k$ dalam **Q'**. Pada baris 16, DSS dari objek m perlu dihapus dari BTree hasil agar tidak adanya *false positive* pada *edge* yang tidak menjadi DSS dan agar tidak ada redundansi data pada *edge* yang menjadi DSS dari objek m.

Pada baris 19-27, komputasi penghitungan DSS dilakukan dengan *query* yang berasal dari **Q'**. Pada segmen ini, algoritma menghitung *voronoi cell* dari objek m pada setiap *query* dan menyimpannya pada BTree. Dengan demikian, *k-dominant skyline* dari objek m selesai diperbarui pada *group*.



Gambar 3.8 Alur keluarnya objek/*deletion*

Gambar 3.8 menampilkan alur keluarnya sebuah objek dari jaringan jalan raya dan proses komputasi ulang agar *k-dominant skyline* tetep terbarui. Alur dimulai dengan dihapusnya objek dari jaringan jalan raya. Selanjutnya, objek dihapus dari semua BTree hasil. Selanjutnya, sebagaimana alur pada *insertion*, algoritma mencari semua objek yang dulunya didominasi dengan jarak kurang dari $2 \times d_{\varepsilon}$. Objek yang didominasi tersebut dihitung ulang DSS-nya dengan diawali dengan *query* yang memiliki nilai *k* yang kecil kemudian dilanjutkan komputasinya kepada *query* yang memiliki nilai *k* yang lebih besar. Setelah semua

```
   Input: object o is deleted from G(V, E)
   Output: updated k-dominant skyline in G(V, E)
1  for g ∈ G do
2      for q ∈ g do
3          remove DSS of o from related BTree of query q;
4      end
5      for each object m ∈ Dominate(o) do
6          Q' = ∅;
7          for q ∈ G do
8              if o (q.k)-dominate m then
9                  append q into Q';
10                 remove DSS of m from related BTree of query q;
11             end
12         end
13         for q ∈ Q' do
14             if q.k is the smallest then
15                 V = compute initial voronoi graph;
16             else
17                 V = delete voronoi cells in V;
18                 V = compute unused road network in V;
19             end
20             save voronoi V as DSS of m into BTree on each edge;
21         end
22     end
23 end
```

Gambar 3.9 Algoritma *deletion*: objek o keluar dari sistem.

DSS dihitung, DSS dari objek-objek yang didominasi tersebut disimpan pada BTree pada setiap *edge* untuk setiap *query*.

Proses *deletion* sebagaimana yang terdapat pada Gambar 3.9 memiliki alur yang hampir sama dengan proses *insertion*. Masing-masing *insertion* dan *deletion* memiliki bagian algoritma yang digunakan untuk menghitung ulang setiap objek yang didominasi agar *k-dominant skyline* tetap terbarui. Hal tersebut terdapat pada baris 11-28 pada algoritma *insertion* dan baris 5-22 pada algoritma *deletion*.

Secara umum, algoritma *deletion* bekerja dengan cara penghapus DSS dari semua BTree dari objek *o* yang dihapus kemudian memperbarui *skyline*. Baris 2-4 pada Gambar 3.9 mengiterasi setiap *query* pada *group* dan menghapus setiap DSS dari objek *o* yang dihapus. Cara kerja selanjutnya bisa disamakan dengan algoritma

insertion.

Terkait proses *deletion* yang terdapat pada BTree yang menyimpan hasil komputasi memerlukan catatan khusus bahwa proses *deletion* memerlukan penggabungan objek jika dalam dua segmen memiliki himpunan *k-dominant skyline* yang sama. Sebagai contoh pada BTree awal berikut terdapat objek o_4 yang merupakan *k-dominant skyline* pada jarak 0 hingga 40 dan objek o_3 yang merupakan *k-dominant skyline* dari jarak 30 hingga akhir *edge*.

BTree awal:

- $< 0, \{o_1, o_4\} >$
- $< 15, \{o_1, o_2, o_4\} >$
- $< 30, \{o_1, o_3, o_4\} >$
- $< 40, \{o_1, o_3\} >$

BTree setelah objek o_4 dihapus daris sistem.

