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

Syukron Rifai'il Muttaqi, Royyana Muslim Ijtihadie, dan Bagus Jati Santoso
Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: syukronrifai@gmail.com[1)], roy@if.its.ac.id[2)], bagus@if.its.ac.id[3)]

***Abstrak*—Semakin banyaknya pengguna teknologi mendorong proses pencarian data semakin cepat. Diantara permasalahan mengenai pemrosesan data yaitu pencarian objek terbaik pada data spasial. Pemrosesan data pada jaringan jalan raya menjadi berbeda jika lokasi titik *query* yang berpindah-pindah. Dalam beberapa kasus, data objek berupa *uncertain data* dan bersifat *streaming*.**

Artikel ini mengusulkan algoritme *Continuous Streaming* d_ε-*distance Skyline Query*, sebuah algoritme untuk memproses *Skyline Query* pada jaringan jalan raya dengan *uncertain data streaming*. Algoritma ini menggunakan d_ε sebagai jarak maksimal objek dapat menjadi objek terbaik dari titik *query*. Hasil uji coba menunjukkan bahwa metode yang diusulkan memiliki waktu komputasi 600 kali lebih cepat dan penggunaan memori lebih hemat 1500 kali dibandingkan metode *naive*.

***Kata kunci*—*skyline query*, *uncertain data streaming*, jaringan jalan raya**

I. PENDAHULUAN

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

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

Banyaknya teknologi yang digunakan sekarang membuat *uncertain data* semakin banyak. *Query* data terunggul dari *uncertain data* memerlukan metode tersendiri untuk menyelesaikannya.

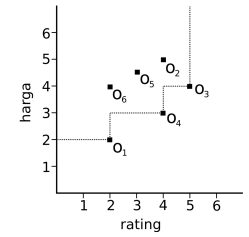
Diantara algoritme pencarian data paling unggul adalah metode *skyline*, termasuk pada jaringan jalan raya. Metode ini dapat diterapkan pada jaringan jalan raya dengan menggunakan *uncertain data*. Struktur data khusus diperlukan agar proses pencarian data/*query* dapat dilakukan secara *real time*.

Artikel ini membahas mengenai metode pencarian objek yang menjadi *skyline* pada jaringan jalan raya. Masing-masing objek berupa *uncertain* data dan bersifat streaming.

II. TINJAUAN PUSTAKA

A. Skyline

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



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

Gambar 1. Contoh *skyline* sederhana

B. Skyline Query pada Jaringan Jalan Raya

Skyline query tradisional seperti pada Gambar 1 hanya menggunakan atribut statis sebagai penentuan *SP*, yaitu atribut ranking dan hotel. Jika *skyline query* ditempatkan pada jaringan jalan raya seperti pada Gambar 2.3, maka perlu menambahkan jarak sebagai salah satu atribut dalam menentukan *SP*.

Pada Gambar 1, hotel yang menjadi *SP* adalah o_1, o_2 dan o_3. Ketika hotel o_1 hingga o_5 dimasukkan dalam jaringan jalan raya seperti pada Gambar 2, o_5 tidak lagi didominasi oleh o_4 sehingga o_5 bergabung menjadi bagian dari *SP*. Hal tersebut dikarenakana jarak antara titik *query* q dengan o_5 tidak didominasi oleh o_1, o_2 maupun o_3. Atribut jarak ini akan berubah-ubah sesuai dengan jarak titik *query* q dengan masing-masing objek. Oleh karena itu, atribut jarak disebut sebagai atribut dinamis [1].

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

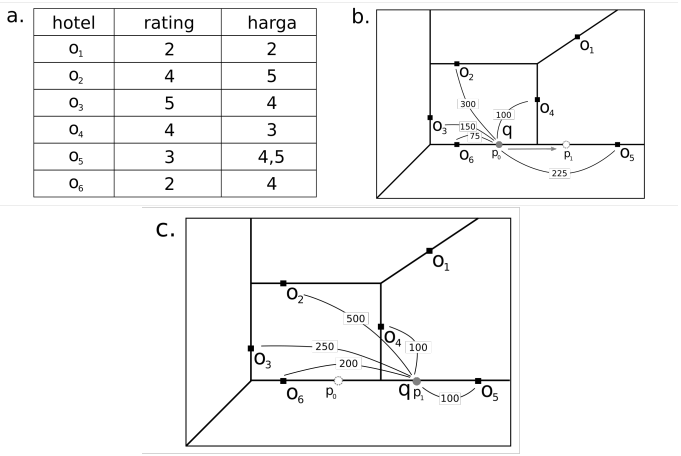
Pada kasus tertentu objek *SP* memiliki jarak yang sangat jauh dari titik *query*. Objek tersebut seringkali tidak diinginkan karena kita mencari titik *query* yang dekat karena untuk mencapai objek yang jauh membutuhkan biaya lebih besar dalam hal transportasi.

