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

Hafara Firdausi, Bagus Jati Santoso, Henning Titi Ciptaningtyas
Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: hafarafirdausi@gmail.com[1)], bagus@if.its.ac.id[2)], henning@if.its.ac.id[3)]

***Abstrak*—Kemajuan ilmu pengetahuan dan teknologi telah mempengaruhi cara perusahaan dalam menjalankan bisnis, yaitu dengan mengumpulkan data preferensi pelanggan dari data penjualan produk, kemudian memanfaatkannya untuk mendapatkan informasi yang dapat digunakan untuk membuat keputusan bisnis yang tepat. Saat ini, sudah ada strategi pemilihan produk dengan melakukan pencarian k-produk yang paling banyak diminati oleh pelanggan, yaitu *k-Most Promising Products (k-MPP)*. Sayangnya, komputasi k-MPP tidak mempertimbangkan variabel waktu dalam algoritme perhitungannya dan tidak dapat digunakan untuk memproses kueri berbasis interval waktu.**

Artikel ini mengusulkan algoritme k-MPPTI *(k-Most Promising Products in Time Intervals)* untuk menjawab permasalahan k-MPP berbasis interval waktu pada data multidimensi dengan serial waktu. Ada tiga jenis algoritme yang dibuat dan dibandingkan, yaitu k-MPPTI (menggunakan dua tipe kueri *skyline*, yaitu *dynamic skyline* dan *reverse skyline*), k-MPPTI NoRSL (menggunakan kueri *dynamic skyline* saja), dan k-MPPTI paralel (menggunakan teknik komputasi paralel). Efektivitas dan efisiensi algoritme diuji menggunakan data asli dan sintetis. Hasil uji coba menunjukkan bahwa algoritme k-MPPTI NoRSL dapat memberikan hasil dengan waktu eksekusi lima kali lebih cepat dan penggunaan memori satu kali lebih hemat dibandingkan dengan algoritme k-MPPTI.

***Kata kunci*—Strategi pemilihan produk, Kueri, *Dynamic skyline*, *Reverse skyline*, Interval waktu**

I. PENDAHULUAN

Pesatnya kemajuan ilmu pengetahuan dan teknologi telah mempengaruhi cara perusahaan dalam menjalankan bisnis, yaitu dengan mengumpulkan data preferensi pelanggan dari data penjualan produk, kemudian memanfaatkannya secara cerdas untuk mendapatkan informasi yang dapat digunakan untuk membuat keputusan bisnis yang tepat. Misalnya, dengan mendapatkan informasi k-produk yang paling diminati oleh pelanggan beserta fitur-fiturnya, perusahaan dapat menentukan harga produk baru yang akan diluncurkan atau menentukan fitur apa yang diunggulkan dari produk baru yang hendak diproduksi.

Saat ini, sudah ada penelitian yang mengembangkan strategi pemilihan produk dengan melakukan pencarian k-produk yang paling banyak diminati oleh pelanggan. Islam dan Liu (2010) memodelkannya sebagai kueri *k-Most Promising Products* (k-MPP) [1] serta membuat kerangka kerja algoritme untuk memproses kueri tersebut. Komputasi k-MPP menggunakan dua tipe kueri *skyline*, yaitu *dynamic skyline* [2] dan *reverse skyline* [3]. Kueri *dynamic skyline* digunakan untuk mengambil data produk terbaik berdasarkan sudut pandang pelanggan, sedangkan kueri *reverse skyline* digunakan untuk mengambil data pelanggan potensial berdasarkan sudut pandang produk atau perusahaan [1].

Sayangnya, komputasi k-MPP tidak mempertimbangkan variabel waktu dalam algoritme perhitungannya sehingga informasi yang didapatkan kurang valid dengan kondisi yang sebenarnya. Komputasi k-MPP juga tidak dapat memproses kueri berbasis interval waktu. Sebagai contoh, pertanyaan yang mungkin diajukan adalah *"k-produk apa yang paling banyak diminati oleh pelanggan pada bulan Februari hingga September?"*. Dalam hal ini, bulan Februari hingga September disebut dengan interval waktu kueri dan data yang berbasis interval waktu disebut dengan data *time series* atau serial waktu.