- $< 0, \{o_1\} >$
- $< 15, \{o_1, o_2\} >$
- $< 30, \{o_1, o_3\} >$
- $< 40, \{o_1, o_3\} >$

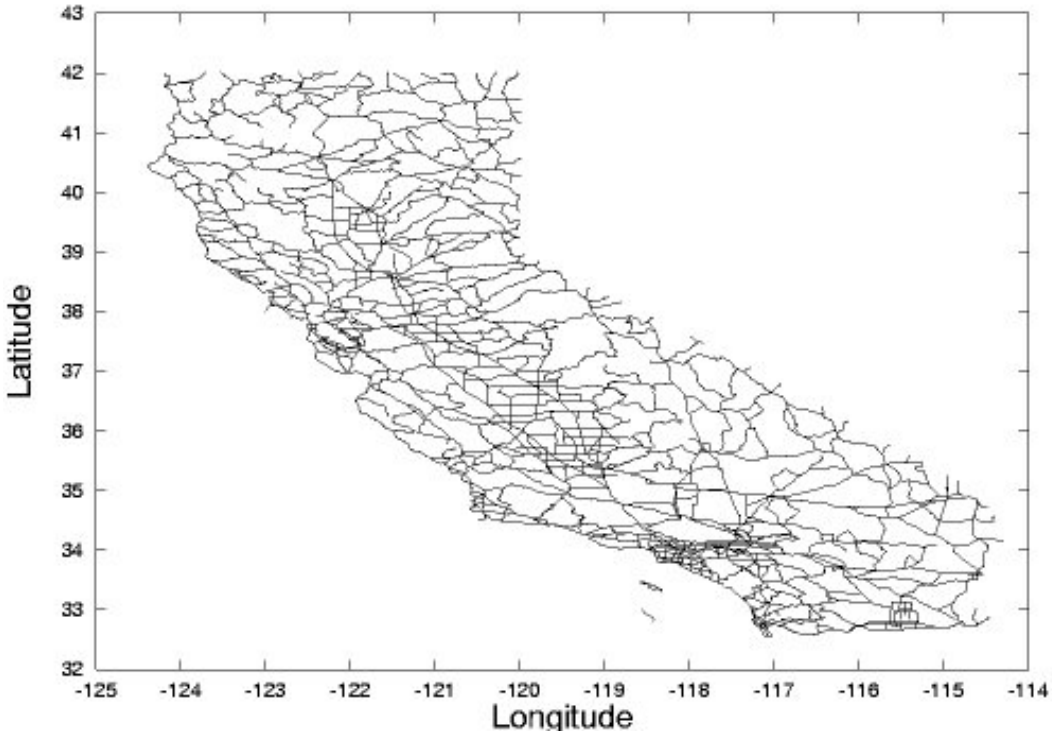
BTree setelah *merging*.

- $< 0, \{o_1\} >$
- $< 15, \{o_1, o_2\} >$
- $< 30, \{o_1, o_3\} >$

Ketika objek o_4 dihapus dari sistem, maka terdapat dua *key* pada BTree yang memiliki *value*, yaitu *key* 30 dan 40. Dengan demikian perlu mekanisme *merging* agar untuk memastikan tidak ada redundansi data. Mekanisme *merging* dijalankan dengan mengiterasi setiap *key* dan mengecek *neighbor*-nya. Jika terdapat *neighbor* yang memiliki *value* yang sama maka *key* dari data yang akhir dihapus dari BTree.

3.4 Pengujian dan Evaluasi

Penulis menguji algoritma dengan mempertimbangkan beberapa faktor yang mempengaruhi efektivitas algoritma, yaitu jumlah objek O , jarak maksimal d_ε, dan jumlah atribut nonspasial D dengan rentang yang disebutkan pada Tabel 3.4. Pengujian dilakukan pada data peta asli California (Li et al., 2005) yang memiliki *node* sejumlah 21048 dan *edge* sejumlah 21692. Peta tersebut memiliki *node* sejumlah dinormalisasi sehingga jarak terjauh objek di sumbu x adalah 100 satuan dan begitu juga dengan sumbu y. Algoritma IKSR dibandingkan dengan algoritma *k-dominant skyline* tanpa ada penggabungan *query* dengan subspace yang sama, penulis menyebutnya dengan algoritma *naïve*. Algoritma *naïve* melakukan komputasi tanpa ada hubungan antara setiap *query*.



Gambar 3.10 Peta jaringan jalan raya California (Li et al., 2005)

Gambar 3.10 menampilkan jaringan raya yang digunakan pada pengujian. Perlu diketahui bahwa tidak setiap *node* pada data set tersebut berarti ada persimpangan. Jarak yang berkelok juga direpresentasikan dalam bentuk *node* agar dapat menyesuaikan dengan kondisi jaringan jalan raya pada aslinya.

