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, terutama di bidang analisis data, 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 kueri *dynamic skyline* dan *reverse skyline*), k-MPPTI NoRSL (menggunakan kueri *dynamic skyline* saja), dan k-MPPTI NoRSL-P (menggunakan teknik komputasi paralel). Efektivitas dan efisiensi algoritme diuji menggunakan data asli dan sintetis. Hasil uji coba menunjukkan bahwa algoritme k-MPPTI NoRSL memiliki performa yang lebih baik daripada algoritme k-MPPTI karena dapat memberikan hasil kueri 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 hendak diunggulkan dari produk baru yang ingin dibuat.

Saat ini, sudah ada penelitian yang mengembangkan strategi pemilihan produk dengan melakukan pencarian k-produk yang paling banyak diminati oleh pelanggan. Dalam [1], Islam et al. memodelkannya sebagai kueri *k-Most Promising Products* (k-MPP) 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 daripada 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 adalah 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 daripada 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 NoRSL-P, yaitu algoritme k-MPPTI NoRSL 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. Dalam pengembangannya, hasil *skyline* dapat berubah bergantung pada titik kuerinya (dominansi dinamis). Suatu titik $ob_1 \in D$ dikatakan mendominasi titik $ob_2 \in D$ secara dinamis berdasarkan titik kueri $ob_3 \in D$, 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. Dynamic Skyline

Kueri *dynamic skyline* dalam komputasi k-MPP digunakan untuk mencari produk terbaik dari sudut pandang pelanggan [1]. *Dynamic skyline* [2] dari seorang pelanggan $c_1 \in C$, dinotasikan dengan $DSL(c_1)$, berisi semua produk $p_1 \in P$ yang tidak didominasi oleh produk lain $p_2 \in P$ berdasarkan preferensi pelanggan c_1, $p_2 \nprec_{c_1} p_1$.

Dynamic skyline dapat dihitung menggunakan algoritme komputasi *skyline* tradisional [6] dengan cara mentransformasikan semua titik $p \in P$ ke ruang data baru dengan menganggap titik kueri c sebagai titik asal dan jarak abolut titik p ke c digunakan sebagai fungsi pemetaan yang didefinisikan sebagai $f^i(p^i) = |c^i - p^i|$.

E. Reverse Skyline

Kueri *reverse skyline* dalam komputasi k-MPP digunakan untuk mencari pelanggan potensial dari sudut pandang produsen [1]. *Reverse skyline* [3] dari sebuah produk $p_1 \in P$, dinotasikan dengan $RSL(p_1)$, berisi semua pelanggan $c \in C$ yang memiliki p_1 pada hasil *dynamic skyline*-nya.

Ada beberapa tahapan yang harus dilakukan dalam komputasi *reverse skyline* [1], yaitu (1) menentukan *orthant*, dinotasikan dengan O, sejumlah 2^d pada data $d-$dimensi; (2) menghitung semua *midpoint* atau titik tengah antara produk kueri dan setiap produk $p \in P$ menggunakan persamaan $m_2^i = \frac{(p_1^i + p_2^i)}{2}$; (3) menentukan *midpoint skyline* $MSL(o)$ (juga dikenal sebagai *mid-skyline* [8]) pada setiap *orthant*; (4) menentukan *reverse skyline* dengan mencari semua pelanggan $c \in C$ yang tidak didominasi oleh *midpoint skyline* $m \in M$ berdasarkan produk p_1, $c \nprec_{p_1} m$.

F. k-Most Promising Products (k-MPP)

Islam et al. memodelkan kueri *k-Most Promising Products* (k-MPP) dan merancang algoritme pemrosesan untuk memproses kueri tersebut [1]. Terdapat empat langkah pemrosesan kueri k-MPP, yaitu: (1) mencari *reverse skyline* masing-masing produk $p \in P$; (2) mencari *dynamic skyline* masing-masing pelanggan $c \in RSL(p)$; (3) menghitung kontribusi pasar masing-masing produk dengan cara mengakumulasikan probabilitas produk dipilih oleh pelanggan, sebagaimana persamaan $E(C, p|P) = \sum_{\forall c \in RSL(p)} Pr(c, p|P)$; (4) memilih k-produk dengan kontribusi pasar terbesar.