Sebagai ilustrasi, produk A adalah produk yang paling banyak diminati oleh pelanggan pada bulan Januari hingga Juni, namun posisinya diungguli oleh produk B yang lebih diminati pelanggan pada bulan Juli hingga September. Pada bulan Oktober, produk B tidak diproduksi lagi karena suatu alasan, sehingga produk A kembali diminati pelanggan.

Berdasarkan ilustrasi tersebut, produk yang paling unggul berdasarkan kueri k-MPP adalah produk B karena produk B pernah mengungguli produk A walaupun rentang waktu unggulnya lebih pendek dibandingkan dengan produk A. Hal ini terjadi karena komputasi k-MPP hanya mempertimbangkan skor kontribusi pasar yang dihitung dari banyaknya jumlah pelanggan yang lebih menyukai produk tersebut dibandingkan produk lainnya tanpa mempertimbangkan faktor durasi waktu.

Sedangkan jika berdasarkan kueri dengan interval waktu Januari hingga Juli maka produk yang paling unggul adalah produk A; jika berdasarkan kueri dengan interval waktu Juli hingga Agustus maka produk yang paling unggul ada-

lah produk B; jika berdasarkan kueri dengan interval waktu Januari hingga Desember maka produk yang paling unggul adalah produk A karena rentang waktu unggulnya lebih lama dibandingkan dengan produk B.

Artikel ini membahas metode yang dapat menjawab permasalahan k-MPP berbasis interval waktu pada data multidimensi dengan serial waktu, yaitu dengan memodelkan kueri k-MPPTI *(k-Most Promising Products in Time Intervals)* dan merancang kerangka kerja algoritme yang dapat memproses kueri tersebut. Ada tiga jenis algoritme pemrosesan yang dibuat dan dibandingkan: (a) k-MPPTI, yaitu algoritme yang mengadaptasi komputasi k-MPP asli; menggunakan dua tipe kueri *skyline*, yaitu *dynamic skyline* dan *reverse skyline*; (b) k-MPPTI NoRSL, yaitu algoritme yang hanya menggunakan kueri *dynamic skyline* saja; (c) k-MPPTI paralel, yaitu algoritme k-MPPTI yang mengimplementasikan teknik komputasi paralel supaya pemrosesan data menjadi lebih cepat. Efektivitas dan efisiensi ketiga algoritme diuji menggunakan data asli dan sintetis.

II. Tinjauan Pustaka

A. Data Multidimensi dengan Serial Waktu

Data multidimensi dengan serial waktu adalah data *multi-attribute* yang memiliki *timestamp* dan berurutan menurut waktu, sebagaimana contoh *dataset* produk dan preferensi pelanggan yang ditunjukkan pada Tabel I. Pada tabel tersebut, *timestamp* ditulis sebagai interval waktu yang dinotasikan dengan $[t_i : t_e]$, dengan asumsi bahwa nilai masing-masing atribut konstan setiap waktu.

Tabel I: Contoh *dataset* (a) produk P dan (b) preferensi pelanggan C

(a)

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

(b)

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

B. Skyline

Operasi *skyline* digunakan untuk mencari data yang menarik dari suatu himpunan data, yaitu data yang tidak didominasi oleh data lain. *Skyline* didefinisikan sebagai titik-titik yang tidak didominasi oleh titik lain [6]; titik adalah representasi dari data dalam bidang d-dimensi. Sebuah titik $p_1 \in P$ dikatakan mendominasi titik lain $p_2 \in P$, dinotasikan dengan $p_1 \prec p_2$, jika nilai p_1 baik atau lebih baik dari p_2 pada semua dimensi dan ada nilai p_1 yang lebih baik dari p_2 setidaknya pada satu dimensi. Secara matematis, relasi $p_1 \prec p_2$ dapat terbentuk jika dan hanya jika: (a) $p_1^i \leq p_2^i, \forall i \in [1, ..., d]$ dan (b) $p_1^i < p_2^i, \exists i \in [1, ..., d]$.