Parameter	Rentang
Jumlah objek O	5K, **10K**, 15K, 20K
Jarak maksimal d_ε	3%, **5%**, 7%, 9%
Jumlah atribut nonspasial D	**8**, 10, 12, 14
Jenis Data	***independent***, *correlated*, *anti-correlated*

Tabel 3.4 Skenario pengujian.

Secara *default*, pengujian menggunakan nilai yang ditebalkan. Pengujian hanya mengambil data waktu eksekusi data dalam proses insertion karena fokus dari penelitian ini adalah pemrosesan dengan *continuous*. Waktu eksekusi diukur dengan memasukkan 300 objek kemudian mencari rata-rata dari masing-masing proses *insertion* tersebut. Proses insertion mengkomputasi *k-dominant skyline* terbaru dengan nilai *k* terendah hingga tertinggi dalam sekali komputasi. Nilai *k* terendah adalah $\frac{D}{2} + 1$ dan *k* tertinggi adalah $D - 1$. Penulis menggunakan *default* parameter pengujian sesuai pada Tabel 3.4 karena waktu komputasi yang tidak memberatkan proses pengujian dan sudah mewakili proses pengujian secara umum.

Pengujian menggunakan data *independen*, yaitu data yang memiliki distribusi normal. Untuk mengetaui performa algoritma secara menyeluruh, pengujian juga dilakukan dengan menggunakan data sintetis jenis *correlation* dan *anticorrelation*. Ketiga jenis data tersebut dibuat sesuai dengan cara yang terdapat pada (Börzsönyi et al., 2001).

3.5 Algoritma *Naive*

Dalam pengukuran performa algoritma IKSR, algoritma pembanding diperlukan untuk mengetahui kelebihan dan kekurangan masing-masing algoritma, khususnya IKSR. Dalam penelitian ini penulis membuat sebuah algoritma *naive* yang merupakan metode yang dari segi alur dasar tidak terdapat optimasi yang semaksimal algoritma IKSR.

Algoritma *naive* pada dasarnya merupakan algoritma IKSR yang belum dilakukan *improvement*. *Improvement* yang dimaksud adalah proses optimasi *query* dengan menggabungkan beberapa *query* dalam sekali komputasi sebagaimana yang

terdapat pada algoritma IKSR. Metode perbandingan dominansi objek-objek untuk algoritma *naive* memiliki cara yang sama dengan algoritma IKSR.

```
  Input: the set of objects O in road network G(V, E), the set of queries Q
  Output: k-dominant skyline BTree for each query in each edge
1 for o ∈ O do
2     for q ∈ Q do
3         V = compute initial voronoi graph;
4         save voronoi V as DSS of o into BTree on each edge;
5     end
6 end
```

Gambar 3.11 Algoritma *construction* untuk metode *naive*.

Gambar 3.11 menunjukkan algoritma *construction* untuk pembuatan BTree hasil pada setiap *edge* di setiap *query*. Setiap objek o yang sudah terdapat pada jaringan jalan raya diiterasi. Dalam iterasi tersebut setiap *query* q diiterasi lagi dan komputasi *voronoi graph* dilakukan. Komputasi voronoi dilakukan dengan konsep yang sama dengan algoritma IKSR. Kecuali algoritma *naive* tidak memiliki mekanisme untuk menyimpan data objek yang pada nilai *k* lain mendominasi atau didominasi oleh objek lain. Komputasi keseluruhan *voronoi* pada algoritma *naive* hanya dijalankan sekali dan digunakan sekali setiap *query*. Hal ini berbeda dengan konsep algoritma IKSR yang menjalankan keseluruhan *voronoi* dan menyimpan informasi yang kemudian digunakan pada komputasi *query* lain yang memiliki *subspace* yang sama.