III. Metode

Bagian ini memaparkan pendekatan yang digunakan untuk menjawab kueri k-MPP berbasis interval waktu, meliputi struktur data dan algoritma pemrosesan yang digunakan.

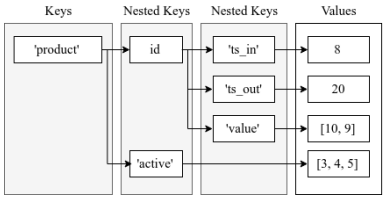
Tabel II: Daftar notasi

Simbol	Deskripsi
P	Himpunan data produk
C	Himpunan data preferensi pelanggan
p	Sebuah data produk dalam P
c	Sebuah data preferensi pelanggan dalam C
D	$P \cup C$
ob	Sebuah objek data pada D
$ob_1 \prec ob_2$	Objek data ob_1 mendominasi ob_2
$ob_1 \prec_{ob_3} ob_2$	Objek data ob_1 mendominasi ob_2 berdasarkan ob_3
d	Jumlah dimensi pada D
i	Dimensi ke-1, ..., d
$[t_i : t_e]$	Interval waktu
E	Himpunan *event*
e	Sebuah *event* dalam E, $e \in E$
p_i	Data produk masuk
p_o	Data produk keluar
c_i	Data pelanggan masuk
c_o	Data pelanggan keluar
PA	Himpunan data produk yang sedang aktif
CA	Himpunan data pelanggan yang sedang aktif
$diff$	Selisih nilai
O	*Orthant*
m	*Midpoint* antar produk
$DSL(c)$	*Dynamic skyline* dari pelanggan c
$RSL(p)$	*Reverse skyline* dari produk p
$MSL(o)$	*Midpoint skyline* dari orthant o
$Pr_t(c, p\|PA)$	Probabilitas produk $p \in PA$ dibeli oleh pelanggan $c \in CA$ pada waktu t
$E_{[t_i:t_e]}(CA, p\|PA)$	Kontribusi pasar p dalam interval waktu $[t_i : t_e]$
k	Jumlah data
$k - MPPTI$	*k-Most Promising Products in Time Intervals*
PB	*Pandora Box*

A. Struktur Data

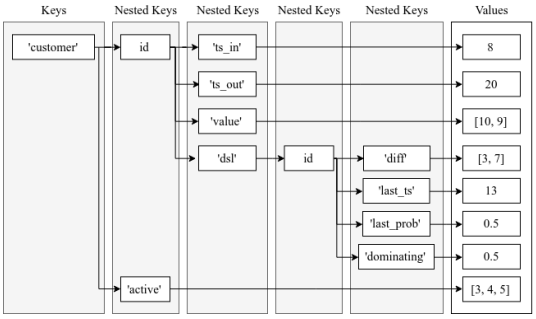
Ada tiga struktur data utama yang digunakan dalam komputasi k-MPPTI, yaitu *Data Storage*, *Event Queue*, dan *Pandora Box*.

1) ***Data Storage****:* Sebuah struktur data *nested dictionary* yang digunakan untuk menyimpan data produk (Gambar 1) dan pelanggan (Gambar 2). Struktur data *dictionary* lebih efisien untuk pencarian data karena menggunakan konsep *key-value pairs* dibandingkan dengan *list* atau *array* yang menggunakan indeks untuk mengakses nilai suatu data.