C. Dominansi Dinamis

Para ahli menyebut *original skyline* sebagai *static skyline* [7] karena sifat dominansinya yang statis. Sebagai pengembangan, hasil *skyline* dapat berubah berdasarkan titik kueri. Sifat ini disebut dengan dominansi dinamis. Suatu titik $ob_1 \in D$ dikatakan mendominasi objek data $ob_2 \in D$ secara dinamis berdasarkan objek data $ob_3 \in D$, dinotasikan dengan $ob_1 \prec_{ob_3} ob_2$, jika nilai ob_1 dekat dengan ob_3 pada semua dimensi dan ada nilai ob_1 yang lebih dekat dengan ob_3 dibandingkan nilai ob_2 dengan ob_3 minimal pada satu dimensi. Secara matematis, relasi $ob_1 \prec_{ob_3} ob_2$ terbentuk jika dan hanya jika: (a) $|ob_3^i - ob_1^i| \leq |ob_3^i - ob_2^i|, \forall i \in [1, ..., d]$ dan (b) $|ob_3^i - ob_1^i| < |ob_3^i - ob_2^i|, \exists i \in [1, ..., d]$.

D. 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 [**?**]. 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 [**?**].

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 [**?**].

III. Metode

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

Tabel II: 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*

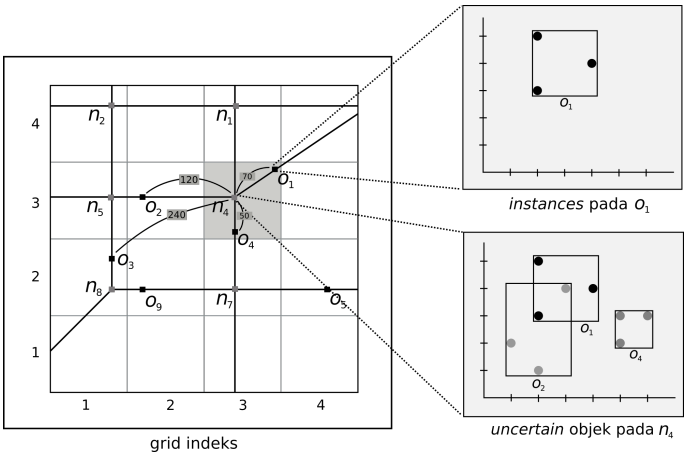
sebagai jalan. Struktur ini efisien untuk pencarian data pada peta.

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 III, 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 III: Objek

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



Gambar 1: Struktur data grid indeks

Perhatikan struktur tabel *node* pada Tabel IV. 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 IV: 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 VI. 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 V: 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 V 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 VI: 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 $CSd_\varepsilon - SQ$

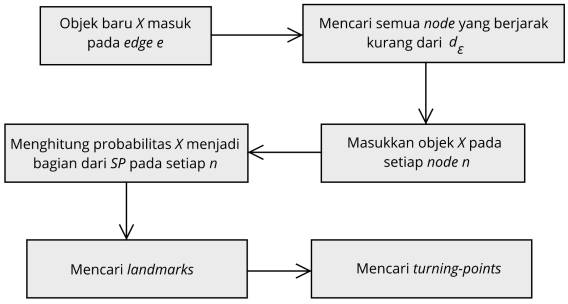
Artikel ini mengusulkan metode *Continuous Streaming distance-based Skyline Query* ($CSd_\varepsilon - SQ$). Pencarian titik *skyline* pada jaringan jalan raya menggunakan algoritme $Cd_\varepsilon - SQ$ [**?**]. Untuk mencari *skyline point*, SP dari suatu titik *query*, komputasi dilakukan untuk mencari 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

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

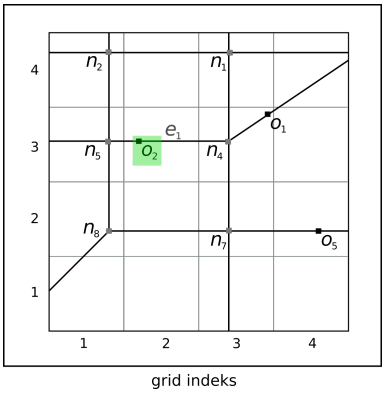
Gambar 2: Algoritme *DetermineGSP*

Untuk menentukan *GSP* pada *uncertain data streaming*, algoritme yang digunakan adalah metode EPSU [**?**]. 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 3: Alur pemrosesan

Perhatikan Gambar 3 dan Gambar 4, 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 4: 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*

Input: uncertain data object X, threshold p
for each object Q overlapped with *PDR(X)* **do**
    if *MBR(Q) is within DDR(X)* **then**
        mark Q as impossible object
    else
        if $\sum_{q\ in\ MBR(Q) \cap DDR(X)} Pr(q) > (1-p)$ **then**
            mark Q as as impossible object
        else
            update SkyPr(Q) with X
compute SkyPr(X) with objects overlapped with *PDD(X)*
insert X into *SW-Tree*