Gambar 3.12 menampilkan algoritma *insertion* yaitu masuknya objek pada sistem. Sebagaimana algoritma *naive*, komputasi algoritma *insertion* didahului dengan komputasi algoritma *construction* karena memerlukan pembuatan BTree untuk penyimpanan hasil *k-dominant skyline* pada tiap *query*. Baris 1 merupakan iterasi dari setiap *query* dari himpunan *query*. Iterasi ini dilakukan tanpa adanya *grouping* dan *sorting* sebagaimana algoritma IKSR karena algoritma *naive* tidak memiliki optimasi untuk penggabungan *query*. Selanjutnya, Baris 2 melakukan komputasi *voronoi* dan menyimpan DSS untuk objek *o* pada *query* q. Baris 3, DSS tersebut disimpan pada BTree sesuai *edge* masing-masing.

```
Input: object o in inserted to G(V, E)
Output: updated k-dominant skyline in G(V, E)
for q ∈ Q do
    V = compute initial voronoi graph;
    save voronoi V as DSS of o into BTree on each edge;
    for each object m ∈ Dominate(o) do
        remove DSS of m from related BTree of query q;
        V = compute initial voronoi graph;
        save voronoi V as DSS of m into BTree on each edge;
    end
end
```

Gambar 3.12 Algoritma *insertion* untuk metode *naive*

```
Input: object o is deleted from G(V, E)
Output: updated k-dominant skyline in G(V, E)
for q ∈ Q do
    remove DSS of o from related BTree of query q;
    for each object m ∈ Dominate(o) do
        remove DSS of m from related BTree of query q;
        V = compute initial voronoi graph;
        save voronoi V as DSS of m into BTree on each edge;
    end
end
```

Gambar 3.13 Algoritma *deletion*: objek o keluar dari sistem.

Kemudian, baris 4, algoritma mencari objek yang didominasi sesuai *subspace* dari q.**D** dan nilai k $q.k$ dengan jarak kurang dari $2 \times d_{\varepsilon}$. Pada baris 5, DSS dari objek yang didominasi tersebut dihapus dari semua BTree pada semua *edge* karena memerlukan komputasi ulang agar tidak terdapat *false positive* dan redundansi pada BTree. Baris 16 melakukan komputasi pembuatan *voronoi graph* dan menyimpan *voronoi cell* dari objek yang didominasi tersebut. Terakhir, objek yang didominasi tersebut disimpan pada DSS pada BTree yang sesuai.

Gambar 3.13 merupakan algoritma *deletion* yang digunakan untuk menghapus objek dari sistem dan melakukan pembaruan *k-dominant skyline*. Cara kerja algoritma tersebut hampir sama seperti algoritma *insertion*. Perbedaannya adalah ketika objek keluar, DSS objek dihapus dari BTree. Sedangkan algoritma *insertion* mengkomputasi DSS dari objek tersebut dan menyimpan pada BTree.

Halaman ini sengaja dikosongkan

BAB 4
HASIL DAN PEMBAHASAN

Bab ini menjelaskan mengenai proses implementasi berdasarkan rancangan serta skenario yang telah disampaikan pada bab sebelumnya. Penjelasan pada bab ini meliputi lingkungan pengujian berdasarkan perangkat keras dan perangkat lunak, implementasi jenis data, serta analisis hasil pengujian. Analisis hasil dilakukan dengan membandingkan performa algoritma IKSR dan *naive* dalam waktu komputasi.

4.1 Lingkungan Pengujian

Tabel 4.1 menampilkan spesifikasi perangkat keras dan lunak yang digunakan untuk mengimplementasikan metode pada tesis ini. Perangkat keras yang dimaksud adalah RAM, prosesor, dan perangkat penyimpanan. Perangkat lunak yang dimaksud adalah sistem operasi dan bahasa pemrograman yang digunakan.

Komponen	**Spesifikasi**
Sistem Operasi	Pop!_OS 19.04 64-bit
RAM	16 GB
Prosesor	Intel Core i7-1050U
Bahasa Pemrograman	Rust 1.43
Penyimpanan	512 GB PCIe® NVMe™ M.2 SSD

Tabel 4.1 Skenario pengujian.

4.2 Jenis Data

Tiga jenis data sintetis digunakan pada pengujian, yaitu *independent*, *correlated*, dan *anti-correlated*. Pembuatan ketiga jenis data ini menggunakan fungsi-fungsi *standard library* yang terdapat pada Python versi 3.6.

4.2.1 Independent

Data *independent* merupakan data yang dibuat secara acak/*random*. Setiap atribut dan nilai yang terdapat pada objek di data *independent* tidak memiliki keterkaitan. Objek secara acak diletakkan pada *edge* yang memiliki ID 1 hingga 21692 sesuai jumlah *edge* yang terdapat pada peta California (Li et al., 2005). Nilai setiap atribut pada objek memiliki tipe *float* dengan rentang 0.0 hingga 1.0. Data dibuat menggunakan fungsi `random()` pada Python.