Gambar 1: Struktur data *dictionary* produk

2) ***Event Queue****:* Sebuah struktur data *queue* dengan prinsip FIFO (*First In First Out*) yang berfungsi untuk menyimpan



Gambar 2: Struktur data *dictionary* pelanggan

Tabel III: Deskripsi *key* dalam *Data Storage*

Key	**Deskripsi**
$'product'$	Menyimpan data produk
$'customer'$	Menyimpan data pelanggan
id	ID data produk atau pelanggan dijadikan sebagai *key*
$'active'$	Menyimpan ID data produk atau pelanggan yang sedang aktif dalam bentuk *array*
$'ts\_in'$	Menyimpan *timestamp* atau waktu masuk
$'ts\_out'$	Menyimpan *timestamp* atau waktu keluar
$'value'$	Menyimpan nilai data produk atau pelanggan pada semua dimensi dalam bentuk *array*
$'dsl'$	Menyimpan hasil *dynamic skyline* dalam bentuk *dictionary* dengan id produk sebagai *key*
$'diff'$	Menyimpan selisih antara nilai data produk dan pelanggan pada masing-masing dimensi
$'last\_ts'$	Menyimpan *timestamp* terakhir saat diperbarui ke *Pandora Box*
$'last\_prob'$	Menyimpan probabilitas terakhir saat diperbarui ke *Pandora Box*
$'dominating'$	Menyimpan ID produk lain yang pernah didominasi

semua titik terjadinya perubahan di dalam himpunan data, yaitu jika ada data yang masuk atau keluar. Titik-titik ini disebut dengan *event*. Ada empat jenis *event* yang terjadi: (1) *Product Insertion* (data produk masuk), (2) *Product Deletion* (data produk keluar), (3) *Customer Insertion* (data pelanggan masuk), dan (4) *Customer Deletion* (data pelanggan keluar). Masing-masing *event* memiliki empat jenis informasi yang disimpan, yaitu *timestamp*, *role* (produk atau pelanggan), ID data, dan aksi (masuk atau keluar).

3) ***Pandora Box****:* Sebuah struktur data *array* dua dimensi, terdiri dari sumbu x (*time series*) dan sumbu y (produk), yang digunakan untuk menyimpan skor kontribusi pasar setiap produk pada setiap waktu. Menggunakan contoh *dataset* pada Tabel I, maka model *Pandora Box* yang terbentuk adalah seperti pada Gambar 3.

B. Algoritme Utama

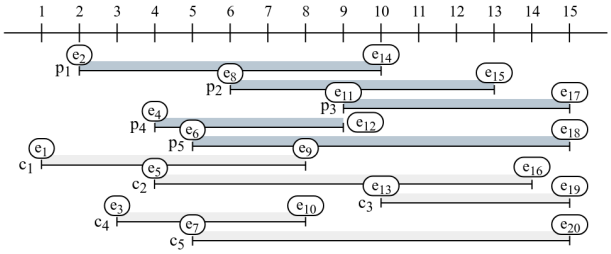
Algoritme k-MPPTI terdiri dari dua tahap pemrosesan, yaitu *data precomputing* dan *query processing*.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p_1	0	1	2	1	0.67	0	0	0	0	0	0	0	0	0	0
p_2	0	0	0	0	0	1.33	1.33	1.33	0.5	1	1	1	1	0	0
p_3	0	0	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	1	0.5
p_4	0	0	0	2	1.67	1.33	1.33	1.33	0.5	0	0	0	0	0	0
p_5	0	0	0	0	1.67	1.33	1.33	1.33	1	1.5	1.5	1.5	1.5	2	1.5

Gambar 3: Contoh *Pandora Box* dari *dataset* I

1) ***Data Precomputing****:* Tahap pertama pemrosesan yang dapat menunjang performa algoritme *query processing* supaya dapat bekerja lebih efektif dan efisien, yaitu dengan menghitung kontribusi pasar masing-masing produk berdasarkan preferensi pelanggan dan mengakumulasinya dalam *Pandora Box*. Diawali dengan mencatat semua *event* ke dalam *Event Queue*, kemudian memproses setiap *event* secara berurutan menggunakan algoritme pemrosesan tertentu berdasarkan jenis *event*-nya, dan diakhiri dengan mengekspor *Pandora Box* yang nantinya akan digunakan sebagai masukan pada tahap *query processing*.

