TUGAS AKHIR – KI141502

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

Cahya Setya Adhi
NRP 05111440000049

Dosen Pembimbing
Royyana M Ijtihadie, S.Kom.,M.Kom.,Ph.D
Bagus Jati Santoso, S.Kom, Ph.D.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR – KI141502

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

Cahya Setya Adhi
NRP 05111440000049

Dosen Pembimbing
Royyana M Ijtihadie, S.Kom.,M.Kom.,Ph.D
Bagus Jati Santoso, S.Kom, Ph.D.

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

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – KI141502

DESIGN AND IMPLEMENTATION OF SKYLINE QUERY PROCESSING ON UNCERTAIN DATA BY STATIC QUERY POINT AND DYNAMIC OBJECTS ON ROAD NETWORK

Cahya Setya Adhi
NRP 05111440000049

Advisor
Royyana M Ijtihadie, S.Kom.,M.Kom.,Ph.D
Bagus Jati Santoso, S.Kom, Ph.D.

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

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

Desain Dan Implementasi Aplikasi Pengolahan *Skyline Query* Pada *Dynamic Uncertain Data* Oleh Titik Tidak Bergerak Dan Objek 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 :
Cahya Setya Adhi
NRP : 05111440000049

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 *Dynamic Uncertain Data* Oleh Titik Tidak Bergerak Dan Objek Bergerak Pada Jaringan Jalan Raya

Nama Mahasiswa : Cahya Setya Adhi
NRP : 05111440000049
Jurusan : Informatika FTIK-ITS
Dosen Pembimbing 1 : Royyana M Ijtihadie, S.Kom.,M.Kom.,Ph.D
Dosen Pembimbing 2 : Bagus Jati Santoso, S.Kom, Ph.D

ABSTRAK

Skyline adalah sekumpulan objek yang tidak didominasi oleh objek lain. Misalkan suatu objek mempunyai nilai atribut sebanyak n, maka suatu objek dikatakan mendominasi apabila dari n atribut yang dimiliki tidak ada atribut yang lebih buruk dari atribut objek lain dan minimal ada satu nilai dari atribut yang lebih baik dibandingkan dengan objek lain.

Seiring perkembangan teknologi berbasis lokasi, *skyline* dapat digunakan terhadap objek yang berada di jalan raya. Sebagai contohnya kita dapat menerapkan *skyline query* pada objek daring. Perkembangan ojek daring mengakibatkan munculnya kejahatan terhadap penumpang oleh oknum pengemudi. Dengan *skyline query* penumpang dapat mencari pengemudi ojek dengan penilaian terbaik untuk menjamin rasa aman.

Tujuan dari pembuatan tugas akhir ini adalah mendesain algoritma untuk melakukan analisis dan mendesain struktur data untuk pengolahan *skyline query* pada *dynamic uncertain data* oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya. Pada tugas akhir ini akan diperkenalkan algoritma berbasis CNO dan algoritma *naive*. Hasil pengujian menunjukkan bahwa algoritma CNO memiliki performa yang lebih baik 100 kali lipat dibandingkan dengan metode *naive.*

Kata kunci: ***Road Network , Skyline, Uncertain Data***

[Halaman ini sengaja dikosongkan]

Design And Implementation Of Skyline Query Processing On Uncertain Data By Static Query Point And Dynamic Objects On Road Network

Student Name	: Cahya Setya Adhi
Student ID	: 05111340000097
Major	: Department of Informatics Faculty of ICT-ITS
Advisor 1	: Royyana M Ijtihadie, S.Kom.,M.Kom.,Ph.D
Advisor 2	: Bagus Jati Santoso, S.Kom, Ph.D

ABSTRACT

Skyline is a set of object that not dominated by other object. If a set of objects have attributes, then an object dominating other object if the object have equals or better than other object and at least has one attribute better than other object.

Skyline can be combined with the location based object. For the example skyline query can be applied to ojek online. The development of online riding service causing crime by individualy driver. With skyline query passenger can choose driver with good rating for ensuring sense of security.

The goals of this research is to design algorithm, data structure and then analyze the algorithm to process skyline query of dynamic uncertain data by static query point and moving objects in road networks. This research will introduce CNO algorithm and naive algorithm. The results shows that CNO algorithm has better performance 100 times than naive approach.

Kata kunci: ***Road Network , Skyline, Uncertain Data***

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur kehadirat Allah SWT karena atas limpahan karunia dan rahmatnya tugas akhir dengan judul **"Desain Dan Implementasi Aplikasi Pengolahan Skyline Query Pada Dynamic Uncertain Data Oleh Titik Tidak Bergerak Dan Objek Bergerak Pada Jaringan Jalan Raya"** ini dapat terselesaikan.

Terimakasih yang sebesar – besarnya saya ucapkan kepada berbagai pihak yang telah mendukung dan membantu saya dalam pengerjaan tugas akhir ini, antara lain :

1. Orang Tua dan keluarga saya yang selalu mendo'akan dan senantiasa memotivasi dalam menyelesaikan pengerjaan tugas akhir ini.
2. Bapak Bagus Jati Santoso, S.Kom, Ph.D. selaku dosen pembimbing 2.
3. Bapak Royyana M Ijtihadie, S.Kom.,M.Kom.,Ph.D, selaku dosen pembimbing 1.
4. Saudara Muhsin Bayu Aji F. dan saudara Buthoro Kunto R yang selalu memberikan dukungan dan motivasi untuk.
5. Rekan-rekan bimbingan yang selalu bergotong royong agar tugas akhir ini dapat selesai..
6. Rekan-rekan indekos yang selalu mendukung dan memberikan lingkungan yang kondusif selama pengerjaan tugas akhir ini.
7. Seluruh keluarga Mahasiswa Teknik Informatika angkatan 2014.
8. Rekan-rekan seperjuangan Ikemas Surabaya yang selalu mendukung saya bahkan di luar organisasi.
9. Dan seluruh pihak yang telah membantu dan mendukung.

Semoga tugas akhir ini dapat dimanfaatkan dengan sebagai mestinya, dan bermanfaat bahkan setelah tugas akhir ini selesai untuk Almamater, masyarakat dan bangsa.

Surabaya, 6 Juni 2018
Penulis

Cahya Setya Adhi
05111440000049

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman ini sengaja dikosongkan

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1. Latar Belakang

Akhir-akhir ini banyak spasial query berbasis lokasi telah diusulkan dan diteliti, seperti range queries, nearest neighbor queries [1], skyline queris [2] dan lain-lain. Skyline query adalah query yang menerima data objek-objek yang tidak didominasi oleh objek lain. Skyline query pada jaringan jalan raya meliputi atribut spasial dan atribut yang bukan spasial, yang memenuhi kebutuhan pada aplikasi berbasis lokasi seperti kendali lalu lintas, membantu pengobatan, dan saran periklanan [3].

Penelitian yang telah dilaksanakan hingga saat ini adalah skyline query untuk titik tidak bergerak dan objek bergerak pada jaringan jalan raya dengan data yang statis dan diskrit [3]. Sebagai contoh ketika seseorang yang mengirim query pada aplikasi berbasis lokasi untuk menemukan taksi dengan parameter harga yang murah dan memiliki tempat duduk yang banyak atau kita sebut taksi tersebut adalah skyline. Penelitian sebelumnya tidak bisa diaplikasikan pada objek-objek yang dalam hal ini direpresentasikan sebagai taksi memiliki data yang tidak diskrit. Sebagai contoh dari permasalahan tersebut adalah apabila suatu taksi tidak memberikan data pasti mengenai harga seperti Rp 100.000,00 atau Rp 200.000,00 melainkan rentang harga seperti Rp 100.000,00 – Rp 200.000,00. Penelitian sebelumnya juga tidak bisa diaplikasikan pada permasalahan apabila objek-objek target query yang direpresentasikan sebagai taksi adalah objek yang dinamis. Dinamis yang dimaksud adalah dimungkinkan muncul objek-objek baru ataupun suatu objek dimungkinkan untuk menghilang.

Tugas akhir ini diharapkan dapat menemukan solusi untuk melengkapi penelitian sebelumnya yang tidak dapat diaplikasikan pada objek yang memiliki data yang tidak diskrit. Tugas akhir ini juga diharapkan dapat menambahkan fungsional terbaru berkatian dengan objek-objek yang dinamis. Diharapkan solusi yang dihasilkan dari tugas akhir ini memiliki akurasi yang tinggi, waktu eksekusi yang rendah dan ridak memakan memori yang banyak.

1.2. Perumusan Masalah

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

1. Bagaimana menganalisis dan menentukan struktur data dan algoritma untuk desain dan implementasi pengolahan *skyline query* pada *dynamic uncertain data* oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya?
2. Bagaimana cara mengetahui biaya komputasi dan penyimpanan pada desain dan implementasi pengolahan *skyline query* pada *dynamic uncertain data* oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya?
3. Faktor apa yang mempengaruhi performa algoritma untuk desain dan implementasi pengolahan *skyline query* pada *dynamic uncertain data* oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya?

1.3. Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan antara lain:

1. Implementasi algoritma dan pemrosesan menggunakan python.
2. *Dataset* yang digunakan adalah jaringan jalan raya sintetis.
3. Pergerakan objek tidak memiliki pergerakan yang acak dan hanya menuju satu arah.
4. Kecepatan objek konstan.

1.4. Tujuan Penelitian

1. Melakukan analisis dan mendesain struktur data dan algoritma untuk pengolahan *skyline* query pada dynamic uncertain data oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya.
2. Menemukan metode optimal untuk mengurangi biaya komputasi dan penyimpanan pada desain dan implementasi pengolahan *skyline* query pada dynamic uncertain data oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya.

1.5. Manfaat Penelitian

Mengetahui struktur data dan algoritma yang tepat untuk pengolahan *skyline query* pada dynamic uncertain data oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya.

1.6. Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu:

1.6.1. Penyusunan Proposal Tugas Akhir

Tahap pertama dalam penyusunan tugas akhir ini adalah penyusunan proposal tugas akhir. Proposal tugas akhir ini memberikan gambaran mengenai masalah yang ingin diselesaikan dan metode-metode yang akan digunakan dan metode baru yang diajukan.Studi Literatur

1.6.2. Studi Literatur

Pada tahap studi literatur akan dipelajari referensi-referensi yang berkaitan dengan tugas akhir. Referensi dapat diambil dari tugas akhir, internet, paper, jurnal, maupun konferensi.

1.6.3. Analisis dan Desain Perangkat Lunak

Sebelum memulai implementasi akan dianalisa mengenai permasalahan yang akan diselesaikan lalu dilanjutkan dengan pembuatan solusi yang meliputi penentuan struktur data dan algoritma baru dan dilanjutkan dengan implementasi.

1.6.4. Implementasi Perangkat Lunak

Tahap implementasi meliputi implementasi algoritma dan struktur data pada perangkat lunak yang telah didukung oleh hasil analisis dan desain pada tahap sebelumnya. Implementasi ini dilakukan dengan menggunakan python.

1.6.5. Pengujian dan Evaluasi

Pengujian dalam algoritma ini berfokus pada keberhasilan dalam algoritma pengolahan *skyline query* pada dynamic uncertain data oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya.

1.7. Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan Tugas Akhir ini.

Bab III Analisis dan Perancangan Sistem

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan algoritma dan struktur data.

Bab IV Implementasi

Bab ini berisi implementasi dari perancangan perangkat lunak.

Bab V Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian objektif untuk mengetahui hasil dari implementasi terhadap metode lain.

Bab VI Kesimpulan dan Saran

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar istilah yang penting pada sistem ini.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI

Bab ini menjelaskan tentang tinjauan pustaka yang menjadi dasar pembuatan tugas akhir. Penjelasan secara khusus masing-masing tinjauan pustaka dapat dilihat pada masing-masing subbab berikut ini.

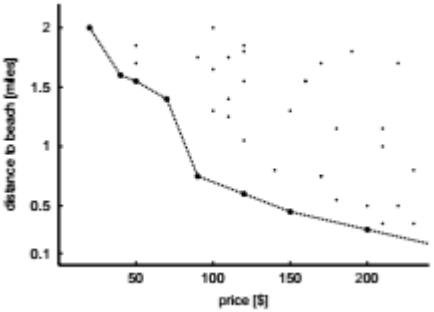
2.1. *Skyline*

Skyline adalah daftar objek yang tidak didominasi oleh objek lain. Apabila sekumpulan objek memiliki n atribut, suatu objek dikatakan mendominasi objek lain apabila n atribut objek tersebut tidak lebih buruk dari n atribut objek lain dan paling tidak ada satu atribut yang lebih baik dari atribut objek lain. Sebagai contoh apabil a terdapat objek A memiliki 3 atribut dengan nilai masing-masing yaitu 5, 6, dan 7 atau dinotasikan dengan A=[5,6,7] dan terdapat objek B juga memiliki 3 atribut dengan nilai 5, 6, dan 6 atau dinotasikan dengan B=[5,6,6]. Dengan asumsi bahwa semakin besar nilai atribut berarti semakin baik, maka dapat dikatakan bahwa objek A mendominasi objek B karena nilai atribut pertama, kedua, maupun ketiga dari objek A tidak lebih buruk dari objek B, dan paling tidak terdapat satu atribut dari objek A yang lebih baik dari atribut objek B yaitu atribut ketiga dimana atribut ketiga objek A memiliki nilai 7 dan atribut ketiga objek B memiliki nilai 6. Objek A dapat dikatakan sebagai *skyline* karena objek A tidak didominasi objek lain sedangkan B tidak menjadi *skyline* karena didominasi oleh objek A. Dengan demikian tidak peduli sebaik apapun nilai n-1 atribut suatu objek apabila atribut ke-n memiliki nilai lebih buruk dari objek lain, objek tersebut tidak bisa dikatakan *skyline*. Sedangkan apabila objek A dan objek B tidak saling mendominasi, maka objek A dan objek B adalah *skyline.*

Misal seseorang ingin berlibur ke suatu tempat dan dia mencari hotel yang murah dan terletak dekat pantai. Dua aspek tersebut saling berlawanan dimana hotel dekat pantai pasti lebih mahal. Sistem basis data pada agen perjalanan tidak bisa

menentukan hotel terbaik untuk orang tersebut, tapi paling tidak dapat menunjukkan pada orang tersebut hotel-hotel yang menarik. Hotel-hotel tersebut tidak lebih buruk dari hotel yang lain pada kedua aspek tersebut. Kita dapat katakan hotel-hotel tersebut sebagai *skyline*. Dari *skyline* orang tersebut dapat membuat keputusan dengan mempertimbangkan harga dan jarak hotel ke pantai sesuai dengan kondisi kita. [1]

Contoh aplikasi skyline yang lain adalah pada agen perjalanan. Pengguna aplikasi agen perjalanan dapat menemukan rekomendasi tujuan terbaik dengan *skyline*. Namun demikian *skyline* juga dapat diaplikasikan pada bidang lain, contohnya adalah merekrut pegawai terbaik dengan gaji terendah dan menemukan rumah makan dengan menu terbanyak dan ulasan terbaik. Skyline juga dapat digambarkan dalam bentuk grafik(Gambar 2.1)



Gambar 2.1 *Skyline* dari Hotel [2]

Pada gambar 9.1, sumbu y merepresentasikan jarak hotel dengan pantai dan sumbu x merepresentasikan harga hotel. Pada gambar di atas diasumsikan bahwa semakin kecil harga dan semakin dekat jarak menunjukkan semakin baik hotel. Titik-titik yang dihubungkan oleh garis menunjukkan bahwa titik-titik tersebut mendominasi titik-titik yang tidak dihubungkan oleh garis karena titik-titik yang dihubungkan garis memiliki jarak terdekat dengan pantai dan memiliki harga yang lebih murah dibandingkan dengan titik-titik yang tidak dihubungkan oleh garis.