4.2.2 Correlated

Data *correlated* merupakan data sintetis yang memiliki persebaran yang saling mendominasi. Setiap atribut pada data *correlated* memiliki nilai yang tidak terpaut jauh. Dalam pengujian ini penulis menggunakan variasi 0.2 antar atribut sehingga distribusi data dapat diilustrasikan yang terdapat pada (Börzsönyi et al., 2001). Objek tetap ditempatkan pada lokasi yang acak pada *edge* sebagaimana yang terdapat pada data *independent*. Lokasi tetap diacak karena pengujian ini fokus untuk mengkomputasi objek *k-dominant skyline* dan bukan dalam bentuk lokasi yang berarti menempatkan objek pada suatu konsentrasi tertentu. Objek pada data *corrlated* memiliki hubungan dominansi yang besar sehingga menjadikan *skyline* semakin sedikit.

4.2.3 Anti-Correlated

Data *anti-correlated* dibuat dengan tujuan mengurangi dominansi antarobjek. Data ini dibuat dengan memberikan nilai atribut yang saling berkebalikan sehingga objek memiliki kemungkinan dominansi yang sedikit. Objek tetap diletakkan pada tempat yang acak pada *edge*.

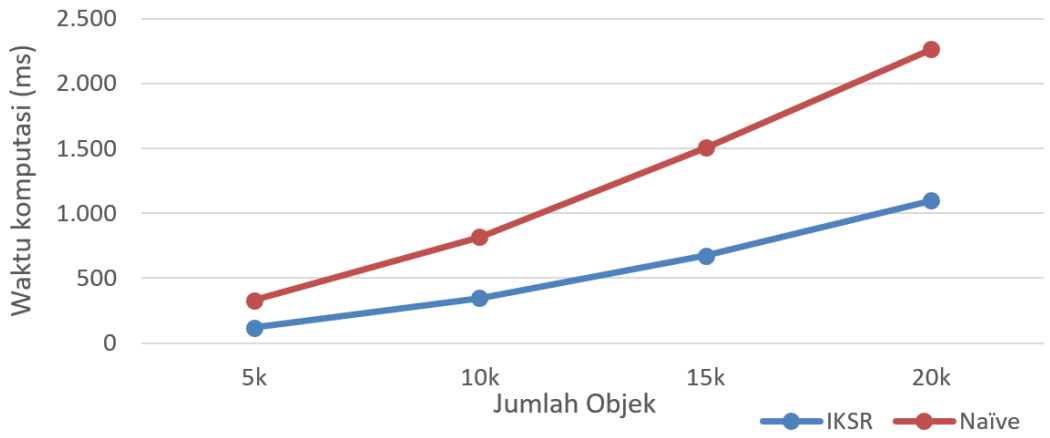
Penulis menggunakan bahasa pemrogramana Rust karena Rust merupakan *low-level system language* yang memungkinkan pengguna mengatur penggunaan memori sesuai kebutuhan secara manual. Rust bersifat *type-safe* yang memungkinkan kesalahan umum dapat dideteksi pada proses kompilasi. Sebagai bentuk keamanan, Rust memiliki konsep *ownership* yang menjadikan setiap data hanya dimiliki oleh satu entitas dalam satu waktu dan konsep *lifetime* yang

mendefinisikan *scope* dari suatu data dianggap valid (Jung et al., 2017).

4.3 Analisis Hasil

Empat jenis evaluasi dilakukan untuk mengukur performa dari metode IKSR yang dibandingkan dengan algoritma *naïve*. Dari keempat evaluasi tersebut, tiga evaluasi pertama ditujukan untuk membandingkan performa algoritma IKSR dan algoritma *naive*. Satu evaluasi terakhir ditujukan untuk mengetahui pengaruh ketiga jenis data terhadap performa algoritma IKSR.