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

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

a.

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



Gambar 2. Contoh *skyline* pada jaringan jalan raya

Gambar 2 mengilustrasikan pemrosesan *skyline* pada jaringan jalan raya. Gambar 2(a) menampilkan atribut statis dari keempat objek. Contoh tersebut mensimulasikan pencarian $d_\varepsilon - SP$ pada titik *query* q yang bergerak dari titik p_1 ke p_2. Ketika q berada pada koordinat p_1(lihat Gambar 2(b)), anggota dari $d_\varepsilon - SP$ hanyalah o_2. o_4 tidak dapat menjadi $d_\varepsilon - SP$ karena didominasi oleh o_2 dalam hal atribut statis(harga dan rating) maupun dinamis(jarak). o_1 dan o_3 tidak tergabung dalam $d_\varepsilon - SP$ karena memiliki jarak lebih dari d_ε. Selanjutnya, seperti yang terlihat pada Gambar 2(c), titik *query* q bergerak ke kiri sampai tepat di titik p_2 sehingga jarak antara titik *query* q dengan o_2 dan o_4 menjadi sama. Ketika titik *query* q bergerak ke kiri, o_4 menjadi lebih dekat kepada q dibandingkan o_2. Dengan demikian, o_2 tergabung menjadi $d_\varepsilon - SP$ karena jarak o_2 tidak lagi didominasi oleh o_4. Selanjutnya $d_\varepsilon - SP$-nya adalah o_2 dan o_4.

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

C. Uncertain Data

Pemrosesan *uncertain data* mendapat banyak perhatian pada beberapa tahun terakhir. *Uncertain data* dapat ditemukan pada berbagai bidang, seperti jaringan sensor, jaringan RFID, sistem pelacakan lokasi menggunakan GPS, dan sosial media [4]. Beberapa perangkat memang menghasilkan data yang cenderung tidak pasti (*uncertain*). Sebagai contoh, data yang dihasilkan oleh sensor cenderung tidak pasti karena hilangnya data ketika transmisi, galat pada perangkat sensor itu sendiri atau karena lingkungan yang berubah-ubah [2].

Seringkali data pada lingkungan bersifat dinamis dan kontinu. Sebagai contoh, pada jaringan sensor, *gateway sensor*

mengirim hasil secara kontinu, pemantauan cuaca mendapatkan data secara kontinu dengan komputasi waktu nyata. Hal tersebut menjadi tantangan tersendiri, yaitu pemrosesan *uncertain data streaming* dengan efektif dan efisien sehingga hasil didapatkan di waktu itu juga [2].

III. METODE

Bab ini memaparkan mengenai struktur data grid indeks serta algoritma yang digunakan pada struktur data tersebut.

Tabel I
RINGKASAN NOTASI

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

A. Struktur Data

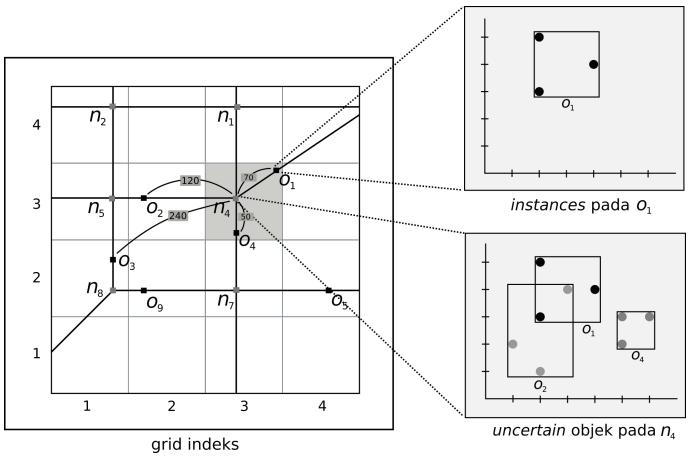
Jaringan jalan raya menggunakan jalan dan persimpangan sebagai objek utamanya. Hal tersebut dapat tersebut dapat dimodelkan dengan *undirected weighted graph* yang terdiri dari *node* dan *edge*. *Node* sebagai persimpangan dan *edge* sebagai jalan. Struktur ini efisien untuk pencarian data pada peta.