2.2. *Uncertain Data*

Pada era saat ini, banyak teknologi dikembangakan untuk menyimpan dan merekam data yang besar secara terus menerus. Pada beberapa kasus, data mungkin memiliki *error* atau mungkin tidak lengkap. Data tersebut biasa disebut dengan *uncertain data*.

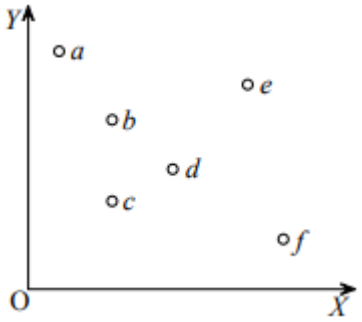
Contoh dari *uncertain data* adalah data dari sensor biasanya banyak menghasilkan *uncertain data* [3]. Dapat kita simpulkan bahwa uncertain data adalah data yang memiliki *error* ataupun tidak lengkap. Contoh yang lain adalah data yang berkaitan dengan tubuh manusia. Kita tidak dapat merepresentasikan data-data tersebut dengan angka pasti. Ketika kita merepresentasikan *uncertain data* dengan angka pasti, akan menyebabkan tiga *uncertainity bug* antara lain 1) Tidak menghiraukan error pada estimasi. 2) Komputasi melibatkan error tersebut. 3) pertanyaan ya/tidak pada data yang memiliki probabilitas dapat menimbulkan false positive dan negative [4].

Contoh lain lagi adalah pada kasus pemilihan pemain terbaik di NBA. Sebagai contoh seorang pemain basket akan dinilai secara multikriteria(contoh: *rebound*, *assist*, dan lain-lain). Idealnya akan dipilih pemain yang baik dari segala aspek. Sayangnya pemain yang sedemikian tidak ada. Pada pemilihan pemain terbaik NBA, penilaian dihitung dalam kurun waktu satu tahun. Dalam kurun waktu satu tahun, setiap pemain pasti memiliki performa yang berbeda setiap bermain. Cara tradisional adalah merepresentasikan atribut setiap pemain dengan fungsi agregasi lalu perhitungan *skyline* akan dilakukan menggunakan nilai tersebut. Namun sayangnya fungsi agregasi tidak dapat mewakili distribusi performa setiap pemain.

Contoh lain yang diangkat pada penelitian tugas akhir ini adalah ojek daring. Setiap ojek daring dapat memiliki nilai atau ulasan yang bermacam-macam dari para penumpangnya. Ada kalanya seorang pengemudi ojek daring memiliki penilaian yang bagus, namun ada kalanya seorang pengemudi ojek daring memiliki penilaian yang tidak bagus. Adakalanya seorang pengemudi

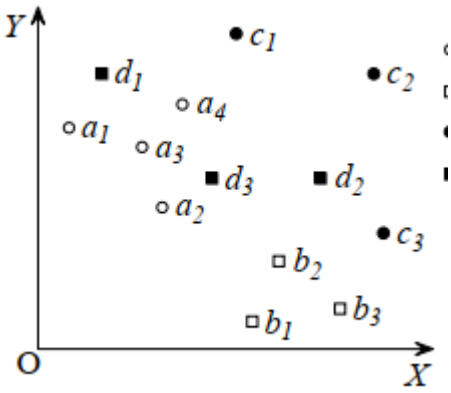
mendapatkan penilaian bagus di hari tertentu, namun mendapatkan penilaian yang tidak bagus keesokan harinya.

Pada awalnya skyline digunakan pada certain data. Sebagai contoh apabila seorang pengemudi ojek daring diwakili oleh nilai dan ulasan maka dapat dinotasikan dengan P=[nilai, ulasan]. Jika kita gambarkan pada grafik dua dimensi, maka setiap pengemudi dapat diwakili oleh sebuah titik seperti pada Gambar 2.2. Pada Gambar 2.2 sumbu x mewakili nilai, dan sumbu y mewakili ulasan.



Gambar 2.2 Representasi *certain data* [5]

Berbeda dengan certain data yang dapat diwakili dengan titik, uncertain data diwakili oleh titik-titik. Titik-titik ini biasa disebut dengan *instance.* Sebagai contoh apabila seorang pengemudi bekerja selama satu minggu, maka seorang pengemudi memiliki 7 *instance* dimana satu *instance* mewakili satu hari. Atau dalam kasus lain seorang pengemudi mendapatkan penilaian dan ulasan dari 10 orang berbeda, maka seorang pengemudi mempunyai 10 instance dimana satu *instance* mewakili penilaian dan ulasan dari seorang penumpang. Apabila digambarkan dalam grafik maka dapat digambarkan seperti pada Gambar 2.3.



Gambar 2.3 Representasi *uncertain data* [5]

2.3. *Python*

Python adalah sebuah *interpreter,* berbasis objek, dan bahasa pemrograman tingkat tinggi. Sebagai bahasa pemrograman tingkat tinggi, python juga didukung oleh struktur data tingkat tinggi pula. Python adalah bahasa pemrograman yang mudah dipelajari. Salah satu aspek yang ditekankan oleh python adalah kemudahan sumber kode untuk dibaca. Python dapat digunakan untuk membuat *script* maupun menghubungkan antar komponen. Python mendukung modul dan paket yang dapat membuat sekumpulan kode untuk digunakan kembali. Python memiliki banyak *library* dan didistribusikan secara gratis. Sampai saat ini python sudah tersedia dalam 2 versi yaitu versi 2.x dan versi 3.x.

Python menawarkan produktivitas pada penggunanya. Karena python adalah *interpreter*, python tidak memakan biaya untuk kompilasi sehingga proses penngubahan, pengujian, dan *debug* menjadi lebih cepat. Melakukan debug pada python sangat mudah karena tidak akan mengakibatkan *segmentation fault* akan tetapi akan menimbulkan exception apabila terdapat kesalahan dalam penulisan kode. Ketika program tidak menangkap exception, maka python akan menampilkan *stack trace* yang dapat kita gunakan untuk menganalisa kesalahan yang terjadi.

2.4. PyCharm

PyCharm adalah salah satu IDE yang dikembangkan oleh Jetbrains. Saat ini PyCharm sudah tersedia untuk profesional dan komunitas. Versi profesional memiliki fitur yang lebih banyak daripada versi komunitas akan tetapi kita diharuskan membayar untuk dapat menggunakannya. PyCharm menyediakan fitur intelligent code completion yang akan membantu kita untuk menulis kode lebih cepat.

PyCharm menyediakan fitur *on-the-flye error checking* dan *quick-fixes* dimana PyCharm dapat mendeteksi kesalahan penulisan kode dan memberikan rekomendasi perbaikan yang akan dilakukan oleh PyCharm sesuai opsi perbaikan yang kita pilih. PyCharm juga menyediakan fitur untuk melakukan tes dan *debugging.* PyCharm juga menyediakan fitru refactoring yang akan memudahkan pengguna saat pengguna ingin mengubah sesuatu tanpa merusak integritas susunan kode. [5]

2.5. Struktur Data

Dalam istilah ilmu komputer, struktur data adalah cara penyimpanan , pengorganisasian , dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien.

Dalam teknik pemrograman, struktur data berarti tata letak data yang berisi kolom-kolom data,baik itu kolom yang tampak oleh pengguna (*user*) ataupun kolom yang hanya digunakan untuk keperluan pemrograman yang tidak tampak oleh pengguna. Setiap baris dari kumpulan kolom-kolom tersebut dinamakan catatan (*record*). Lebar kolom untuk data dapat berubah dan bervariasi. Ada kolom yang lebarnya berubah secara dinamis sesuai masukan dari pengguna dan juga ada kolom yang lebarnya tetap. [6]

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Bab ini menjelaskan tentang analisis dan perancangan mengenai sistem yang akan dibuat.

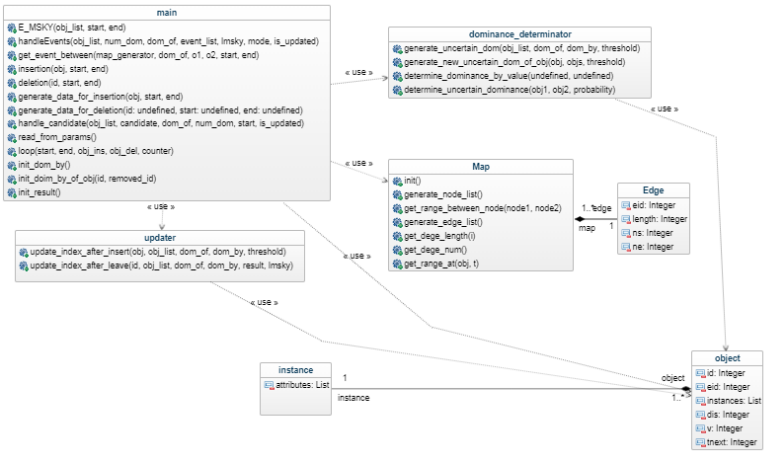
3.1. Daftar Simbol

Pada bab ini akan digunakan beberapa simbol seperti pada tabel:

L_{msky}	Hasil penjadwalan dari algoritma *E_MSKY*
O_{cand}	Objek-objek yang tidak termasuk *skyline* namun berpotensi menjadi skyline
Dom_{ns}	Struktur data dictionary yang menyimpan daftar objek yang direlasikan dengan objek-objek yang mendominasi objek tersebut secara non-spasial.
$Dom_{ns}(o)$	Objek-objek yang mendominasi objek o secara non-spasial
Dom_By(o)	Objek-objek yang didominasi objek o secara non-spasial

3.2. Arsitektur Sistem

Modul utama sistem adalah main. Pada saat program berjalan, modul main akan menggunakan kelas Map untuk membaca jaringan jalan dari file. Selanjutnya kelas Map akan menggunakan kelas edge untuk memetakan jaringan jalan. Apabila proses pemetaan selesai selanjutnya program membaca objek masukan dan dipetakan menggunakan kelas objek. Untuk membandingkan dominasi modul main akan menggunakan modul dominance_determinator. Apabila semua proses telah selesai, modul main akan menggunakan modul updater untuk mengubah memperbarui indeks sistem.



Gambar 3.1 Arsitektur Sistem

3.3. *Skyline Query*

Skyline dapat diartikan suatu kondisi dimana suatu objek tidak didominasi oleh objek-objek lain. Pada kasus pencarian *skyline* pada objek-objek bergerak di jaringan jalan raya, dominasi dapat dilihat dari dua sudut pandang, yaitu secara spasial maupun non-spasial. Suatu objek dikatakan mendominasi secara spasial apabila objek tersebut memiliki jarak yang lebih dekat terhadap query point dibandingkan objek lain yang mendominasi objek tersebut secara non-spasial. *Query point* adalah titik yang menjadi acuan pencarian *skyline*. Sedangkan suatu objek dikatakan mendominasi secara non-spasial dilihat dari nilai objek tersebut yang tidak berkaitan dengan lokasi ataupun jarak. Oleh karena itu, walaupun suatu objek didominasi secara non-spasial, objek tersebut tetap menjadi *skyline* apabila mendominasi secara spasial.

Karena objek-objek pada kasus ini berupa objek yang berjalan, maka objek-objek tersebut akan memiliki jarak yang selalu berubah-ubah terhadap *query point*. Karena jarak yang berubah-ubah tersebut, tidak menutup kemungkinan bahwa objek yang sebelumnya tidak menjadi *skyline* akan selamanya tidak

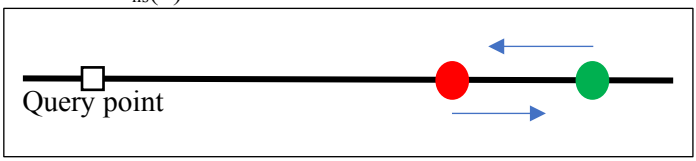
menjadi *skyline* begitupula sebaliknya. Untuk menangani kondisi tersebut, digunakan pemrosesan berbasis *event*. *Event* adalah suatu kondisi dimana dua objek yang bergerak mengalami perubahan dominasi secara jarak. Sebagai contoh apabila objek 1 lebih dekat terhadap query point dibandingkan objek 2 pada detik 5 sedangkan pada detik 11 objek 1 lebih jauh terhadap query point dibandingkan dengan objek 2, maka pada detik 11 akan dicatat sebagai *event* antara objek 1 dan objek 2.