Gambar 5: Algoritme *Insertion*

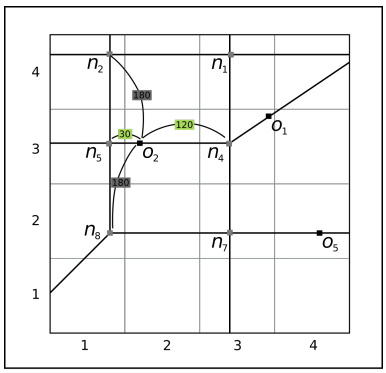
Algorithm 3 *Deletion*

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

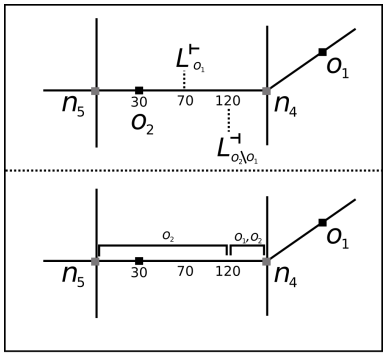
Gambar 6: Algoritme *Deletion*

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.



Gambar 7: Dengan d_ε=150, o_2 hanya masuk pada *node* n_4 dan n_5



Gambar 8: Atas: *landmark* yang didapat dari *edge*. Bawah: *turning-point* didapat dari *landmark*

.

Setiap *node* memiliki objek dan terhubung dengan *edge*. Dari proses *Skyline edge e* direpresentasikan dalam bentuk interval beserta *SP* yang terdapat pada masing-masing interval. Antara interval satu dengan interval lain terdapat pergantian objek yang menjadi *SP*. Titik pergantian *SP* ini diistilahkan dengan *turning-point*.

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 [**?**] dengan *node* sebanyak 8716 dan *edge* sebanyak 9077. Algoritma diimplementasikan menggunakan bahasa Scala de-

Algorithm 4 ComputeTurningPoint

Input: threshold p, a distance d_ε, a set *GSP*, an edge e connecting two nodes n_s and n_e
Output: A set of tuples in form of $< [n_i, n_j], SP^\varepsilon >$ where SP^ε is the $d_\varepsilon - SP$ set between $[n_i, n_j]$
create an empty queue Q
for object $o \in GSP$ **do**
  determine o's landmarks $L^{\vdash}_o$ and $L^{\dashv}_o$
  if $L^{\vdash}_o$ is on e **then** insert $L^{\vdash}_o$ into Q
  if $L^{\dashv}_o$ is on e **then** insert $L^{\dashv}_o$ into Q
  for object $o' \in (GSP - \{o\}) \cap \sum_{q \text{ in } MBR(o') \cap DDR(o)} Pr(q) > (1-p)$ **do**
    determine o's landmarks $L^{\vdash}_o$ or $L^{\dashv}_o$
    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**
      insert $L^{\vdash}_{o\setminus o'}$ into Q
    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**
      insert $L^{\dashv}_{o\setminus o'}$ into Q
sort landmarks in Q in ascending order of their distances to n_s
/* determining the result turning points */
while Q is not empty **do**
  dequeue $o.L$
  switch $o.L$ **do**
    case $L^{\vdash}_o$
      if there is no $L^{\dashv}_{o'\setminus o}$ **then**
        return $< [n_i, L^{\vdash}_o], SP^\varepsilon >, n_i = L^{\vdash}_o$,
        and add o into SP^ε
    case $L^{\dashv}_o$
      if $o \in SP^\varepsilon$ **then**
        return $< [n_i, L^{\dashv}_o], SP^\varepsilon >, n_i = L^{\dashv}_o$,
        and remove o from SP^ε
    case $L^{\vdash}_{o\setminus o'}$
      if $o \in SP^\varepsilon$ and $o' \in SP^\varepsilon$ **then**
        return $< [n_i, L^{\vdash}_{o\setminus o'}], SP^\varepsilon >, n_i = L^{\vdash}_{o\setminus o'}$,
        and remove o' from SP^ε
    otherwise
      if $o \in SP^\varepsilon$ and there is no $L^{\dashv}_{o''\setminus o'} \in Q$ **then**
        return $< [n_i, L^{\dashv}_{o\setminus o'}], SP^\varepsilon >$,
        $n_i = L^{\dashv}_{o\setminus o'}$, and add o' into SP^ε