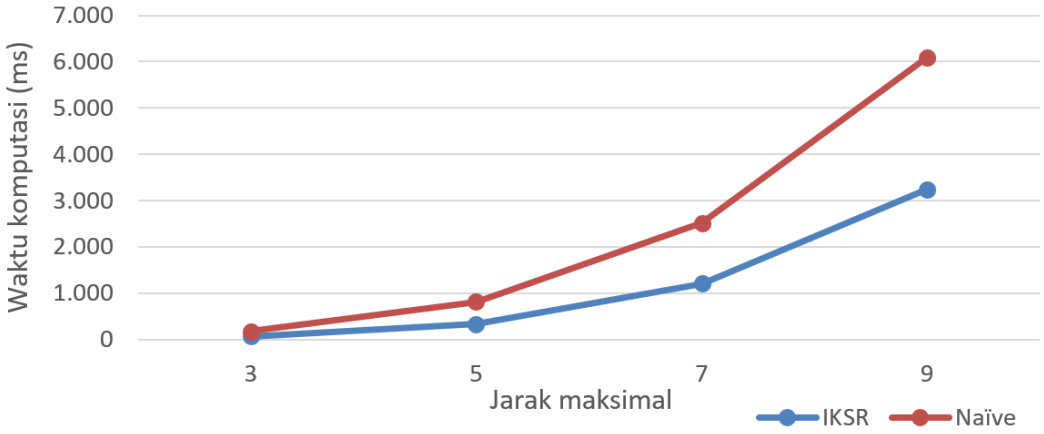
4.3.1 Pengaruh Jumlah Objek Terhadap Waktu Komputasi



Gambar 4.1 Grafik pengaruh jumlah objek terhadap waktu komputasi dalam *miliseconds*

Gambar 4.1 merupakan grafik hubungan antara jumlah objek dengan waktu komputasi. Kenaikan waktu komputasi cenderung bersifat linear pada kedua metode. Metode IKSR maupun *naïve* mengalami kenaikan waktu komputasi karena semakin banyak objek yang didominasi, dengan demikian objek yang didominasi tersebut perlu dikomputasi ulang karena adanya perubahan *skyline scope*-nya. IKSR memiliki performa yang lebih baik dibandingkan *naïve*. Hal tersebut dikarenakan IKSR menghitung *voronoi cell* untuk setiap objek hanya pada komputasi *k* terkecil. Komputasi *k* yang lebih besar hanya perlu menghapus objek yang tidak diperlukan dan komputasi area yang kosong. Di sisi lain, algoritma *naïve* menghitung *voronoi cell* secara keseluruhan pada semua komputasi *k*.

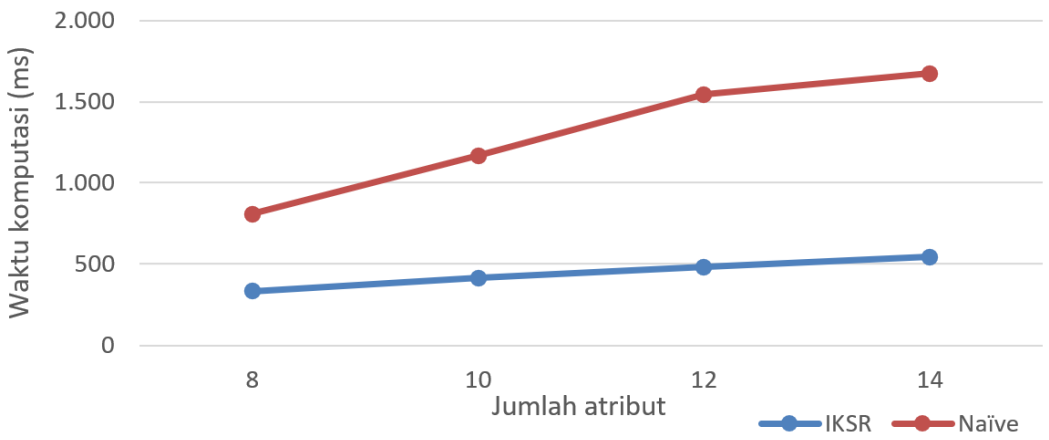
4.3.2 Pengaruh Jarak Maksimal d_ε Terhadap Waktu Komputasi



Gambar 4.2 Grafik pengaruh jarak maksimal/d_ε terhadap waktu komputasi.