3.4. *Skyline Query* pada Jaringan Jalan Raya

Skyline query pada objek bergerak di jalan raya tidak sama dengan pencarian *skyline* pada objek yang tidak bergerak. Pencarian *skyline* pada objek bergerak di jalan raya akan melibatkan jarak. Jarak akan menjadi salah satu atribut dari objek yang bergerak dimana semakin dekat jarak berarti objek tersebut memiliki satu atribut yang semakin baik pula. Dengan demikian objek yang bergerak tersebut akan memiliki 2 atribut yaitu atribut non-spasial dan atribut spasial yaitu jarak. Suatu objek dikatakan mendominasi secara spasial apabila objek tersebut memiliki jarak lebih dekat dibandingkan dengan objek lain. sedangkan suatu objek dikatakan mendominasi objek lain secara non-spasial apabila objek tersebut mendominasi pada atribut statis seperti yang telah dijelaskan pada subbab 2.1. Dengan demikian suatu objek dikatakan mendominasi objek lain apabila telah memenuhi dua syarat yaitu mendominasi secara spasial dan mendominasi secara non-spasial. Dan objek bergerak di jalan raya dapat menjadi *skyline* apabila tidak didomimasi secara non-spasial ataupun didomiansi secara non-spasial namun mendominasi secara spasial. Adanya jarak sebagai atribut spasial akan mengakibatkan dua hal, yaitu:

1. Suatu objek o yang tidak termasuk *skyline* dapat menjadi *skyline* apabila pada mulanya didominasi secara non-spasial dan secara *spasial* tapi beberapa waktu kemudian

objek tersebut tidak didominasi secara spasial oleh semua $o' \in Dom_{ns}(o)$.



Gambar 3.2 Ilustrasi pengaruh jarak terhadap *skyline*

Pada gambar di atas, objek yang berwarna hijau didominasi oleh objek yang berwarna merah secara non-spasial maupun secara spasial. Tanda panah di atas dan bawah objek menandakan arah pergerakan objek. Objek berwarna merah bergerak menjauhi *query point*. Sedangkan objek berwarna hijau bergerak mendekati *query point*. Kondisi awal yang termasuk *skyline* hanyalah objek yang berwarna merah sedangkan objek yang berwarna hijau tidak termasuk *skyline* karena didominasi oleh objek berwarna merah secara spasial maupun non-spasial. Beberapa waktu kemudian jarak objek berwarna hijau lebih dekat dibandingkan objek berwarna merah, maka objek hijau termasuk *skyline* karena objek berwarna merah tidak lagi mendominasi secara spasial.

2. Suatu objek o yang termasuk *skyline* bisa keluar dari *skyline* apabila pada mulanya mendominasi secara spasial akan tetapi didominasi secara non-spasial dan beberapa waktu kemudian objek tersebut didominasi secara spasial oleh paling tidak satu objek yang termasuk $Dom_{ns}(o)$.

   Sebagai gambaran, pada Gambar 3.1, objek berwarna merah didominasi secara non-spasial oleh objek berwarna hijau. Pada mulanya objek berwarna merah dan objek berwarna hijau termasuk *skyline* walaupun objek berwarna merah didominasi secara non-spasial oleh objek berwarna hijau akan tetapi objek berwarna merah tidak didominasi secara spasial oleh objek berwarna hijau. Beberapa saat kemudian objek berwarna merah didominasi oleh objek berwarna hijau secara spasial atau dengan kata lain objek

berwarna merah memiliki jarak lebih jauh terhadap *query point* dibandingkan dengan objek berwarna hijau. Dengan kondisi yang seperti itu, objek berwarna merah sudah tidak termasuk *skyline* karena didominasi oleh objek berwarna hijau baik secara non-spasial dan secara spasial.

Perhitungan *skyline* pada *dynamic data* oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya memerlukan optimasi agar proses pengolahan *query* cepat. Proses perhitungan *skyline query* sendiri harus dihitung antara satu objek dengan semua objek yang lain. Lebih spesifik lagi hubungan dominasi antara dua objek tidak bisa berlaku selamanya karena ada faktor jarak yang berubah-ubah. Sebagai contoh suatu objek saat ini tidak menjadi *skyline*, namun beberapa waktu kemudian objek tersebut bisa menjadi *skyline* karena memiliki jarak yang lebih dekat daripada objek-objek yang mendominasi objek tersebut. Hal itu membuat komputasi menjadi sangat berat apabila harus mengiterasi setiap waktu. Untuk mengatasi permasalahan tersebut akan digunakan algoritma *E_MSKY* [8]. Algoritma *E_MSKY* adalah algoritma untuk melakukan *skyline query* di jalan raya pada objek bergerak. Algoritma *E_MSKY* menawarkan pemrosesan berbasis *event*. *Event* adalah suatu kondisi dimana antara objek yang memiliki hubungan dominasi non-spasial terjadi perubahan dominasi spasial.

Algoritma *E_MSKY* melakukan inisiasi terlebih dahulu sebelum penghitung waktu pada program berjalan. Algoritma tersebut akan menghasilkan semacam penjadwalan kapan suatu objek menjadi *skyline* ataupun sudah tidak lagi menjadi *skyline* yang dinotasikan dengan L_{msky}. Penjadwalan tersebut sudah mencakup mulai dari awal sampai akhir program berjalan sehingga pada saat program berjalan, program hanya cukup melihat penjadwalan yang sudah dihasilkan sebagai acuan pada waktu tertentu apakah ada objek baru yang menjadi *skyline* ataupun ada objek yang sudah tidak menjadi *skyline*. Dengan begitu komputasi dapat dikurangi karena program tidak perlu melakukan komputasi ulang pada saat program berjalan. Optimasi dapat dilakukan dengan algoritma di bawah:

```
     E_MSKY
     Initialization
1    for each  o ∈ O:
2       compute Dom_ns[o]:
3       if Dom_ns [o] = ∅
4          insert the entry (o,∞,∞) to lmsky
5       else:
6          insert o into O_cand
7    end for
     // create the events queue Q_s
8    for each oi ∈ O_cand:
9       for each oj ∈ Dom_ns[oi]:
10         If ∀j,s.t.dt(q,oi)<dt(q,oj) then
11            Insert the entry (oi,t0,∞) into lmsky
12         compute the intersection time tx of each event
13         If tx < oi.tnext and tx<oj.tnext:
14            enqueue <oi,oj,tx> into Q_e
15      end for
16   end for
     //continous query
17   while t<to+T
18      pick time interval [tsk, tek] based on tnext in Q_t
19      handleEvents(q,[t_sk, t_ek])
20      Add new tnext to Q_t and new events to Q_e
21   end while
22   return lmsky
```

Kode Sumber 3.1 *E_MSKY* [8]

Pada tahap inisiasi program akan melakukan pemrosesan untuk mencari *skyline* dari semua objek secara menyeluruh. Pada tahap inisiasi akan diperoleh keluaran berupa daftar objek dengan struktur data *dictionary* dan dinotasikan dengan L_{msky}. Setiap objek pada L_{msky} akan memiliki waktu ketika objek tersebut menjadi *skyline* dan waktu ketika objek tersebut sudah tidak menjadi *skyline* yang dinotasikan dengan (o,et,lt). Et menandakan waktu saat objek menjadi *skyline* dan lt menunjukkan waktu saat objek sudah tidak lagi menjadi *skyline*. Selanjutnya kita dapat menggunakan L_{msky} sebagai acuan saat berada pada tahap berjalan.

Tahap pertama pada proses inisiasi adalah menghimpun $Dom_{ns}(o)$ untuk setiap o yaitu objek bergerak. Untuk setiap objek o yang ada di Dom_{ns} apabila o tidak didominasi objek-objek lain secara non-spasial atau $Dom_{ns}(o)=\emptyset$, berarti objek tersebut adalah *skyline* dan kita tambahkan nila (id objek, -,-) ke L_{msky} yang menandakan bahwa objek tersebut adalah *skyline* dan akan selamanya menjadi *skyline* sampai ada objek lain yang masuk pada saat program berjalan(Kode Sumber 3.1 baris 4). Oleh karena itu waktu saat objek tersebut menjadi *skyline* maupun waktu saat objek tersebut sudah tidak menjadi *skyline* tidak didefinisikan. Setelah itu objek-objek yang bukan *skyline* akan dikumpulkan sebagai kandidat *skyline* (Kode Sumber 3.1 baris 6) yang dinotasikan dengan O_{cand}. Pada dasarnya cara algoritma *E_MSKY* menentukan apakah suatu objek merupakan *skyline* atau tidak yaitu dengan membandingkan objek o pada O_{cand} dengan semua o' di $Dom_{ns}(o)$. Apabila tidak ada yang lebih dekat berarti objek tersebut adalah *skyline*.

Selanjutnya dari setiap objek o yang termasuk dalam O_{cand} akan diproses dengan $Dom_{ns}(o)$. Dari hasil pemrosesan tersebut akan diketahui apakah objek tersebut mendominasi secara spasial atau tidak dan daftar *event*. Setiap *event* akan dinotasikan dengan <oi,oj,t> dimana oi mewakili objek pertama, oj mewakili objek kedua, dan t mewakili waktu *event* terjadi. Apabila objek tersebut mendominasi secara spasial, maka kita tambahkan nilai (id objek, waktu mulai, -) (Kode Sumber 3.1 baris 8-16) . Setelah mendapatkan *event* yang terjadi, selanjutnya kita proses kumpulan *event* tersebut dengan algoritma handleEvents pada Kode Sumber 3.2

```
handleEvents
1    e=Qe.erase()
2    while(e.tx<te):
3       If oi ∈ Domns[oj]:
4          oj.Num_Domd--
5          If oj.Num_Domd = 0 then
6             insert the entry (oj,e.tx,∞)
7       else if oj ∈ Domns(oi):
```

```
8           oi.Num_Dom_d++
9           If oi.Num_Dom_d <=1 then
10             update the entry of oi in lmsky to (oi,t_si, e.t_x)
11      e=Q.erase()
12  end while
```

Pada Kode Sumber 3.2 Num_Dom_d digunakan untuk menampung jumlah objek yang mendominasi suatu objek secara spasial. Setiap *event* dinotasikan dengan <*oi,oj,t*> dimana *oi* memiliki kondisi awal lebih dekat terhadap *query point* dibandingkan dengan *oj*. Pada saat *t*, *oi* akan memiliki jarak lebih jauh daripada *oj*. Apabila *oj* didominasi *oi*, maka objek yang mendominasi *oj* secara spasial berkurang. Apabila jumlah objek

Kode Sumber 3.2 *handleEvents [8]*

yang mendominasi *oj* secara spasial adalah 0, maka *oj* akan menjadi *skyline*(Kode Sumber 3.2 baris 3-6). Sedangkan apabila *oi* didominasi *oj*, maka itu berarti objek *oi* sudah tidak memiliki kesempatan untuk menjadi *skyline* karena didominasi oleh objek yang telah mendominasi secara non-spasial(Kode Sumber 3.2 baris 7-10)

3.5. *Skyline Query* pada *Uncertain Data*

Perlakuan *skyline query* pada *certain data* dan *uncertain data* berbeda. Hal itu terjadi karena pada *uncertain data*, sebuah objek terjadi dari banyak *instance*. Untuk melakukan pengolahan *skyline query* pada tugas akhir ini digunakan *probabilistic skyline query* [9]. Apabila $U=\{u_1,...,u_n\}$ dan $V=\{u_1,...,u_n\}$ adalah dua objek uncertain data dan instance setiap objek, maka probabilitas V mendominasi U seperti pada Gambar 3.2.

$$\begin{aligned} Pr[V \prec U] &= \int_{u \in D} f(u)\left(\int_{v \prec u} f'(v)\, dv\right) du \\ &= \int_{u \in D} \int_{v \prec u} f(u) f'(v)\, dv\, du \end{aligned}$$

Gambar 3.3 Rumus probabilitas dominasi *uncertain data* [5]

3.6. *Skyline Query* pada *Dynamic Data*

Dynamic data berarti data yang terus berubah-ubah. Lebih khusus pada permasalahan yang dibahas dalam tugas akhir ini adalah jumlah objek yang berubah-ubah. Pada saat program berjalan akan ada objek baru yang masuk/muncul atau objek yang keluar/hilang. Kondisi seperti ini akan membuat L_{msky} yang telah dihitung sebelum program berjalan menjadi berubah.

Akan ada dua kondisi pada saat program berjalan yaitu objek baru masuk atau objek keluar. Pada saat objek baru masuk atau keluar, akan dilakukan perhitungan ulang terhadap L_{msky} namun dengan jumlah objek yang tidak sebanyak saat inisiasi. Apabila terdapat objek awal sebanyak 1000 dan setelah tahap inisiasi terdapat 10 objek di $L_{msky,}$ maka perhitungan hanya akan dilakukan terhadap objek yang baru masuk ataupun keluar dengan objek-objek yang berada di L_{msky} yaitu sebanyak 10. Pada saat objek baru masuk ataupun keluar, masih digunakan *E_MSKY* namun dengan jumlah objek yang lebih sedikit.

3.6.1. Objek Baru Masuk

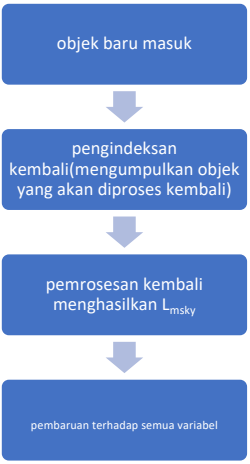
Pada saat ada objek baru masuk, terdapat dua kategori objek yang berada di jalan raya yaitu objek yang berpotensi menjadi *skyline* atau yang termasuk ke dalam L_{msky} dan objek yang tidak berpotensi menjadi *skyline* yaitu objek yang tidak termasuk ke dalam L_{msky}. Membandingkan objek yang baru masuk dengan objek yang tidak berpotensi menjadi *skyline* tidak akan menghasilkan kesimpulan apapun. Oleh karena itu untuk mengetahui apakah objek yang baru masuk bisa menjadi *skyline* maka objek yang baru masuk cukup dibandingkan dengan objek yang berpotensi menjadi *skyline*. Itu adalah prinsip utama untuk mengurangi biaya komputasi saat ada objek baru yang masuk.