Menggunakan *dataset* pada Tabel I, data multidimensi dengan serial waktu diilustrasikan sebagai lini masa sebagaimana pada Gambar 4. Lini masa atau alur waktu adalah suatu representasi kronologis urutan peristiwa atau kejadian (*event*) yang diwakili oleh titik-titik yang dinotasikan dengan $e \in E$.



Gambar 4: *Event* dalam lini masa data produk dan pelanggan

a) ***Product Insertion****:* Proses yang dijalankan ketika ada data produk yang masuk, dinotasikan dengan p_i. Langkah-langkah pemrosesan *product insertion* dijelaskan sebagai berikut: (1) menambahkan produk p_i ke dalam daftar produk aktif PA, (2) menghitung $RSL(p_i)$, (3) menghitung $DSL(c)$ untuk setiap $c \in RSL(p_i)$ dan menghitung probabilitas masing-masing produk $p \in DSL(c)$, dan (4) memperbarui *Pandora Box*. Proses *product insertion* ditunjukkan pada *pseudocode* baris 13-19 pada Algoritme 1.

b) ***Product Deletion****:* Proses yang dijalankan ketika ada data produk yang keluar, dinotasikan dengan p_o. Langkah-langkah pemrosesan *product deletion* dijelaskan sebagai berikut: (1) memperbarui *Pandora Box* untuk mengisi indeks PB sebelumnya jika ada yang kosong, (2) menghitung $RSL(p_o)$, (3) menghitung $DSL(c)$ untuk setiap $c \in RSL(p_o)$ yang dimaksudkan untuk mencari produk-produk yang pernah didominasi (*child*) dan menghitung probabilitas masing-masing produk $p \in DSL(c)$, dan (4) menghapus produk p_o dari daftar produk aktif PA. Proses *product deletion* ditunjukkan pada *pseudocode* baris 20-26 pada Algoritme 1.

Algorithm 1 Precomputing

```
    Input  : product dataset P, customer dataset C
    Output : pandora box PB
 1: EQ ← init event queue
 2: D ← data indexing P, C
 3: for all d ∈ D do
 4:     EQ ← enqueue d
 5: PA ← ∅ of active products
 6: CA ← ∅ of active customers
 7: PB ← init pandora box
 8: RSL ← init reverse skyline
 9: DSL ← init dynamic skyline
10: while EQ is not empty do
11:     e ← dequeue EQ
12:     if e role is product then
13:         if e action is insertion then
14:             PA ← append p ∈ e
15:             RSL(p) ← compute reverse skyline
16:             for all c ∈ RSL(p) do
17:                 DSL(c) ← compute dynamic skyline
18:             for all c ∈ CA do
19:                 PB ← update pandora box DSL(c)
20:         else if e act is deletion then
21:             for all c ∈ CA do
22:                 PB ← update pandora box DSL(c)
23:             RSL(p) ← compute reverse skyline
24:             for all c ∈ RSL(p) do
25:                 DSL(c) ← find active child and
                               compute new dynamic skyline
26:             PA ← remove p ∈ e
27:     else if e role is customer then
28:         if e act is insertion then
29:             CA ← append c ∈ e
30:             DSL(c) ← compute initial dynamic skyline
31:             PB ← update pandora box DSL(c)
32:         else if e act is deletion then
33:             PB ← update pandora box DSL(c)
34:             CA ← remove c ∈ e
35: export PB
```

c) ***Customer Insertion****:* Proses yang dijalankan ketika ada data pelanggan yang masuk, dinotasikan dengan c_i. Langkah-langkah pemrosesan *customer insertion* dijelaskan sebagai berikut: (1) menambahkan pelanggan c_i ke dalam daftar pelanggan aktif CA, (2) menghitung *initial* $DSL(c_i)$ untuk mendapatkan hasil *dynamic skyline* awal dan menghitung probabilitas masing-masing produk $p \in DSL(c)$, dan (4) memperbarui *Pandora Box*. Proses *customer insertion* ditunjukkan pada *pseudocode* baris 28-31 pada Algoritme 1.