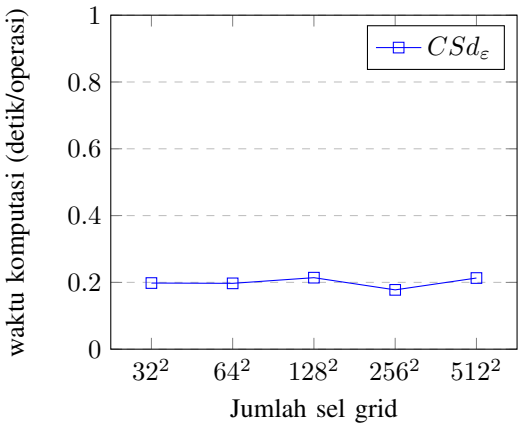
Gambar 9: Algoritme *Compute Turning Point*

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*.

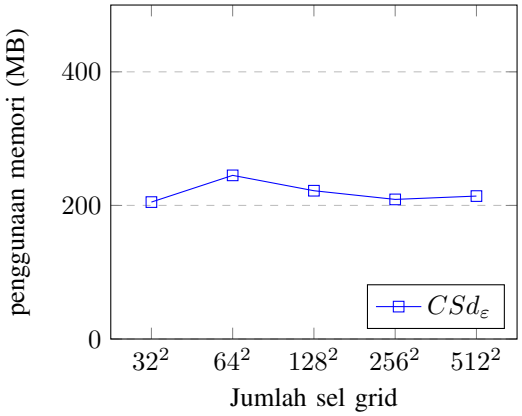
Tabel VII: Variasi pengujian

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*



Gambar 10: Pengaruh jumlah sel terhadap waktu komputasi tiap operasi dalam satuan detik

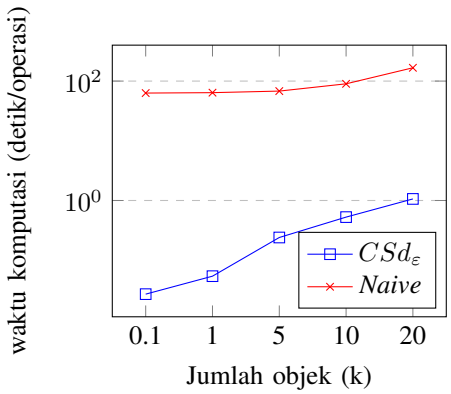


Gambar 11: Pengaruh jumlah sel grid terhadap penggunaan memori dalam satuan megabita

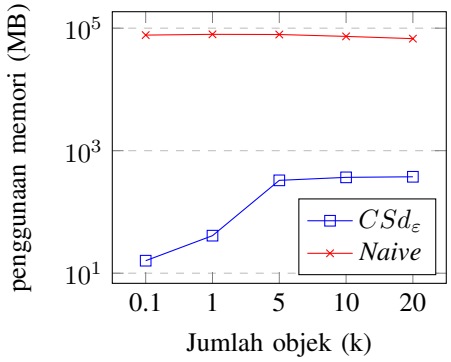
antara proses memuat data dengan komputasi. Grid indeks yang memiliki sel sedikit menjadikan data yang dimuat lebih banyak sehingga menjadikan data yang diproses labih banyak. Tetapi di sisi lain, sistem tidak banyak mencari data secara berulang-ulang karena setiap sel sudah mengaver area yang besar.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

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.

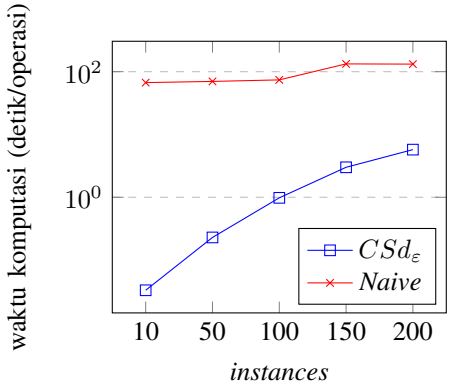


Gambar 12: Pengaruh jumlah objek terhadap waktu komputasi tiap operasi dalam satuan detik