Pada saat ada objek baru masuk, algoritma *E_MSKY* akan digunakan untuk membangun L_{msky} yang baru. Yang berbeda adalah jumlah objek yang akan diproses ulang lebih sedikit.

Ada beberapa variabel yang berbeda pada saat objek masuk dibandingkan dengan algoritma *E_MSKY* yaitu:

- O_{cand} atau kandidat *skyline*
  Pada saat objek baru masuk maka objek baru tersebut akan dibandingkan dengan semua objek yang berada di L_{msky}. Apabila objek baru tersebut adalah *skyline* maka jumlah O_{cand} yang akan diproses sebanyak objek yang didominasi objek baru tersebut. Apabila objek yang baru masuk bukan *skyline* maka jumlah O_{cand} yang akan diproses sejumlah objek yang didomiansi objek baru tersebut ditambah dengan objek baru tersebut.
- $Dom_{ns}(o)$ untuk setiap $o \epsilon O_{cand}$
  Untuk setiap $o \epsilon O_{cand}$, $Dom_{ns}(o)$ adalah objek-objek yang mendominasi o secara non-spasial dan termasuk ke dalam L_{msky}. Hal ini menerapkan prinsip bahwa cukup membandingkan dengan objek yang termasuk dalam *skyline* atau L_{msky} untuk mengurangi biaya komputasi.

Dengan adanya dua variabel yang berbeda akan mengurangi biaya komputasi. Pembentukan dua variabel tersebut akan disebut dengan pengindeksan kembali. Adapun alur kerja program pada saat ada objek baru masuk terlihat di Gambar 3.3.



Gambar 3.4 Alur objek masuk

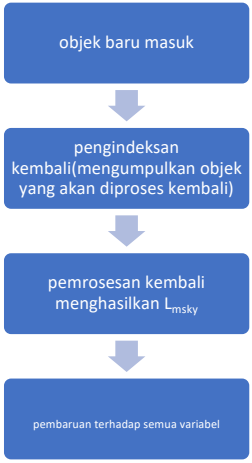
Pada saat ada objek baru masuk akan dilakukan pengindeksan kembali guna mendapatkan variabel-variabel baru untuk mengurangi biaya komputasi. Setelah itu variabel-variabel tersebut akan diproses kembali dengan algoritma *E_MSKY* untuk mendapatkan L_{msky} yang baru. Tahap akhir adalah memperbarui variabel global.

3.6.2. Objek Keluar

Ketika suatu objek keluar dapat digunakan algoritma CNO. Algoritma CNO mempertimbangkan objek tersebut termasuk dalam L_{msky} atau tidak termasuk dalam L_{msky}. Ketika objek yang keluar tidak termasuk dalam L_{msky} maka hal itu tidak berpengaruh terhadap L_{msky}. Yang diperlukan hanya memperbarui semua variabel global. Tapi apabila objek yang keluar termasuk dalam L_{msky}, hal itu dapat berpengaruh terhadap L_{msky}. Pada saat objek o keluar, maka perhitungan ulang L_{msky} hanya perlu dilakukan terhadap semua $o' \in Dom\_By(o)$. Ada 2 kemungkinan yang akan terjadi apabila suatu objek keluar yaitu:

- Mengubah waktu objek lain dapat menjadi *skyline* ataupun tidak menjadi *skyline*. Sebagai contoh apabila *o4* didominasi oleh *o2* dan *o4* bisa menjadi *skyline* detik 8 karena sampai detik 8 *o4* masih didominasi secara spasial oleh *o2*. Apabila *o2* keluar sebelum detik 8, mungkin waktu o4 untuk menjadi *skyline* bisa menjadi lebih awal.
- Membuat objek yang awalnya bukan *skyline* menjadi *skyline*. Kondisi ini dapat dicapai apabila objek yang didominasi objek yang keluar, tidak didominasi oleh objek lain di *skyline* baik secara spasial ataupun non-spasial.

Adapun alur kerja pada saat ada objek yang keluar ditenkukan pada Gambar 3.5. Pada saat ada objek yang kaluar, maka program akan membentuk variabel sementara dari O_{cand} dan Dom_{ns} untuk mengurangi biaya komputasi. Selanjutnya *skyline* akan dicari menggunakan variabel-variabel tersebut agar biaya komputasi tidak berat seperti algoritma *E_MSKY* tanpa modifikasi.



Gambar 3.5 Alur kerja saat ada objek keluar

3.7. Struktur Data

Bagian ini menjelaskan rancangan struktur data yang akan digunakan pada sistem. Pada jaringan jalan raya terdapat dua komponen yaitu node dan edge. *Node* merupakan persimpangan jalan dan *edge* adalah jalan yang dibatasi dua persimpangan jalan atau *node*. Adapun struktur data yang akan digunakan pada sistem adalah sebagai berikut:

1. *Edge*

   Edge adalah garis ataupun jalan yang dibatasi oleh dua persimpangan jalan atau *node*.

Atribut	**Keterangan**
eid	Nomor *edge*
length	Panjang *edge*
ns	*Node* awal
ne	*Node* akhir

Tabel 3.1 Struktur data *edge*

2. *Instance*

   Instance dalam hal ini adalah sekumpulan nilai yang dimiliki oleh suatu objek.

Atribut	**Keterangan**
attributes	Kumpulan dari nilai-nilai yang dimiliki sebuah instance

Tabel 3.2 Struktur data *instance*

3. Objek

   Objek dalam permasalahan yang dibahas pada tugas akhir ini adalah objek yang bergerak pada jaringan jalan raya. Objek juga dapat diturunkan menjadi *query point*, yaitu titik atau objek yang diam yang menjadi acuan pencarian *skyline.*

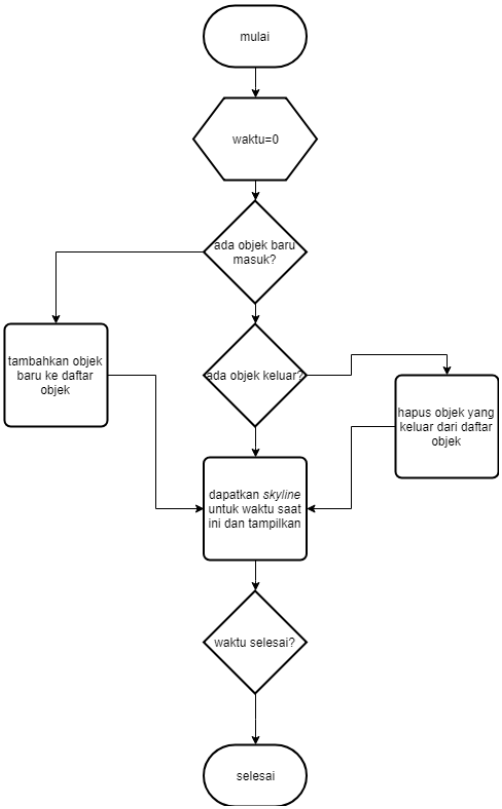
Attribut	**Keterangan**
id	Nomor objek
eid	Nomor *edge* yang dilalui objek
instances	Sekumpulan *Instance* yang dimiliki suatu objek

dis	Jarak objek terhadap node awal dari *edge* yang dilalui objek tersebut
v	Kecepatan objek
tnext	Waktu dimana objek sampai pada node

Tabel 3.3 Struktur data *object*

3.8. *A*lgoritma *Naive*

Pada penelitian ini juga dihasilkan algoritma *naive*. Prinsip kerja algoritma naive adalah melakukan pemrosesan setiap satuan waktu. Adapun alur pemrosesan algoritma naive seperti pada Gambar 3.6.



Gambar 3.6 Alur kerja algoritma *naive*

Algoritma *naive* akan melakukan iterasi terhadap waktu. Apabila saat melakukan iterasi ada objek baru masuk, maka objek baru tersebut ditambahkan ke daftar objek dan dilakukan perhitungan *skyline* terhadap daftar objek. Pada saat iterasi apabila ada objek yang keluar maka objek tersebut akan dihapus dari daftarr ojek dan dilakukan perhitungan *skyline* terhadap daftar objek. Apabila tidak terdapat objek yang masuk ataupun keluar, algoritma *naive* akan melakukan perhitungan *skyline* terhadap objek yang ada.

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Bab ini akan menjelaskan tentang implementasi Tugas Akhir berdasarkan rancangan perangkat lunak. Proses implementasi mengacu pada rancangan perangkat yang telah dilakukan sebelumnya, namun juga dimungkinkan terjadinya perubahan-perubahan jika dirasa perlu.

4.1. Lingkungan Implementasi

Implementasi dilakukan pada sebuah komputer dengan sistem operasi windows. Komputer yang digunakan memiliki memori sebesar 8 GB. Bahasa pemrograman yang digunakan untuk implementasi adalah python dan untuk menuliskan kode digunakan editor PyCharm.

4.2. Implementasi Algoritma CNO

Subbab ini akan membahas mengenai implementasi algoritma berbasis *E_MSKY* [8] yaitu algoritma CNO.

4.2.1. Implementasi Objek Baru Masuk

Untuk mengurangi biaya komputasi, objek yang baru masuk cukup dibandingkan secara non-spasial dengan objek-objek yang berada di $L_{msky.}$ Dari perbandingan tersebut akan didapatkan objek-objek di L_{msky} yang mendominasi maupun didominasi oleh-objek yang baru masuk(Kode Sumber 4.1 baris 1). Saat objek baru masuk, maka akan ada 2 kondisi, yaitu:

1. Objek yang baru masuk tidak didominasi secara non-spasial. Apabila objek yang baru masuk tidak didominasi secara non-spasial, maka komputasi akan dilakukan terhadap objek yang didominasi secara non-spasial oleh objek yang baru masuk(Kode Sumber 4.1 baris 3-9). Objek-objek tersebut adalah objek-objek yang berpotensi untuk

berubah waktu masuk dan keluarnya di L_{msky} dan dapat kita notasikan dengan O_{cand}.

2. Apabila objek yang baru masuk didominasi secara non-spasial maka perhitungan akan dilakukan terhadap objek-objek yang didominasi secara non-spasial oleh objek yang baru masuk begitu juga objek yang baru masuk tersebut(Kode Sumber 4.1 baris 11-21). Objek-objek tersebut adalah objek-objek yang berpotensi untuk berubah waktu masuk dan keluarnya di L_{msky} dan dapat kita notasikan dengan O_{cand}.

Setelah kedua kondisi tersebut terpenuhi, maka dilakukan pengindeksan ulang terhadap objek-objek yang terhimpun di O_{cand}(Kode Sumber 4.1 baris 24-29). Tidak semua objek yang didominasi secara non-spasial oleh objek yang baru masuk akan diproses. Sebagai contoh apabila objek baru masuk pada detik ke 20 sedangkan objek yang didominasi secara non-spasial sudah tidak valid. Objek itu akan ditambahkan di *dom_of_new*(Kode Sumber 4.1 32-34) yang berikutnya akan diproses oleh Kode Sumber 3.4. Dan yang terakhir adalah inisiasi *num_dom*(Kode Sumber 4.1 baris 37-39).

generate_data_for_insertion(L_{msky}, dom_by,dom_of,obj_list,obj,start,end)

```
1   dominating,dominated=dominance_determinator.gen
    erate_new_uncertain_dom_of_obj(obj,msky_obj,thr
    eshold)
2   if obj not dominated:
3       append [[],[]] to Lmsky
4       for each i in dominated:
5           if obj_list[i].tnext>start:
6               append i to candidate
7               obj_list_new[i]=obj_list[i]
8           end if
9       end for
10  else:
11      for each i in dominating:
12          if obj_list[i].tnext > start:
13              obj_list_new[i]=obj_list[i]
14          end if
15      end for
```

```
16      for each i in dominated:
17          if obj_list[i].tnext>start:
18              append i to candidate
19              obj_list_new[i]=obj_list[i]
20      append obj.id to candidate
21      dom_of[obj.id]=dominating
22  end if
23  for i in candidate:
24      if obj.id in dom_of_new[i]:
25          append obj.id to dom_of_new[i]
26      end if
27      for j in dom_of[i]:
28          append j to dom_of_new[i]
29      end for
30  end for
31  for i in dominated:
32      if i not in candidate:
33          dom_of_new[i]=[obj.id]
34      end if
35  end for
36  obj_list_new[obj.id]=obj
37  for each i in dom_of:
38      num_dom[i]=0
39  end for
40  return obj_list_new, candidate, dom_of_new,
    num_dom, new_lmsky
```

Kode Sumber 4.1 ***generate_data_for_insertion***

Setelah mendapatkan objek-objek yang akan diproses pada Kode Sumber 4.1, selanjutnya akan dilakukan pemrosesan ulang dengan Kode Sumber 4.5 dan akan didapatkan L_{msky} terbaru setelah objek baru masuk(Kode Sumber 4.2 baris 5). Selanjutnya lakukan pembaruan terhadap L_{msky} yaitu sebelum objek baru masuk dengan L_{msky} terbaru. Pembaruan dalam hal ini dilakukan terhadap objek di L_{msky}. Setelah dilakukan pemrosesan ulang oleh Kode Sumber 4.5, selanjutnya dilakukan pembaruan terhadap indeks L_{msky}. Hal ini dilakukan karena tidak semua objek diproses kembali saat objek baru masuk(Kode Sumber 4.2 baris 7-13). Dan proses yang terakhir adalah pembaruan indeks(Kode Sumber 4.2 baris 16).

```
insertion(obj,start,end)
1    obj_list_new, candidate, dom_of_new,
     num_dom_new,new_lmsky=generate_data_for_inserti
     on(obj,start,end)
2    for each i in is_updated:
3        is_updated[i]=false
4    end for
5    lmsky=handle_candidate(obj_list_new,candidate,d
     om_of_new,num_dom_new,start,is_updated)
6    for each i in lmsky:
7        if i!=obj.id:
8            if obj,id in dom_of_new[i]:
9                update lmsky[i] to [[start-
     s],[start]]
10           else:
11               append start to lmsky[[i][0]
12           end if
13       end if
14   end for
15   init_result()
16   updater.update_index_after_insert(obj,obj_list,
     dom_of,dom_by,lmsky,logging,threshold)
```

Kode Sumber 4.2 *insertion*

4.2.2. Implementasi Objek Keluar

Apabila suatu objek o keluar maka pemrosesan ulang cukup dilakukan pada semua $o' \in Dom\_By(o)$. Keluarnya suatu objek dapat mempengaruhi waktu objek lain menjadi *skyline* atau waktu objek lain tidak menjadi *skyline* dan juga dapat membuat objek yang awalnya tidak termasuk *skyline* dapat menjadi *skyline*. Dua kondisi di atas hanya terjadi terhadap objek yang didominasi secara non-spasial oleh objek yang keluar maka pemrosesan ulang akan dilakukan terhadap objek yang didominasi secara non-spasial oleh objek yang keluar. Pada saat objek keluar, akan ada dua kondisi berkaitan dengan keberadaan objek yang keluar terhadap L_{msky}. Apabila objek yang keluar tidak termasuk *skyline*, kita tidak perlu melakukan pengindeksan kembali(Kode Sumber 4.3 baris 34).