d) ***Customer Deletion****:* Proses yang dijalankan ketika ada data pelanggan yang keluar, dinotasikan dengan c_o. Langkah-langkah pemrosesan *customer deletion* dijelaskan sebagai berikut: (1) memperbarui *Pandora Box* untuk mengisi indeks PB sebelumnya jika ada yang kosong dan (2) menghapus pelanggan c dari daftar pelanggan aktif CA. Proses *customer deletion* ditunjukkan pada *pseudocode* baris 32-34 pada Algoritme 1.

e) ***Kueri Dynamic Skyline****:* Kueri yang digunakan untuk mencari produk terbaik dari sudut pandang pelanggan [1]. Langkah-langkah komputasi $DSL(c)$ secara umum adalah (1) menghitung selisih absolut dari nilai setiap dimensi antara

produk dan pelanggan, dinotasikan dengan $diff^i = |p^i - c_1^i|$; (2) mengecek dominansi dinamis antar produk dengan membandingkan selisih absolut-nya sebagaimana yang ditunjukkan pada Algoritme 2. Pengecekan dominansi dinamis dilakukan secara iteratif hingga dipastikan suatu p_1 tidak didominasi oleh p_2 lain sama sekali. Jika p_1 tidak pernah didominasi, maka p_1 menjadi hasil $DSL(c)$.

Algorithm 2 Check Domination

Input : value of subject (ob_1), value of target (ob_2), value of query point (ob_3), dimension of data (d)
Output : $ob_1 \prec_{ob_3} ob_2$ is true/false

procedure ISDOMINATING(ob_1, ob_2, ob_3)
  $dominating \leftarrow 0$
  $dominated \leftarrow 0$
  for each $i \in d$ **do**
    $diff_1^i \leftarrow |ob_1^i - ob_3^i|$
    $diff_2^i \leftarrow |ob_2^i - ob_3^i|$
    if $diff_1^i = diff_2^i$ **then**
      continue
    else if $diff_1^i < diff_2^i$ **then**
      $domininating \leftarrow dominating + 1$
    else if $diff_1^i > diff_2^i$ **then**
      $dominated \leftarrow dominated + 1$
  if $dominated = 0$ and $dominating \geq 1$ **then**
    return True
  else
    return False

Algoritme komputasi DSL dalam k-MPPTI dibagi menjadi 3 jenis, yaitu: (1) *InitDSL* yang dijalankan jika ada data pelanggan yang masuk (*customer insertion*) sebagaimana *pseudocode* baris 1-13 pada Algoritme 3, (2) *ComputeDSL* yang dijalankan jika ada data produk yang masuk (*product insertion*) sebagaimana *pseudocode* baris 14-28 pada Algoritme 3, dan (3) *ProductDeletion* yang dijalankan jika ada data produk yang keluar (*product deletion*) sebagaimana *pseudocode* baris 29-32 pada Algoritme 3.

f) ***Kueri Reverse Skyline****:* Kueri yang digunakan untuk mencari pelanggan potensial dari sudut pandang produsen [1]. Langkah-langkah komputasi $RSL(p)$ adalah (1) menentukan *orthant* O sejumlah 2^d pada data $d-$dimensi yang ditunjukkan pada *pseudocode* baris 5-8 pada Algoritme 4, (2) menghitung semua *midpoint* atau titik tengah antara produk kueri dan setiap produk $p \in P$ yang ditunjukkan pada *pseudocode* baris 9-12 pada Algoritme 4, (3) menentukan *midpoint skyline* $MSL(o)$ pada setiap *orthant* yang ditunjukkan pada *pseudocode* baris 13-26 pada Algoritme 4, (4) menentukan *reverse skyline* dengan mencari semua pelanggan $c \in C$ yang tidak didominasi oleh *midpoint skyline* $m \in MSL(o)$ berdasarkan produk kueri yang ditunjukkan pada *pseudocode* baris 27-34 pada Algoritme 4.