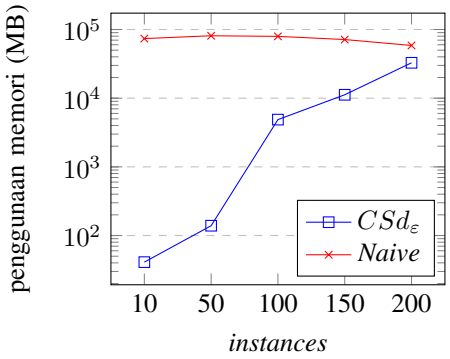


Gambar 13: Pengaruh jumlah objek terhadap penggunaan memori dalam satuan megabita

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.



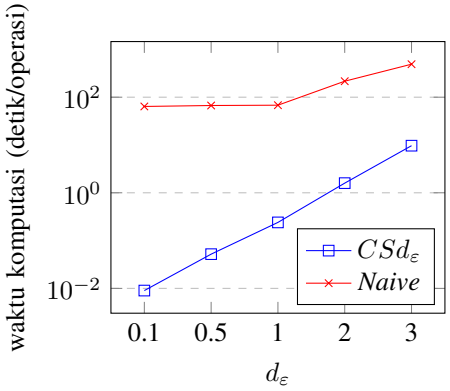
Gambar 14: Pengaruh jumlah *instance* terhadap waktu komputasi tiap operasi dalam satuan detik



Gambar 15: Pengaruh jumlah *instance* terhadap penggunaan memori dalam satuan megabita

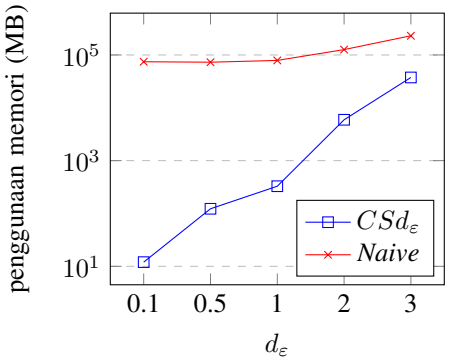
Pada metode *naive*, objek perubahan waktu komputasi terlihat ketika jumlah *instance* diatas 100. Hal ini dikarenakan waktu komputasi lebih banyak digunakan untuk penghitungan jarak terpendek dari setiap *node* ke objek, sehingga jumlah *instance* yang sedikit tidak berpengaruh banyak terhadap waktu 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 16: 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 17: 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] M. S. Islam and C. Liu, "Know Your Customer: Computing K-Most Promising Products," *The VLDB Journal*, pp. 545570, 2016.
[2] D. Papadias, Y. Tao, G. Fu and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM Transactions on Database Systems*, Vol. 30, No. 1, pp. 4182, 2005.
[3] E. Dellis and B. Seeger, "Efficient Computation of Reverse Skyline Queries," *VLDB Endowment*, pp. 291-302, 2007.

[4] B. Jiang and J. Pei, "Online Interval Skyline Queries on Time Series," *IEEE International Conference on Data Engineering*, pp. 1036-1047, 2009.
[5] M. Golfarelli and S. Rizzi, "Introduction to Data Warehousing," in *Data Warehouse Design: Modern Principles and Methodologies*, New York: McGraw-Hill, 2009, pp. 1-42.
[6] S. Borzsonyi, D. Kossmann and K. Stocker, "The Skyline Operator," *In: ICDE*, pp. 421-430, 2001.
[7] L. Zou, L. Chen, M. T. zsu and D. Zhao, "Dynamic Skyline Queries in Large Graphs," *DASFAA'10 Proceedings of the 15th International Conference on Database Systems for Advanced Applications - Volume Part II*, pp. 62-78, 2010.
[8] X. Wu, Y. Tao, R. C.-W. Wong, L. Ding and J. X. Yu, "Finding the Influence Set through Skylines," *EDBT*, pp. 1030-1041, 2009.
[9] G.S. Almasi and A. Gottlieb, *Highly Parallel Computing*. Redwood City, CA: Benjamin-Cummings Publishers, 1989.
[10] J. A. Blackard, D. J. Jean and C. W. Anderson, "UCI Machine Learning Repositories," 1 Agustus 1998. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/covertype. [Accessed 9 Juni 2018].