Sebaliknya apabila objek yang keluar termasuk dalam L_{msky}, maka perlu dilakukan pengindeksan ulang(Kode Sumber 3.5 baris 2-29).

```
generate_data_for_deletion(Lmsky,obj_list,dom_of, ide,start,end)
1     if id in Lmsky:
2         remove id from Lmsky
3         if len(dom_by[id])>0:
4             for each i in dom_by[id]:
5                 remove id from dom_of[i]
6                 for each j in dom_of[i]:
7                     if j in lmsky:
8                         append j to dom_of_new[i]
9                     end if
10                end for
11            end for
12            for each i in dom_of_new:
13                if len(dom_of_new[i])==0:
14                    new_lmsky=[[],[]]
15                else:
16                    append i to candidate
17                end if
18            end for
19            for each i in candidate:
20                for each j in dom_of_new[i]:
21                    if j not in obj_list_new:
22                        obj_list_new[j]=obj_list[j]
23                    end if
24                end for
25            end for
26            for each i in dom_of_new:
27                num_dom[i]=0
28            end for
29            return
      obj_list_new,candidate,dom_of_new,num_dom,new_l
      msky
30        else:
31            return None, None, None, None, None
32        end if
33    Else
34        return None, None, None, None, None
35    end if
```

Kode Sumber 4.3 ***generate_data_for_deletion***

Setelah mendapatkan objek-objek yang akan diproses pada Kode Sumber 4.3, selanjutnya akan dilakukan pemrosesan ulang dengan Kode Sumber 4.5 dan akan didapatkan L_{msky} terbaru setelah objek keluar(Kode Sumber 4.4 baris 6). Selanjutnya dilakukan pembaruan terhadap L_{msky} yaitu sebelum objek baru masuk dengan L_{msky} terbaru. Pembaruan dalam hal ini dilakukan terhadap objek di L_{msky}. Setelah dilakukan pemrosesan ulang oleh algoritma 4.5, selanjutnya dilakukan pembaruan terhadap indeks $L_{msky.}$ Hal ini dilakukan karena tidak semua objek diproses kembali saat objek baru masuk(Kode Sumber 4.2 baris 8-13). Dan proses yang terakhir adalah pembaruan indeks(Kode Sumber 4.2 baris 17).

```
deletion(id,start,end)
1    obj_list_new,candidate,dom_of_new,num_dom,new_l
     msky = generate_data_for_deletion(id, start,
     end)
2    if candidate!=None:
3        for each i in is_updated:
4            is_updated[i]=false
5        end for
6        lmsky=
               handle_candidate
                (obj_list_new,candidate,dom_of_new,
                 num_dom_new,start,is_updated)
7        for each i in lmsky:
8            if i!=obj.id:
9                if obj,id in dom_of_new[i]:
10                   update lmsky[i] to [[start-
     s],[start]]
11               else:
12                   append start to lmsky[[i][0]
13               end if
14           end if
15       end for
16       init_result()
17   updater.update_index_after_insert(obj,obj_list,
     dom_of,dom_by,lmsky,logging,threshold)
```

Kode Sumber 4.4 *deletion*

4.2.3. Implementasi Fungsi *handle_candidate*

Keluaran dari Kode Sumber 4.1 dan Kode Sumber 4.3 adalah objek-objek yang akan dihitung/diproses ulang untuk menghasilkan ataupun memperbarui L_{msky} setelah ada objek yang masuk ataupun objek yang keluar. Selanjutnya setelah objek-objek tersebut diperoleh akan dilakukan pemrosesan terhadap data tersebut dengan Kode Sumber 4.5 untuk menghitung L_{msky} yang baru. Untuk setiap objek di O_{cand}, bandingkan jarak dengan semua objek yang mendominasi o secara non-spasial(Kode Sumber 4.5 baris 4) setelah itu cari *event* yang terjadi(Kode Sumber 4.5 baris 7). Apabila o memiliki jarak paling dekat dibandingkan dengan semua objek yang mendominasi o secara non-spasial itu berarti o menjadi skyline karena tidak ada objek lain yang mendominasi jarak o(Kode Sumber 4.6 baris 9). Pada tahap akhir adalah perhitungan L_{msky} baru(Kode Sumber 4.6 baris 10).

handle_candidate(obj_list_temp, O_{cand}, dom_of_temp, num_dom_temp, start)

```
1   for each i ∈ Ocand
2       dominating_by_distance = True
3       for each j ∈ dom_of_temp[i]
4           if dt(i) > dt(j)
5               dominating_by_distance = False
6               i.Num_Domd++
7           compute event between i and j then append
    to evnet_list
8       if dominating_by_distance = True
9           Lmsky[i] = [start,-]
            Is_updated[i]=true
10  lmsky = handle_event(obj_list_temp, num_dom,
    dom_of_temp, event_list, lmsky,is_updated)
11  return Lmsky
```

Kode Sumber 4.5 *handle_candidate*

4.3. Implementasi Algoritma *Naive*

Subbab ini akan membahas mengenai implementasi algoritma *naive.*

4.3.1. Implementasi Fungsi *insertion*

Fungsi insertion akan menambahkan objek yang baru masuk ke daftar objek. Pada saat objek baru masuk, objek baru akan ditambahkan bersama objek lain yang sudah berada di jalan raya(Kodoe Sumber 4.6 baris 1) lalu dilakukan perhitungan *skyline*(Kode Sumber 4.6 baris 2).

```
insertion(obj,current_counter,threshold)
1  append obj to obj_list
2  get_result(q,current_counter,threshold)
```

Kode Sumber 4.6 *insertion* pada algoritma *naive*

4.3.2. Implementasi Fungsi *deletion*

Fungsi deletion akan menghapus objek yang keluar dari daftar objek. Pada saat objek baru keluar, objek tersebut akan dihapus dari daftar objek di jalan raya(Kode Sumber 4.7 baris 1) lalu dilakukan perhitungan *skyline*(Kode Sumber 4.7 baris 2).

```
deletion(i,current_counter,threshold)
1  remove obj with id=i from obj_list
2  get_result(q,current_counter,threshold)
```

Kode Sumber 4.7 *deletion* pada algoritma *naive*

4.3.3. Implementasi Fungsi *get_result*

Fungsi *get_result* akan melakukan pencarian *skyline* pada waktu tertentu. Untuk setiap objek o dapatkan objek-objek yang mendominasi o secara non-spasial(Kode Sumber 4.8 baris 2). Apabila tidak ada objek yang mendominasi secara non-spasial tambahkan o sebagai *skyline*(Kode Sumber 4.8 baris 4) namun apabila tidak maka tambahkan *o* ke O_{cand}. Selanjutnya untuk setiap objek di O_{cand} dicek apakah didominasi secara non-spasial(Kode Sumber 4.8 baris 9-15). Apabila o tidak didominasi secara spasial dan bukan skyline tambahkan o ke *skyline*(Kode Sumber 4.8 baris 17). Apabila *o* didominasi secara spasial dan *o* dianggap *skyline*, hapus *o* dari *skyline*(Kode Sumber 4.8 baris 20).

```
get_result(q,current_counter,t)
1   for each obj in obj_list:
2       find dom_of(obj) with threshold=t
3       if len(dom_of(obj))==0:
4           append obj.id to result
5       else:
6           append obj.id to Ocand
7       end if
8   end for
9   for each id in Ocand:
10      for each j in dom_of[i]:
11          dominating_by_distance=True
12          if d(obj_list[i],q)>d(obj_list):
13              dominating_by_distance=False
14          end if
15      end for
16      if dominating_by_distance=True and obj not
    in result:
17          append i to result
18      else:
19          if i in result:
20              remove i from result
21          end if
22      end if
23  return result
```

Kode Sumber 4.8 *get_result*

[Halaman ini sengaja dikosongkan]

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada sistem yang dikembangkan. Pengujian yang dilakukan adalah pengujian terhadap performa algoritma.

5.1. Lingkungan Pengujian Sistem

Lingkungan pengujian sistem pada pengerjaan Tugas Akhir ini dilakukan pada lingkungan dan alat kakas seperti yang tertera:

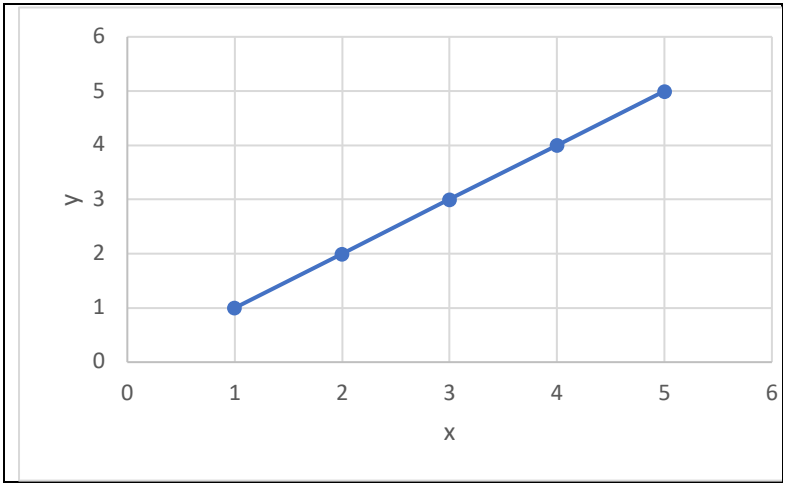
Perangkat Keras	- Processor Intel core i3 - Memory 8 Gb
Perangkat Lunak	- Sistem Operasi Windows 8.1

5.2. Jenis Data Pengujian

Pada pengujian ini akan digunakan tiga jenis data yaitu *correlated data*, *anticorrelated data*, dan *independent data.* Ketiga jenis data tersebut akan digambarkan pada grafik yang memiliki dua sumbu yaitu sumbu x dan sumbu y dimana semakin kecil sumbu x dan semakin kecil sumbu y berarti semakin baik nilai tersebut.

5.2.1. *Correlated Data*

Correlated data adalah data yang memiliki hubungan antara data yang satu dengan data yang lain. Dalam penelitian ini berarti suatu objek memiliki peluang besar untuk mendominasi objek lain. Dengan hubungan dominasi yang kuat tersebut akan mengakibatkan jumlah objek yang menjadi *skyline* sedikit. Adapun *correlated data* dapat digambarkan oleh Gambar 5.1.

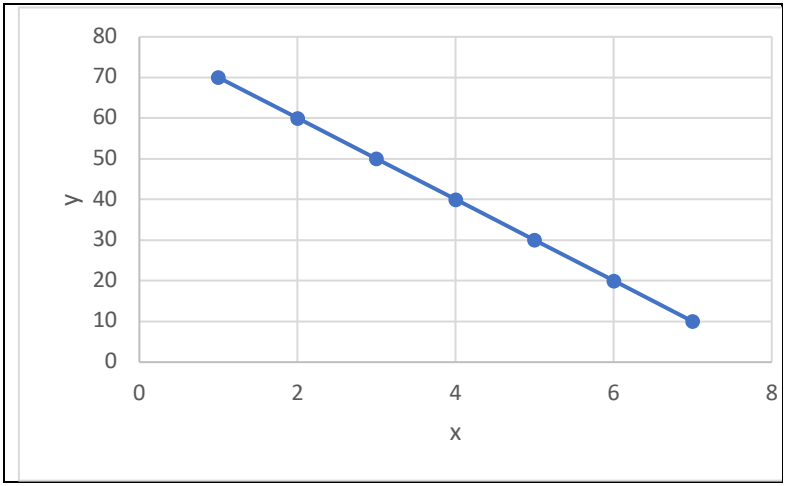


Gambar 5.1 Representasi *correlated data*

Pada grafik di atas kita dapat menyimpulkan bahwa titik di koordinat (1,1) mendominasi titik yang lain. Dengan kata lain yang menjadi *skyline* hanya titik (1,1). *Correlated data* akan mengakibatkan objek yang menjadi *skyline* menjadi sedikit. *Skyline* yang sedikit akan membuat algoritma CNO menjadi lebih optimal dibandingkan dengan jenis data yang lain. Namun, *correlated data* tidak efektif untuk algoritma *naive*. Pada dasarnya setiap objek pada *correlated data* memiliki hubungan dominasi yang besar sehingga besar kemungkinan suatu objek mendominasi objek lain. Sedangkan semakin besar hubungan dominasi antar objek akan membuat performa algoritma *naive* menurun.

5.2.2. *Anticorrelated Data*

Anticorrelated data adalah data yang tidak memiliki hubungan dengan data yang lain. Pada penelitian ini berarti suatu objek memiliki tidak memiliki peluang besar untuk mendominasi objek lain. Karena hal itu maka jumlah objek yang menjadi *skyline* lebih banyak dibandingkan *correlated data.* Adapun *anticorrelated data* dapat digambarkan oleh Gambar 5.2.