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

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

Tabel II
OBJEK

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



Gambar 3. Struktur data grid indeks

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

Tabel III
NODE

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

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

Tabel IV
METADATA

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

Tabel IV menyimpan data yang melekat pada objek ketika objek sudah masuk pada *edge* e. Tabel tersebut menyimpan

jarak d, yaitu jarak antara *node* dengan objek. Dengan adanya d, algoritme yang diusulkan tidak menggunakan komputasi *shortest-path*. *skyProb* menyimpan probabilitas objek menjadi bagian dari *SP* dan *skyProb* bernilai $0 \leq skyProb \leq 1$. Terakhir, *isImpossible* adalah *flag* yang menjadi tanda apabila objek tidak lagi dapat menjadi bagian dari *SP*.

Tabel V
EDGE

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

B. Metode Pemrosesan $CSd_\varepsilon - SQ$

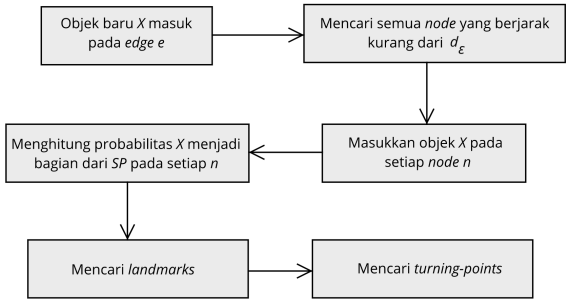
Artikel ini mengusulkan metode *Continuous Streaming distance-based Skyline Query* ($CSd_\varepsilon - SQ$). Pencarian titik *skyline* pada jaringan jalan raya menggunakan algoritme $Cd_\varepsilon - SQ$ [1]. Untuk mencari *skyline point*, SP dari suatu titik *query*, komputasi dilakukan untuk mencari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

Algorithm 1 DetermineGSP

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

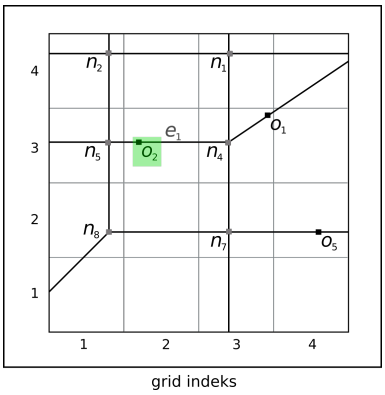
Gambar 4. Algoritme *DetermineGSP*

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



Gambar 5. Alur pemrosesan

Perhatikan Gambar 5 dan Gambar 6, saat objek X yang masuk dari *stream* pada suatu *edge* yang terdapat pada struktur Grid, algoritme BFS mencari semua *node* yang berjarak kurang dari d_ε dari objek X, $d_{X,n} \leq d_\varepsilon$.



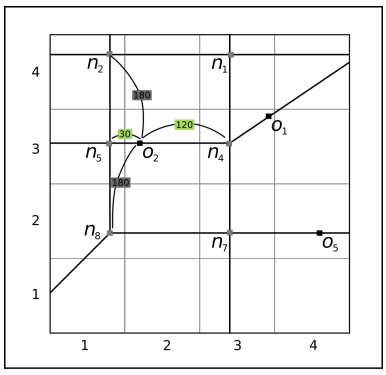
Gambar 6. Objek masuk pada grid indeks

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

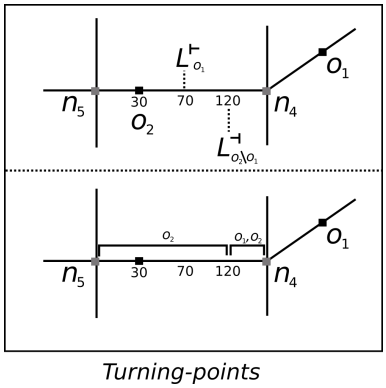
Algorithm 2 *Insertion*

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

Gambar 7. Algoritme *Insertion*



Gambar 9. Dengan d_ε=150, o_2 hanya masuk pada *node* n_4 dan n_5



Gambar 10. Atas: *landmark* yang didapat dari *edge*. Bawah: *turning-point* didapat dari *landmark*

.