Hubungan antara jarak maksimal/d_ε dengan waktu komputasi digambarkan pada Gambar 4.2. Grafik waktu komputasi pada kedua algoritma membentuk kurva kuadratis karena kenaikan jarak secara linear menjadikan luas dari cakupan bertambah secara kuadratis sehingga objek yang didominansi juga bertambah secara kuadratis. Secara umum, waktu komputasi algoritma *naïve* dua kali lebih lama dibandingkan IKSR. Perubahan jarak maksimal pada pengujian mengakibatkan komputasi semakin lama dalam dua cara. Pertama, karena jarak semakin jauh berarti algoritma perlu melakukan *traverse* yang lebih lama. Kedua, jarak yang lebih jauh berarti objek dapat menjangkau objek lain yang lebih jauh untuk didominasi. Kedua hal tersebut menjadikan objek melakukan komputasi *voronoi* lebih lama karena harus membandingkan objek yang lebih banyak. Perbandingan objek yang banyak menghasilkan hubungan dominansi antarobjek semakin banyak sehingga *voronoi cell* yang dibuat semkain banyak. Pada tahap komputasi objek yang masuk tidak banyak mempengaruhi karena hanya perlu memerlukan sekali komputasi voronoi. Bertambahnya jumlah objek yang didominasi menjadikan proses komputasi semakin lama karena algoritma harus memperbarui DSS dari setiap objek yang didominasi dengan menghapus semua DSS objek tersebut dan menghitung ulang DSS dengan membuat *voronoi* dari awal.

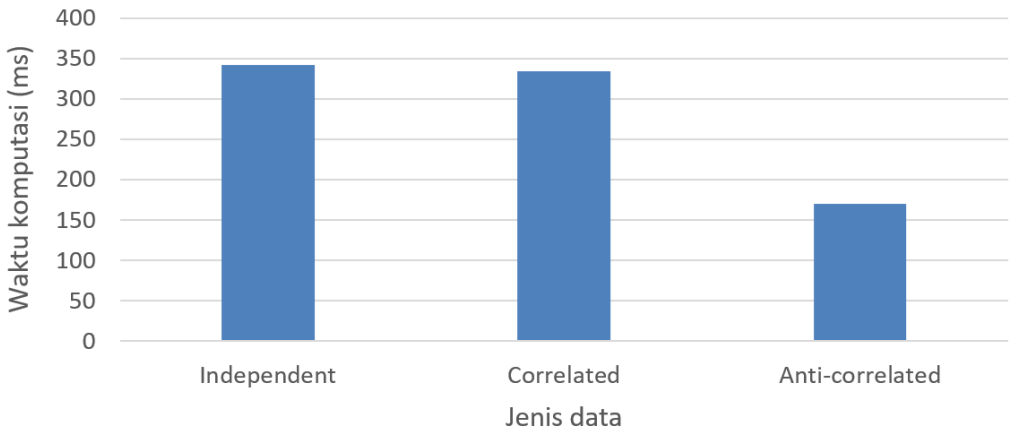
4.3.3 Pengaruh Jumlah Atribut Terhadap Waktu Komputasi



Gambar 4.3 Grafik pengaruh jumlah atribut terhadap waktu komputasi.

Pengaruh jumlah atribut terhadap waktu komputasi terdapat pada Gambar 4.3. Penambahan dua atribut tidak menaikkan waktu komputasi secara signifikan dan bersifat linear, hal tersebut dikarenakan komputasi untuk perbandingan dominansi antarobjek hanya dilakukan sekali di sebelum komputasi *voronoi cell*. *Cost* untuk penghitungan dominansi lebih kecil dibandingkan *cost* untuk melakukan traverse pada jaringan jalan raya sehingga tidak memberikan efek yang signifikan. Waktu komputasi IKSR dua hingga tiga kali lebih cepat dibandingkan *naïve*.

4.3.4 Pengaruh Jenis Data Terhadap Waktu Komputasi



Gambar 4.4 Grafik pengaruh jumlah atribut terhadap waktu komputasi.

Gambar 4.4 menampilkan pengaruh jenis data terhadap waktu komputasi. Waktu komputasi data *independent* dan *correlated* tidak memiliki perbedaan yang signifikan karena objek *independent* cenderung terdapat dominansi pada nilai k yang bervariasi dan objek *correlated* juga terdapat dominansi dengan nilai k yang mendekati D. Data *anti-correlated* memiliki perbedaan yang signifikan karena terdapat sedikit relasi dominasi sehingga tidak banyak *voronoi cell* yang perlu dikomputasi. Dengan demikian, objek pada data *anti-correlated* tidak memiliki *voronoi cell* yang lebih banyak dibandingkan data pada *independent* dan *correlated*. Data *anti-correlated*