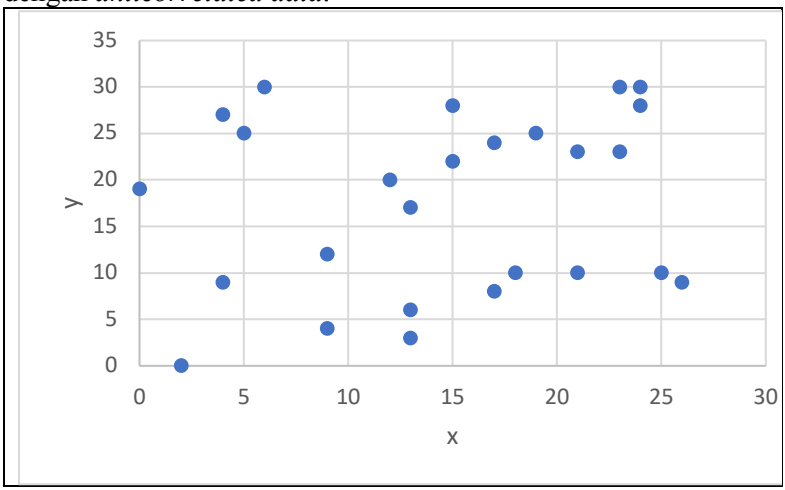
Gambar 5.2 Representasi *anticorrelated data*

Pada grafik di atas, titik pada koordinat (1,70) tidak mendominasi titik pada koordinat(2,60) walaupun titik pada koordinat (1,70) memiliki nilai x lebih baik(lebih kecil) dibandingkan dengan titik pada koordinat (2,60). Begitu juga titik pada koordinat (2,60) tidak mendominasi titik pada koordinat (3,50). Dengan begitu semua titi k pada grafik diatas adalah *skyline. Anticorrelated data* akan membuat objek yang menjadi *skyline* banyak. *Skyline* yang banyak akan membuat performa algoritma CNO kurang bagus dibandigkan dengan *correlated data*. Berkebalikan dengan correlated data, hubungan domiinasi antar objek tidak besar sehingga kecil kemungkinan suatu objek mendominasi objek lain. Yengan kondisi yang sedemikian rupa membuat algoritma *naive* lebih efektif pada *anticorrelated data*.

5.2.3. *Independent Data*

Independent data adalah data yang mimiliki persebaran merata. Dibandingkan dengan *correlated data*, objek-objek pada *independent data* memiliki peluang lebih kecil untuk mendominasi

objek lain, namun memiliki peluang lebih besar untuk mendominasi objek lain dibandingkan dengan *anticorrelated data*. Adapun independent data dapat digambarkan dengan Gambar 5.3. *Independent data* akan mengakibatkan objek yang menjadi *skyline* lebih banyak dibandingkan dengan *correlated data* namun lebih sedikit dibandingkan dengan *anticorrelated data*. Karena itu performa algoritma CNO pada *independent data* tidak lebih bagus dibandingkan *correlated data* namun lebih bagus dibandingkan dengan *anticorrelated data*.



Gambar 5.3 Representasi *independent data*

Independent data memiliki hubungan dominasi lebih kecil dibandingkan dengan *correlated data* namun memiliki hubungan dominasi lebih besar dibandingkan dengan *anticorrelated data*. Hal tersebut membuat performa algoritma *naive* pada *independent data* tidak sebagus *anticorrelated data* namun juga tidak seburuk pada *correlated data*.

5.3. Parameter Pengujian

Pengujian dilakukan terhadap beberapa parameter yaitu jumlah objek(n), jumlah dimensi(d), jumlah *instances*(i). Adapun detail parameter pengujian seperti pada tabel

Paramter	*default*	Rentang
Jumlah objek(n)	1000	500,1000,2000,4000,8000
Jumlah dimensi(d)	3	3,5,7,9,11
Jumlah *instances*	5	3,4,5,6,7

Tabel 5.1 Tabel parameter pengujian

5.4. Hasil Pengujian

Subbab ini membahas mengenai hasil pengujian performa dan memori terhadap parameter dan jenis data yang sudah dijelaskan sebelumnya.

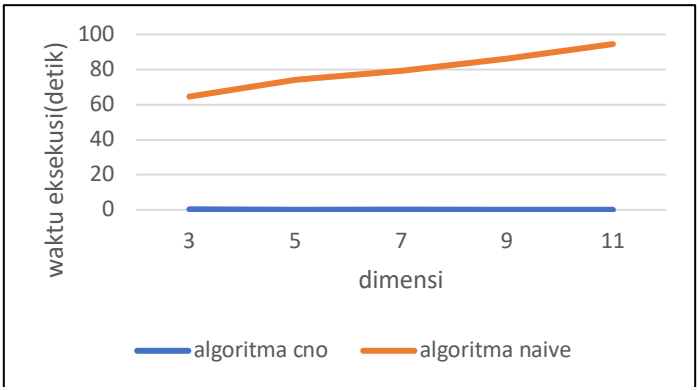
5.4.1. Pengujian Terhadap Jumlah Dimensi

Pengujain dilakukan dengan jumlah objek *default* yaitu 1000 dan jumlah *instances default* yaitu 5. Adapun hasil pengujian terhadap jumlah dimensi adalah sebagai berikut:

5.4.1.1. *Correlated Data*

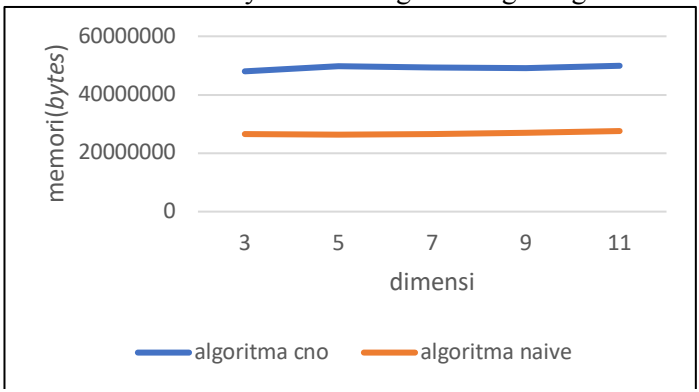
Pengujian dilakukan terhadap jumlah dimensi pada *correlated data.*

- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.4 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil. Hasil pengujian menunjukkan bahwa algoritma CNO memiliki rata-rata waktu eksekusi lebih baik hingga 26 kali lipat dibandingkan dengan algoritma *naive*.



Gambar 5.4 Hasil Pengujian Waktu Eksekusi terhadap Jumlah Dimensi pada *Correlated Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.5 menunjukkan bahwa semakin bertambah dimensi tidak begitu berpengaruh terhadap konsumsi memori baik pada algoritma CNO ataupun algoritma *naive.* Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*.

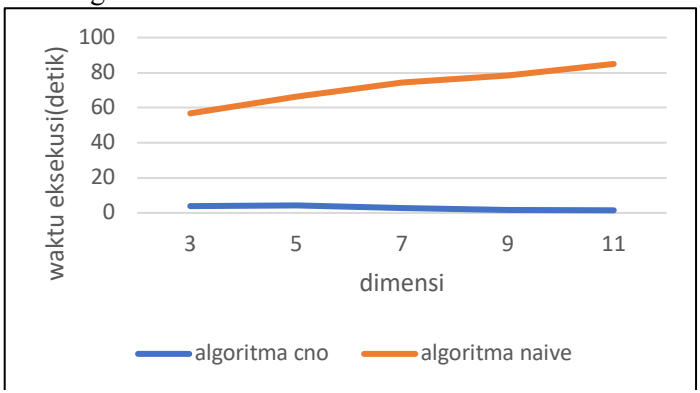


Gambar 5.5 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah Dimensi pada *Correlated Data*

5.4.1.2. *Anticorrelated data*

Pengujian dilakukan terhadap jumlah dimensi pada *anticorrelated data.*

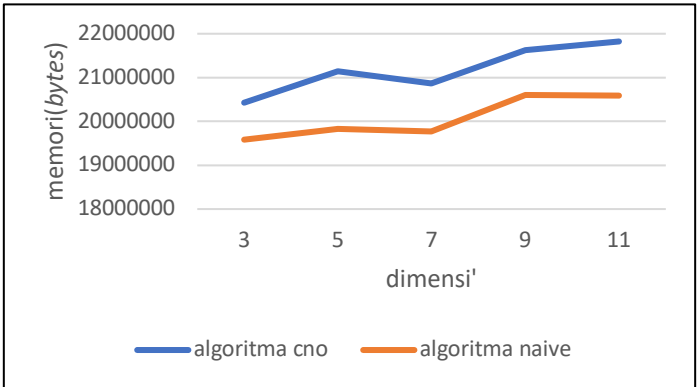
- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.6 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil. Dari hasil pengujian menunjukkan bahwa algoritma CNO memiliki waktu eksekusi lebih cepat hingga 400 kali lipat dibandingkan dengan algoritma naive.



Gambar 5.6 Hasil Pengujian Waktu Eksekusi terhadap Jumlah Dimensi Pada *Anticorrelated Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.7 menunjukkan bahwa semakin bertambah dimensi akan membuat penggunaan memori menjadi meningkat baik pada algoritma CNO maupun algoritma *naive*. Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*. Penggunaan memori pada *anticorrelated data* cenderung tidak stabil dan memiliki kecenderungan naik seiring

bertambahnya dimensi berbeda dengan *correlated data* yang cenderung stabil seiring bertambahnya dimensi.

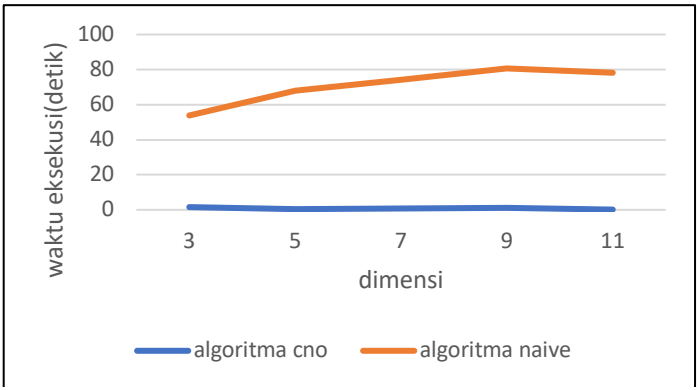


Gambar 5.7 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah Dimensi pada *Anticorrelated Data*

5.4.1.3. *Independent Data*

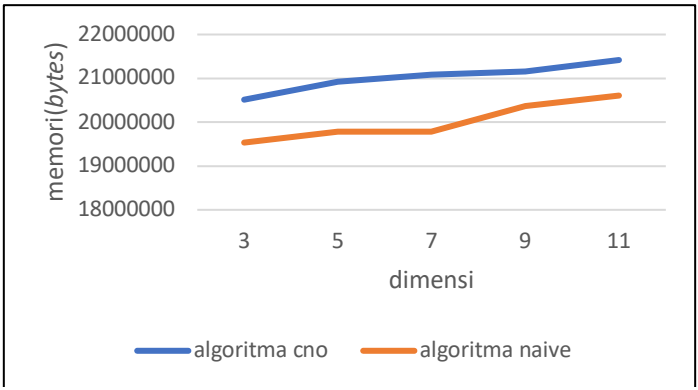
Pengujian dilakukan terhadap jumlah dimensi pada *independent data.*

- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.8 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil. Dari hasil pengujian menunjukkan bahwa algoritma CNO memiliki waktu eksekusi lebih cepat hingga 93 kali lipat dibandingkan dengan algoritma *naive*.



Gambar 5.8 Hasil Pengujian Waktu Eksekusi terhadap Jumlah Dimensi Pada *Independent Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.9 menunjukkan bahwa semakin bertambah dimensi akan membuat penggunaan memori menjadi meningkat baik pada algoritma CNO maupun algoritma *naive*. Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*. Penggunaan memori pada *independent data* cenderung tidak stabil dan memiliki kecenderungan naik seiring bertambahnya dimensi berbeda dengan *correlated data* yang cenderung stabil seiring bertambahnya dimensi.



Gambar 5.9 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah Dimensi pada Independent Data

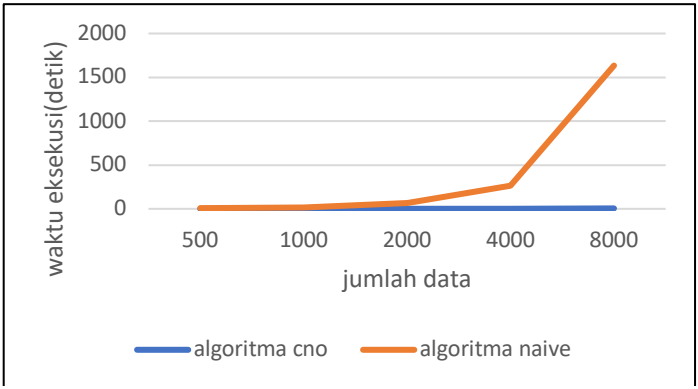
5.4.2. Pengujian Terhadap Jumlah Objek

Pengujain dilakukan dengan jumlah dimensi *default* yaitu 3 dan jumlah *instances default* yaitu 5. Adapun hasil pengujian terhadap jumlah dimensi adalah sebagai berikut:

5.4.2.1. *Correlated Data*

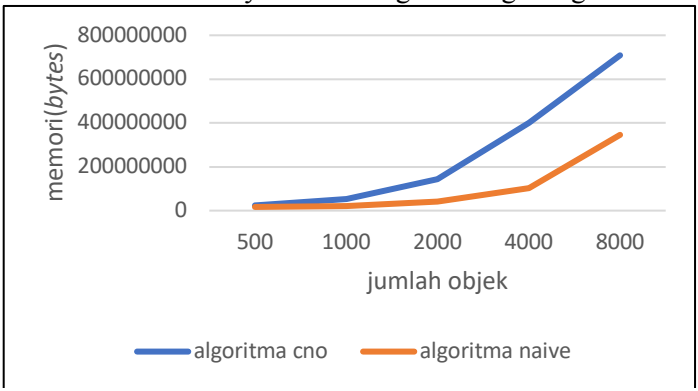
Pengujian dilakukan terhadap jumlah objek pada *correlated data.*

- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.10 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil.



Gambar 5.10 Hasil Pengujian Waktu Eksekusi terhadap Jumlah Objek pada *Correlated Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.11 menunjukkan bahwa semakin bertambah dimensi akan membuat penggunaan memori menjadi meningkat baik pada algoritma CNO maupun algoritma *naive*. Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*.