Algorithm 3 *Deletion*

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

Gambar 8. Algoritme *Deletion*

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

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

Algorithm 4 ComputeTurningPoint

1: **Input:** threshold p, a distance d_ε, a set *GSP*, an edge e connecting two nodes n_s and n_e
2: **Output:** A set of tuples in form of $< [n_i, n_j], SP^\varepsilon >$ where SP^ε is the $d_\varepsilon - SP$ set between $[n_i, n_j]$
3: create an empty queue Q
4: **for** object $o \in$ *GSP* **do**
5: determine o's landmarks $L^{\vdash}_o$ and $L^{\dashv}_o$
6: **if** $L^{\vdash}_o$ is on e **then** insert $L^{\vdash}_o$ into Q
7: **if** $L^{\dashv}_o$ is on e **then** insert $L^{\dashv}_o$ into Q
8: **for** object $o' \in (GSP - \{o\}) \cap \sum_{q \text{ in } MBR(o') \cap DDR(o)} Pr(q) > (1-p)$ **do**
9: determine o's landmarks $L^{\vdash}_o$ or $L^{\dashv}_o$
10: **if** $L^{\vdash}_{o\setminus o'}$ is on e and $L^{\vdash}_o$ is closer to n_s than $L^{\vdash}_{o\setminus o'}$ **then**
11: insert $L^{\vdash}_{o\setminus o'}$ into Q
12: **if** $L^{\dashv}_{o\setminus o'}$ is on e and $L^{\dashv}_{o\setminus o'}$ is closer to n_s than $L^{\dashv}_o$ **then**
13: insert $L^{\dashv}_{o\setminus o'}$ into Q
14: sort landmarks in Q in ascending order of their distances to n_s
15: /* determining the result turning points */
16: **while** Q is not empty **do**
17: dequeue $o.L$
18: **switch** $o.L$ **do**
19: **case** $L^{\vdash}_o$
20: **if** there is no $L^{\dashv}_{o'\setminus o}$ **then**
21: return $< [n_i, L^{\vdash}_o], SP^\varepsilon >, n_i = L^{\vdash}_o$,
22: and add o into SP^ε
23: **case** $L^{\dashv}_o$
24: **if** $o \in SP^\varepsilon$ **then**
25: return $< [n_i, L^{\dashv}_o], SP^\varepsilon >, n_i = L^{\dashv}_o$,
26: and remove o from SP^ε
27: **case** $L^{\vdash}_{o\setminus o'}$
28: **if** $o \in SP^\varepsilon$ and $o' \in SP^\varepsilon$ **then**
29: return $< [n_i, L^{\vdash}_{o\setminus o'}], SP^\varepsilon >, n_i = L^{\vdash}_{o\setminus o'}$,
30: and remove o' from SP^ε
31: **otherwise**
32: **if** $o \in SP^\varepsilon$ and there is no $L^{\dashv}_{o''\setminus o'} \in Q$ **then**
33: return $< [n_i, L^{\dashv}_{o\setminus o'}], SP^\varepsilon >$,
34: $n_i = L^{\dashv}_{o\setminus o'}$, and add o' into SP^ε

Gambar 11. Algoritme *Compute Turning Point*

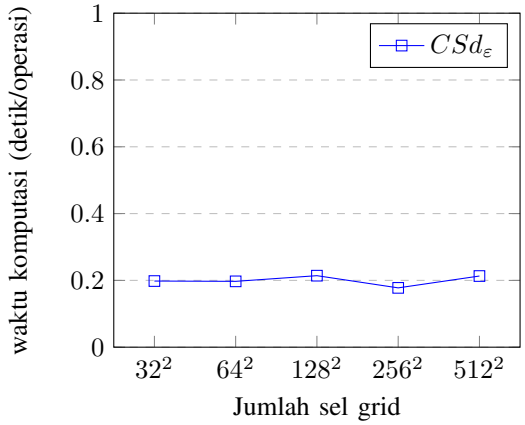
turning-point.