4.3.5 Kesimpulan Analisis

Berdasarkan hasil pengujian yang telah dilakukan, algoritma IKSR memiliki performa yang lebih baik pada semua evaluasi. Berdasarkan ketiga evaluasi pertama, kesimpulan dapat diambil bahwa algoritma IKSR memiliki performa dua hingga tiga kali lebih cepat dibandingkan algoritma *naive*. Hal tersebut dikarenakan adanya penggabungan *query* yang memiliki dimensi yang sama. Pada pengujian jenis data, data *anti-correlated* memberikan pengaruh yang signifikan berupa waktu komputasi yang dua kali lebih cepat dibandingkan data *independent* dan *correlated*. Hal tersebut dikarenakan data *anti-correlated*

memang didesain agar memiliki dominansi yang sedikit sehingga komputasi yang diperlukan tidak sebanyak data *independent* dan *correlated* karena sedikitnya objek yang didominasi berarti komputasi *voronoi* juga berkurang.

Halaman ini sengaja dikosongkan

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan uraian pada bab sebelumnya, tesis ini dapat disimpulkan dengan:

1. Metode IKSR diusulkan untuk menyelesaikan komputasi *k-dominant skyline* di jaringan jalan raya secara *continuous*. Algoritma ini menggabungkan *query* yang beroperasi pada *subspace* **D** dan himpunan objek *O* yang sama dengan nilai *k* yang berbeda dengan cara mengkomputasi mulai dari *k* yang terkecil ke terbesar. Komputasi *k* yang terkecil mengkomputasi sebagian yang diperlukan pada *k* yang lebih besar sehingga komputasi *voronoi cell* tidak dilakukan berulang-ulang.

2. Performa algoritma IKSR diuji dengan membandingkan algoritma *naive* untuk mengetahui pengaruh jumlah objek, jarak maksimal d_ε, jumlah atribut, dan jenis data terhadap waktu komputasi. Semua jenis pengujian menyatakan bahwa algoritma IKSR memiliki waktu yang lebih cepat dibandingkan algoritma *naive* dengan rentang dua hingga tiga kali lebih cepat.

5.2 Saran

Pada dasarnya algoritma IKSR menggabungkan beberapa *query* dalam satu *subspace* dalam satu komputasi. Pengguna mungkin mencari objek-objek yang unggul pada *subspace* yang berbeda untuk mendapatkan wawasan yang lebih luas terkait objek yang terbaik di bidangnya masing-masing. Kasus tersebut tidak dapat diselesaikan dengan algoritma IKSR dan memerlukan algoritma yang bisa mengkomputasi *k-dominant skyline* pada berbagai *subspace* dalam satu komputasi.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

Börzsönyi, S., Kossmann, D. and Stocker, K. (2001), 'The skyline operator', *Proceedings 17th International Conference on Data Engineering* pp. 421–430.

Chan, C.-Y., Jagadish, H. V., Tan, K.-L., Tung, A. K. H. and Zhang, Z. (2006), Finding k-dominant skylines in high dimensional space, *in* 'SIGMOD '06'.

Erwig, M. (2000), 'The graph voronoi diagram with applications', *Networks* **36**(3), 156–163.

Fu, X., Miao, X., Xu, J. and Gao, Y. (2017), 'Continuous range-based skyline queries in road networks', *World Wide Web* **20**, 1443–1467.

Huang, Y.-K., Chang, C.-H. and Lee, C. (2012), 'Continuous distance-based skyline queries in road networks', *Inf. Syst.* **37**, 611–633.

Jung, R., Jourdan, J.-H., Krebbers, R. and Dreyer, D. (2017), 'Rustbelt: securing the foundations of the rust programming language', *Proc. ACM Program. Lang.* **2**, 66:1–66:34.

Kontaki, M., Papadopoulos, A. N. and Manolopoulos, Y. (2008), Continuous k-dominant skyline computation on multidimensional data streams, *in* 'SAC '08'.

Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G. and Teng, S.-H. (2005), On trip planning queries in spatial databases, *in* 'SSTD'.

Peng, Y.-W. and Chen, W.-M. (2019), 'Parallel k-dominant skyline queries in high-dimensional datasets', *Inf. Sci.* **496**, 538–552.

Siddique, M. A., Tian, H. and Morimoto, Y. (2015), 'k-dominant skyline query computation in mapreduce environment', *IEICE Trans. Inf. Syst.* **98-D**, 1027–1034.

Halaman ini sengaja dikosongkan