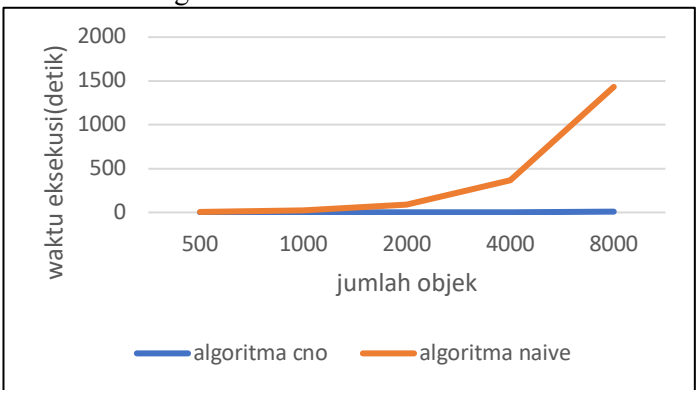


Gambar 5.11 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah Objek pada *Correlated Data*

5.4.2.2. *Anticorrelated Data*

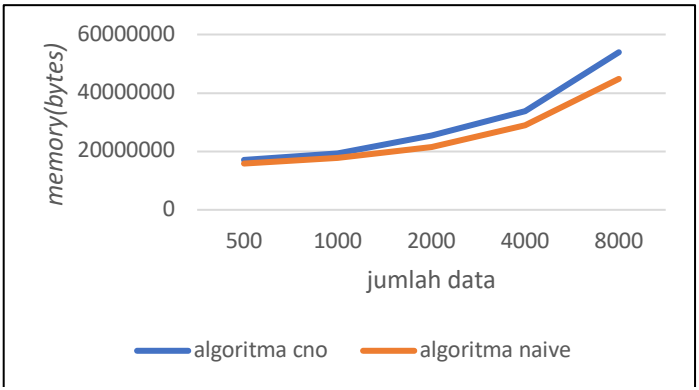
Pengujian dilakukan terhadap jumlah objek pada *anticorrelated data.*

- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.12 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil.



Gambar 5.12 Hasil Pengujian Waktu Eksekusi terhadap Jumlah Objek pada *Anticorrelated Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.13 menunjukkan bahwa semakin bertambah dimensi akan membuat penggunaan memori menjadi meningkat baik pada algoritma CNO maupun algoritma *naive*. Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*.

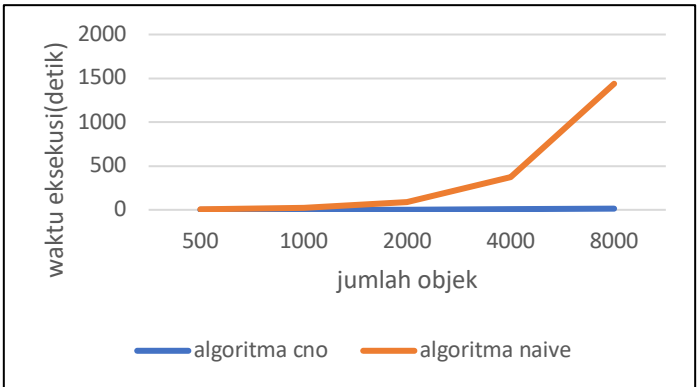


Gambar 5.13 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah Objek pada *Anticorrelated Data*

5.4.2.3. *Independent Data*

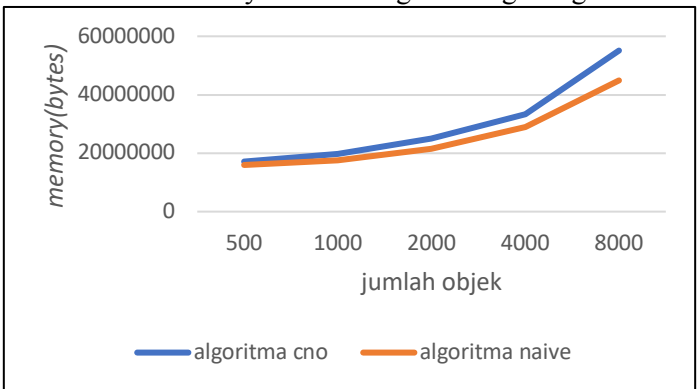
Pengujian dilakukan terhadap jumlah objek pada *independent data.*

- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.14 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil.



Gambar 5.14 Hasil Pengujian Waktu Eksekusi terhadap Jumlah Objek pada *Independent Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.15 menunjukkan bahwa semakin bertambah dimensi akan membuat penggunaan memori menjadi meningkat baik pada algoritma CNO maupun algoritma *naive*. Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*.



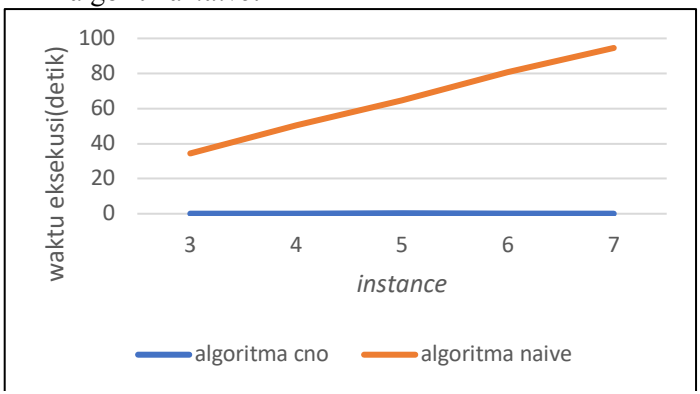
Gambar 5.15 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah Objek pada *Independent Data*

5.4.3. Pengujian Terhadap Jumlah *Instance*

5.4.3.1. *Correlated Data*

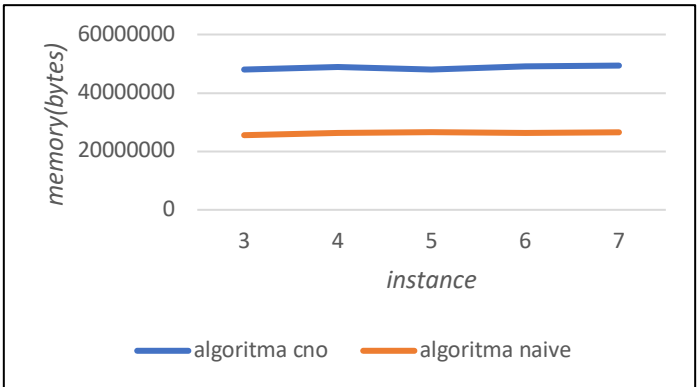
Pengujian dilakukan terhadap jumlah *instance* pada *correlated data.*

- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.16 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil. Hasil pengujian menunjukkan bahwa algoritma CNO memiliki rata-rata waktu eksekusi lebih baik hingga 400 kali lipat dibandingkan dengan algoritma *naive*.



Gambar 5.16 Hasil Pengujian Waktu Eksekusi terhadap Jumlah *Instances* pada *Correlated Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.17 menunjukkan bahwa semakin bertambah dimensi tidak begitu berpengaruh terhadap konsumsi memori baik pada algoritma CNO ataupun algoritma *naive.* Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*.

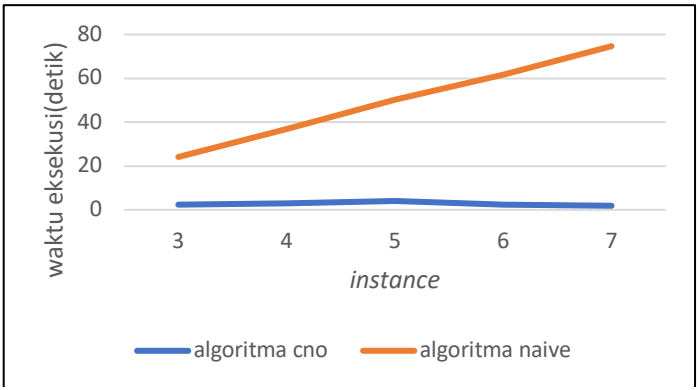


Gambar 5.17 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah *Instances* pada *Correlated Data*

5.4.3.2. *Anticorrelated Data*

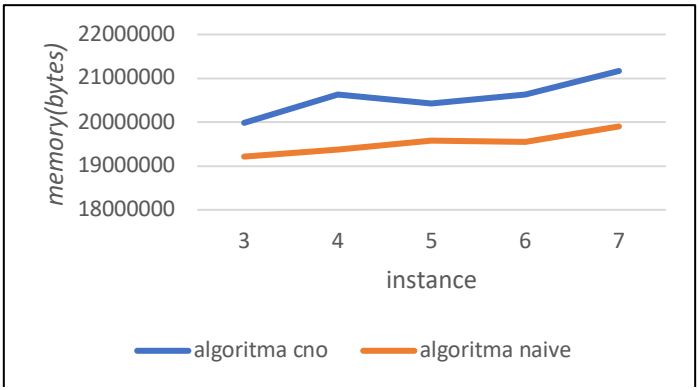
Pengujian dilakukan terhadap jumlah *instances* pada *anticorrelated data.*

- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.16 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil. Hasil pengujian menunjukkan bahwa algoritma CNO memiliki rata-rata waktu eksekusi lebih baik hingga 18 kali lipat dibandingkan dengan algoritma *naive*.



Gambar 5.18 Hasil Pengujian Waktu Eksekusi terhadap Jumlah *Instances* pada *Anticorrelated Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.7 menunjukkan bahwa semakin bertambah dimensi akan membuat penggunaan memori menjadi meningkat baik pada algoritma CNO maupun algoritma *naive*. Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*. Penggunaan memori pada *anticorrelated data* cenderung tidak stabil dan memiliki kecenderungan naik seiring bertambahnya dimensi berbeda dengan *correlated data* yang cenderung stabil seiring bertambahnya dimensi.

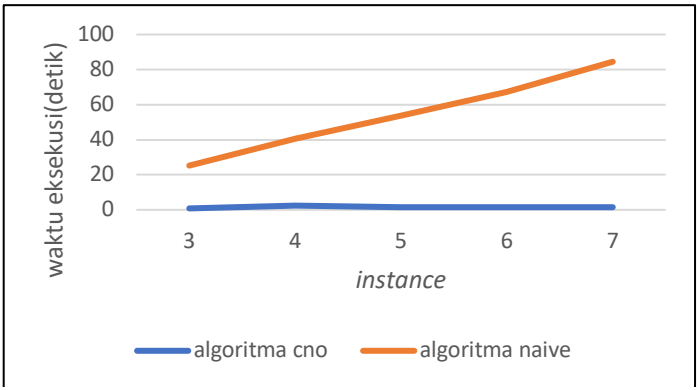


Gambar 5.19 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah *Instances* pada *Anticorrelated Data*

5.4.3.3. *Independent Data*

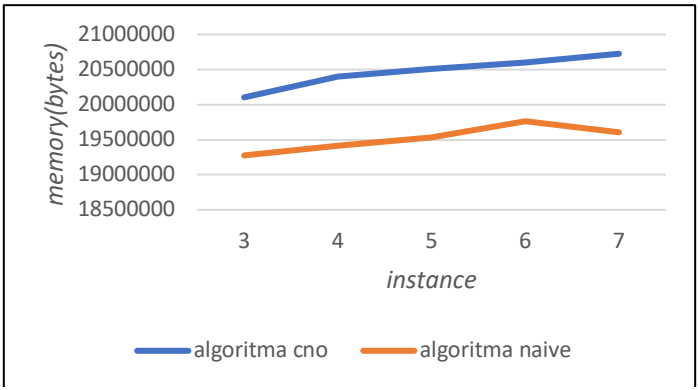
Pengujian dilakukan terhadap jumlah *instances* pada *independent data.*

- Hasil Pengujian Berdasarkan Waktu Eksekusi
  Hasil pengujian sesuai Gambar 5.16 menunjukkan bahwa semakin bertambah dimensi mengakibatkan waktu eksekusi algoritma *naive* cenderung bertambah lama. Berbeda dengan algoritma *naive*, algoritma CNO cenderung lebih stabil. Hasil pengujian menunjukkan bahwa algoritma CNO memiliki rata-rata waktu eksekusi lebih baik hingga 35 kali lipat dibandingkan dengan algoritma *naive*.



Gambar 5.20 Hasil Pengujian Waktu Eksekusi terhadap Jumlah *Instances* pada *Independent Data*

- Hasil Pengujian Berdasarkan Penggunaan Memori
  Hasil pengujian sesuai Gambar 5.21 menunjukkan bahwa semakin bertambah dimensi akan membuat penggunaan memori menjadi meningkat baik pada algoritma CNO maupun algoritma *naive*. Dari hasil pengujian menunjukkan bahwa konsumsi memori pada algoritma CNO lebih banyak dibandingkan dengan algoritma *naive*. Penggunaan memori pada *independent data* cenderung tidak stabil dan memiliki kecenderungan naik seiring bertambahnya dimensi berbeda dengan *correlated data* yang cenderung stabil seiring bertambahnya dimensi.



Gambar 5.21 Hasil Pengujian Penggunaan *Memory* terhadap Jumlah *Instances* pada *Independent Data*

5.5. *Kesimpulan Pengujian*

Berdasarkan pengujian pada subbab 5.4, algoritma berbasis CNO memiliki waktu eksekusi yang lebih cepat dibandingkan dengan algoritma *naive*. Akan tetapi algoritma *naive* menggunakan media penyimpanan atau memori lebih sedikit dibandingkan dengan algoritma berbasis CNO.

BAB VI
KESIMPULAN DAN SARAN

Bab ini akan memaparkan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1. Kesimpulan

1. Penulis menyarankan algoritma CNO dibandingkan dengan algoritma *naive*. Algoritma CNO membuat penjadwalan untuk setiap objek sebelum program berjalan. Saat program berjalan tidak semua objek akan diproses ulang untuk mengurangi biaya komputasi.
2. Biaya komputasi algoritma CNO jauh lebih baik dibandingkan dengan algoritma *naive* yaitu 100 kali lebih cepat.
3. Hubungan dominasi antar objek sangat mempengaruhi performa algoritma CNO maupun *naive*.