Algoritma pada Gambar 11 mengilustrasikan penentuan *landmark* pada *edge* e yang memiliki *node* awal n_s dan *node* lainnya n_e. Queue Q dibuat untuk menampung *landmark*. Penentuan landmark ini berdasarkan pada objek-objek yang menjadi anggota dari *GSP*. Setelah mendapatkan semua landmark, *queue* Q diurutkan berdasarkan jarak terdekat dari *node* n_s.

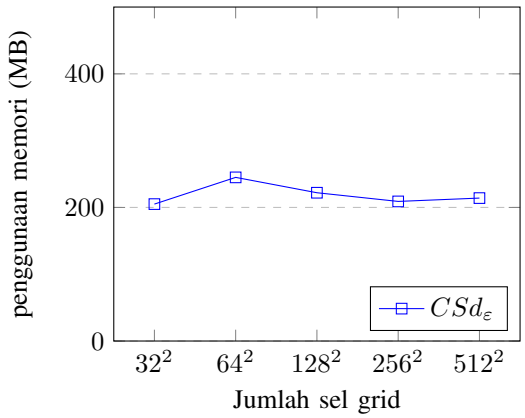
Setelah semua *landmark* didapatkan, pencarian *turning-point* dilakukan. *Turning-point* ini adalah hasil dari algoritma. *Turning-point* berupa sekumpulan *tuple* yang merepresentasikan interval awal n_i, interval akhir n_e, dan objek-objek yang menjadi SP pada interval tersebut SP^ε.

IV. Uji Coba

Uji coba dilakukan pada jaringan jalan raya California [6] dengan *node* sebanyak 8716 dan *edge* sebanyak 9077. Algoritma diimplementasikan menggunakan bahasa Scala de-



Gambar 12. Pengaruh jumlah sel terhadap waktu komputasi tiap operasi dalam satuan detik



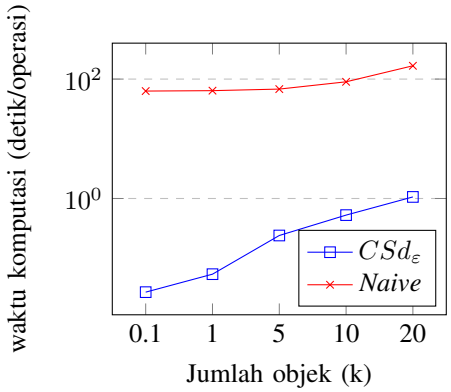
Gambar 13. Pengaruh jumlah sel grid terhadap penggunaan memori dalam satuan megabita

ngan *memory heap* sejumlah 4 GB. Pengujian dilakukan pada komputer dengan *Processor* Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz x 4 dan RAM 6 GB. Pengujian dilakukan untuk mengetahui performa dengan menggunakan waktu komputasi dan untuk mengetahui penggunaan memori pada setiap eksekusi. Uji coba juga dilakukan pada tiga jenis data, yaitu data *independent*, *correlated*, dan *anticorrelated*.

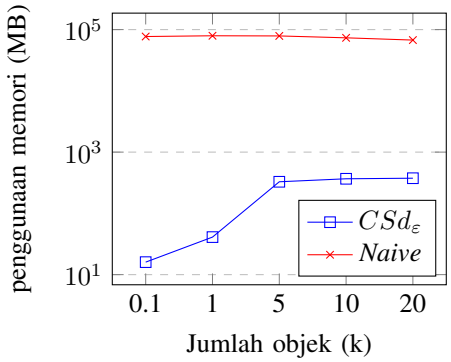
Tabel VI
VARIASI PENGUJIAN

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

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



Gambar 14. Pengaruh jumlah objek terhadap waktu komputasi tiap operasi dalam satuan detik



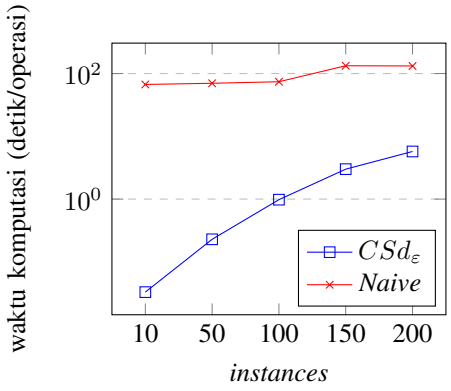
Gambar 15. Pengaruh jumlah objek terhadap penggunaan memori dalam satuan megabita

Tetapi di sisi lain, sistem tidak banyak mencari data secara berulang-ulang karena setiap sel sudah mengaver area yang besar. Sedangkan grid indeks yang memiliki sel yang banyak menjadikan proses komputasi lebih efisien karena melibatkan data yang lebih sedikit. Tetapi di sisi lain, sistem harus melakukan pencarian data berulang-ulang karena sedikitnya data yang didapat pada setiap sel.

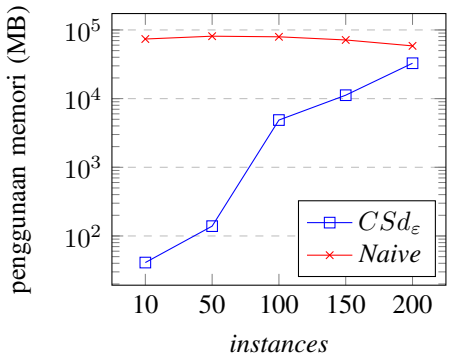
Ketika jumlah objek bertambah, waktu pemrosesan juga bertambah, hal ini dikarenakan bertambahnya objek yang terdapat pada *node*. Dengan bertambahnya objek pada *node*, algoritme perlu membandingkan dengan objek yang lebih banyak untuk mencari probabilitas masing-masing objek menjadi *SP*.

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

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



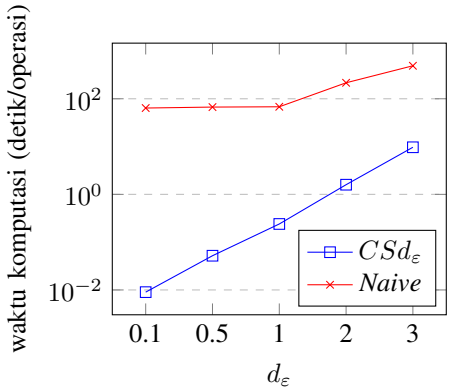
Gambar 16. Pengaruh jumlah *instance* terhadap waktu komputasi tiap operasi dalam satuan detik



Gambar 17. Pengaruh jumlah *instance* terhadap penggunaan memori dalam satuan megabita

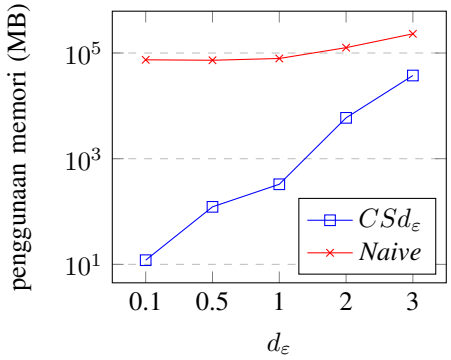
Pada metode *naive*, objek perubahan waktu komputasi terlihat ketika jumlah *instance* diatas 100. Hal ini dikarenakan waktu komputasi lebih banyak digunakan untuk penghitungan jarak terpendek dari setiap *node* ke objek, sehingga jumlah *instance* yang sedikit tidak berpengaruh banyak terhadap waktu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

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



Gambar 18. Pengaruh d_ε terhadap waktu komputasi tiap operasi dalam satuan detik

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



Gambar 19. Pengaruh d_ε terhadap penggunaan memori dalam satuan megabita

V. KESIMPULAN

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

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

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

Berikut beberapa saran terkait pengembangan struktur data dan algoritma lebih lanjut:

1) Pendefinisian jarak d_ε dapat dilakukan secara dinamis. Apabila pencarian objek dengan jarak d_ε tidak menemukan hasil yang diminta, jarak d_ε dapat diperbesar secara dinamis hingga mendapatkan hasil yang sesuai.
2) Pengembangan algoitma untuk memproses objek *uncertain* yang dapat bergerak secara dinamis.
3) Pada algoritme ini proses pembaruan *instance* dari *uncertain* objek dilakukan dengan menghapus dan menambahkan objek baru. Hal ini tentunya tidak efisien. Diperlukan algoritme pembaruan objek agar lebih efisien dalam hal waktu komputasi dan penggunaan memori.

PUSTAKA

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

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

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

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

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

[6] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, Shang-Hua Teng, "On Trip Planning Queries in Spatial Databases", *Advances in Spatial and Temporal Databases*, vol 3633, pp 273-290, 2005