6.2. Saran

1. Penelitian ini dapat dikembangkan untuk objek yang dapat berbelok arah.
2. Pengembangan juga dapat dilakukan pada algoritma untuk menentukan dominasi antara dua objek.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] Stephan Börzsönyi, Donald Kossmann dan Konrad Stocker, "The Skyline Operator," dalam *Proceedings of the 17th International Conference on Data Engineering*, Washington, 2001.

[2] Yuan-Ko Huang, Chia-Heng Chang dan Chiang Lee, "Continuous distance-based skyline queries in road networks," *Information Systems,* vol. 37, no. 7, pp. 611-633, 2012.

[3] Miss Pragati Pandey, Miss Preteeksha Pandey dan Mrs. Minu Choundary, "Uncertain Data Algorithms and Applications," *International Journal of Advanced Research in,* vol. 2, no. 7, pp. 274-280, 2012.

[4] James Bornholt, Todd Mytkowicz dan Kathryn S. McKinley, "Uncertain<T>: A First-Order Type for Uncertain Data," dalam *ASPLOS '14 Architectural Support for Programming Languages and Operating Systems*, Salt Lake City, 2014.

[5] "PyCharm: Python IDE for Professional Developers by Jetbrains," Jetbrains, [Online]. Available: https://www.jetbrains.com/pycharm/. [Diakses 6 6 2018].

[6] I. Efendi, "Pengertian Struktur Data," [Online]. Available: https://www.it-jurnal.com/pengertian-struktur-data/. [Diakses 13 July 2018].

[7] M. A. Basri, "Pengertian Jaringan Client Server Beserta Kelebihan dan Kekurangannya," [Online]. Available: https://www.nesabamedia.com/pengertian-jaringan-client-server/. [Diakses 14 7 2018].

[8] C. Shi, X. Qin dan L. Wang, "Continous Skyline Queries for Moving Objects in Road Networks," *Journal of Software,* 2015.

[9] J. Pei, B. Jiang, X. lin dan Y. Yuan, "VLDB '07 Proceedings of the 33rd international conference on Very large data bases," dalam *VLDB*, Vienna, Austria, 2007.

[10] Fang Wei-Kleiner, "Finding nearest neighbors in road networks: A tree decomposition method," dalam *EDBT '13 Proceedings of the Joint EDBT/ICDT 2013 Workshops*, New York, 2013.

[11] Baihua Zheng, Cindy Chen dan Ken C.K. Lee, "Efficient Index-Based Approaches for Skyline Queries in Location-Based Applications," *IEEE Transactions on Knowledge and Data Engineering,* vol. 25, no. 11, pp. 2507 - 2520, 2013.

[12] H. T. Kung, F. Luccio dan F. P. Preparata, "On Finding the Maxima of a Set of Vectors," dalam *Journal of the ACM(JACM)*, New York, 1975.

[13] G. Koutrika dan Y. Loannidis, "Personalization of queries in database systems," dalam *Data Engineering, 2004. Proceedings. 20th International Conference on*, Boston, 2004.

[14] Werner Kießling, "Foundations of Preferences in Database Systems," dalam *VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases*, Hong Kong, 2002.

[15] Mohammed E. Khalefa, Mohammed F. Mokbel dan Justin J. Levandoski, "Skyline Query Processing for Uncertain Data," dalam *CIKM '10 Proceedings of the 19th ACM international conference on Information and knowledge management*, New York, 2010.

[16] "What is Python? Executive Summary," python.org, [Online]. Available: https://www.python.org/doc/essays/blurb/. [Diakses 20 3 2018].

[17] “Client-server architecture,” [Online]. Available: https://www.britannica.com/technology/client-server-architecture. [Diakses 20 3 2018].

[18] “TraCI,” [Online]. Available: http://sumo.dlr.de/wiki/TraCI. [Diakses 20 3 2018].

[19] “SUMO – Simulation of Urban MObility,” DLR - Institute of Transportation System, [Online]. Available: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/. [Diakses 20 3 2018].

[Halaman ini sengaja dikosongkan]

LAMPIRAN

```
1  comparator
2     if(a[2]>b[2]):
3        return 1
4     else:
5        return -1
```

Kode Sumber 0.1 ***comparator***

```
1   def init_dom_by_of_object(id, removed_id):
2      global dom_of
3      global dom_by
4      for key in obj_list:
5         if key != removed_id and id in dom_of[key]:
6            if id not in dom_by:
7               dom_by[id] = [key]
8            if key not in dom_by[id]:
9               dom_by[id].append(key)
```

Kode Sumber 0.2 ***init_dom_by_of_object***

```
1   def init_dom_by_of_object(id, removed_id):
2      global dom_of
3      global dom_by
4      for key in obj_list:
5         if key != removed_id and id in dom_of[key]:
6            if id not in dom_by:
7               dom_by[id] = [key]
8            if key not in dom_by[id]:
9               dom_by[id].append(key)
```

Kode Sumber 0.3 ***init_dom_by_of_objet***

```
1    def init_dom_by():
2       global dom_of
3       global lmsky
4       global dom_by
```

```
5       for dominating_id in lmsky:
6           for dominated_id in dom_of:
7               if dominating_id in dom_of[dominated_id]:
8                   dom_by[dominating_id] = list()
9                dom_by[dominating_id].append(dominated_id)
```

Kode Sumber 0.4 *init_dom_by*

```
1   def get_event_between(map_generator,
    obj_list,dom_of,o1, o2, start, end):
2       if (o2.id in dom_of and o1.id not in dom_of[o2.id])
    and (o1.id in dom_of and o2.id not in dom_of[o1.id]):
3           return []
4       else:
5           obj1range=map_generator.get_range_at(o1,start)
6           obj2range=map_generator.get_range_at(o2,start)
7           if obj1range > obj2range:
8               closer=o2
9               farther=o1
10          elif obj2range > obj1range:
11              closer=o1
12              farther=o2
13          else:
14              closer=o2
15              farther=o1
16          event_list_temp=[]
17          for t in range(start, end+1):
18              if o1.tnext>t:
19                  if t==o2.tnext and t!=o1.tnext and
    map.get_range_at(o2,t-1)<=map.get_range_at(o1,t-1):
20                      event_list_temp.append((o2.id,o1.id,t))
21                      return event_list_temp
22                  if closer==o1 and
    map_generator.get_range_at(closer, t) >=
    map_generator.get_range_at(farther, t):
23                      if t<closer.tnext and t<farther.tnext:
```

```
24                  event_list_temp.append
    ((closer.id,farther.id,t))
25                  closer, farther = farther, closer
26              elif closer==o2 and
    map_generator.get_range_at(closer, t) >
    map_generator.get_range_at(farther, t):
27                  if t<closer.tnext and t<farther.tnext:
28                      event_list_temp.append
    ((closer.id,farther.id,t))
29                  closer, farther = farther, closer
30          return event_list_temp
```

Kode Sumber 0.5 ***get_event_between***

```
1   def init_result():
2       global result
3       global lmsky
4       result=[]
5       for key,value in lmsky.iteritems():
6           if value[0] == []:
7               result.append(key)
```

Kode Sumber 0.6 ***init_result***

```
1   def
    update_index_after_insert(obj,obj_list,dom_of,dom_by,
    lmsky,logging,threshold):
2       dom_of[obj.id]=[]
3       dom_of_new,dom_by_new=
    dominance_determinator.generate_new_uncertain_dom_
    of_obj(obj, obj_list, threshold)
4       dom_of[obj.id]=dom_of_new
5       dom_by[obj.id] = dom_by_new
6       obj_list[obj.id] = obj
7       for i in dom_by_new:
8           if obj.id not in dom_of[i]:
9               dom_of[i].append(obj.id)
```

```
10      for i in dom_of_new:
11          if obj.id not in dom_by[i]:
12              dom_by[i].append(obj.id)
```

Kode Sumber 0.7 ***update_index_after_insert***

```
1   def update_index_after_leave(id,obj_list,dom_of,dom_by,
    result,lmsky):
2       dom_of.pop(id)
3       obj_list.pop(id)
4       if id in result:
5           result.remove(id)
6       if id in dom_by:
7           dom_by.pop(id)
8       if id in lmsky:
9           lmsky.pop(id)
10
11      for key in dom_of:
12          if id in dom_of[key]:
13              dom_of[key].remove(id)
14
15      for key in dom_by:
16          if id in dom_by[key]:
17              dom_by[key].remove(id)
```

Kode Sumber 0.8 ***update_index_after_leave***

```
1   def read_from_params():
2       start = 0
3       end = 0
4       d = 0
5       threshold = 0.0
6       n = 0
7       if len(sys.argv) == 1:
8           print("Sertakan parameter daftar object, start time, dan end
    time")
9           sys.exit()
```

```
10      elif len(sys.argv) == 8:
11          type = sys.argv[1]
12          start = int(sys.argv[2])
13          end = int(sys.argv[3])
14          n = int(sys.argv[4])
15          d = int(sys.argv[5])
16          i = int(sys.argv[6])
17          threshold = float(sys.argv[7])
18          path = "../../data/dataset/%s/input/d%s/i%s/%s_
        uncertain_data.json" % (type, d, i, n)
19          out = open("../../data/dataset/%s/reports/d%s/
        i%s/%s_implementation.txt" % (type, d, i, n), "w")
20          obj_ins = insertion_object_extractor.get_insertion_
        object(type, n,d, i)
21          obj_del = deletion_object_extractor.get_deletion_
        object(type,n,d,i)
22              json_object = json.load(file)
23              for value in json_object:
24                  obj = Object()
25                  obj.id = value['id']
26                  obj.dis = value['dis']
27                  obj.tnext = value['tnext']
28                  obj.instances = value['instances']
29                  obj_list[obj.id] = obj
30          return obj_list, start, end, n, d, threshold, out, obj_ins,
        obj_del
31      else:
32          with open(sys.argv[1], "r") as file:
33              json_object = json.load(file)
34              for value in json_object:
35                  obj = Object()
36                  obj.id = value['id']
37                  obj.eid = value['eid']
38                  obj.v = value['v']
39                  obj.dis = value['dis']
40                  obj.tnext = value['tnext']
41                  obj.instances = value['instances']
```

```
42              obj_list[obj.id] = obj
43          start = int(sys.argv[2])
44          end = int(sys.argv[3])
45          threshold = float(sys.argv[4])
46          return obj_list, start, end, threshold
```

Kode Sumber 0.9 ***read_from_params***

```
1   def loop(start, end,obj_ins,obj_del,counter):
2       out.write("================================
    ============\n")
3       print("=====================================
    ============================================
    ========================")
4       global lmsky
5       global result
6       global obj_list
7
8       print("detik ke %s"%start)
9       out.write("detik ke %s\n"%start)
10
11      if start in obj_ins:
12          start_time=time.time()
13          if obj_ins[start].tnext>start:
14              insertion(obj_ins[start],start,end)
15          counter.append_insertion_time(time.time()-start_time)
16          obj_ins.popitem()
17
18      if start in obj_del:
19          start_time=time.time()
20          deletion(obj_del[start],start,end)
21          counter.appen_deletion_time(time.time() - start_time)
22
23      for key in lmsky:
24          if start in lmsky[key][0]:
25              result.append(key)
26          if start in lmsky[key][1]:
27              if key in result:
```

```
28                  result.remove(key)
29
30      out.write(str(result)+"\n")
31      print(result)
32      if start==end:
33          out.write("counter: %s\n" % counter.__dict__)
34          print("insertion average:%s" %
    counter.get_insertion_time_average())
35          print("deletion average:%s" %
    counter.get_deletion_time_average())
36          out.write("insertion average:%s\n" %
    counter.get_insertion_time_average())
37          out.write("deletion average:%s\n" %
    counter.get_deletion_time_average())
38          pid = os.getpid()
39          ps = psutil.Process(pid)
40          out.write("memory usage:%s" % ps.memory_info().rss)
41          out.close()
42          sys.exit()
43      threading.Timer(0, loop,[start+1, end,
    obj_ins,obj_del,counter]).start()
44
```

Kode Sumber 0.10 *loop*

```
1   def generate_uncertain_dom(obj_list,dom_of,dom_by,
    threshold):
2       objs = []
3       for key,value in obj_list.iteritems():
4           objs.append(value)
5       x = len(objs)
6       counter=0
7       for i in range(x):
8           for j in range(i+1, x):
9               dominance_status = determine_uncertain_dominance
    (objs[i], objs[j], threshold)
10              if dominance_status[0]==1:
11                  dom_of[objs[j].id].append(objs[i].id)
12                  if dom_by!=None:
```

```
                dom_by[objs[i].id].append(objs[j].id)
            elif dominance_status[0]==-1:
                dom_of[objs[i].id].append(objs[j].id)
                if dom_by!=None:
                    dom_by[objs[j].id].append(objs[i].id)
    return dom_of
```

Kode Sumber 0.11 ***generate_uncertain_dom***

BIODATA PENULIS

Cahya Setya Adhi adalah nama penulis laporan ini. Penulis lari dari pasangan orang tua Sugino dan Harmini. Penulis dilahirkan di Kabupaten Sukoharjo pada tanggal 30 Desember 1995. Kehidupan pendidikan sekolah dasar penulis harus berpindah-pindah mengikuti perjuangan orang tua. Namun kehidupan Pendidikan dasar penulis berakhir di SD N Karanganyar 3 yang terletak di Kabupaten Sukoharjo, tepatnya di Kelurahan Karanganyar. Penulis lulus dari SDN Karangayar 3 pada tahun 2009, melanjutkan Pendidikan ke SMPN 1 Sukoharjo (2009-2012), lalu melanjutkan pendidikan ke SMAN 1 Sukoharjo (2012-2014), dan sekarang sedang menjalani pendidikan S1 Teknik Informatika di ITS. Penulis aktif di forum daerah Ikemas Surabaya. Penulis mengambil bidang minat Komputasi Berbasis Jaringan(KBJ) dan Mobile Developer. Akhir kata penulis mengucapkan rasa syukur yang sebesar-besarnya atas terselesaikan laporan kerja praktik ini. Komunikasi dengan penulis dapat melalui email: **cahya.setyaa@gmail.com**

.