g) ***Perhitungan Probabilitas****:* Probabilitas masing-masing produk $p \in PA$ dipilih oleh pelanggan $c \in CA$ dihitung menggunakan persamaan berikut:

$$Pr_t(c, p|PA) = \begin{cases} \frac{1}{|DSL(c)|} & \text{if } p \in DSL(c) \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 3 Dynamic Skyline Computation

Input : customer c, active products PA, product in/out p, timestamp ts
Output : $DSL(c)$

procedure INITDSL(c, PA) *// if customer insertion*
  $CAND \leftarrow PA$
  sort $CAND$
  for $i \leftarrow 0$ to length of $CAND$ **do**
    for $j \leftarrow i+1$ to length of $CAND$ **do**
      if $p_i \prec_c p_j$ **then**
        add p_j to the child list of p_i
        $CAND \leftarrow$ remove p_j
      else if $p_j \prec_c p_i$ **then**
        add p_i to the child list of p_j
        $CAND \leftarrow$ remove p_i
        break
    return $CAND$ as $DSL(c)$
procedure COMPUTEDSL(c, p) *// if product insertion*
  $CAND \leftarrow p$, current $DSL(c)$
  sort $CAND$
  $x \leftarrow$ get index of p in $CAND$
  for $i \leftarrow 0$ to length of $CAND$ **do**
    if $i < x$ **then**
      if $p_i \prec_c p_x$ **then**
        add p_x to the child list of p_i
        $CAND \leftarrow$ remove p_x
        break
    else if $i > x$ **then**
      if $p_x \prec_c p_i$ **then**
        add p_i to the child list of p_x
        $CAND \leftarrow$ remove p_i
  return $CAND$ as $DSL(c)$
procedure PRODUCTDELETION(c, p) *// if product deletion*
  $ac \leftarrow$ find active childs of p
  $DSL(c) \leftarrow$ call $computeDSL(c, ac)$
  return $DSL(c)$

h) ***Perhitungan Kontribusi Pasar****:* Kontribusi pasar produk p, dinotasikan dengan $E(C, p|P)$, diperoleh dengan mengakumulasikan probabilitas produk dari setiap pelanggan $c \in RSL(p)$ sebagaimana persamaan $E_t(CA, p|PA) = \sum_{\forall c \in RSL(p)} Pr_t(c, p|P)$. Skor kontribusi pasar disimpan dalam *Pandora Box* sebagaimana ditunjukkan oleh *pseudocode* baris 4-8 pada Algoritme 5. Perhitungan probabilitas dan kontribusi pasar dilakukan setiap akhir pemrosesan *event*.

2) ***Query Processing****:* Tahap kedua pemrosesan yang bertujuan untuk memproses kueri k-MPPTI, dinotasikan sebagai $k - MPPTI(k, [t_i : t_e])$, dengan memilih *subset* k produk P' dari P yang memiliki total kontribusi pasar lebih besar dibandingkan dengan *subset* k produk P'' dari P yang lain dalam interval waktu pencarian. Perhitungan total kontribusi pasar dinotasikan dengan persamaan $E_{[t_i:t_e]}(CA, p|PA) = \sum_{t=t_i}^{t_e} \sum_{\forall c \in RSL(p)} E_t(CA, p|PA)$. Kontribusi pasar diambil dari *Pandora Box* sebagai hasil dari *data precomputing*.

Langkah-langkah pemrosesan dalam *query processing* antara lain: (1) menghitung total kontribusi pasar setiap produk $p \in P$ dalam interval waktu pencarian, (2) mengurutkan produk dengan total skor kontribusi pasar terbesar, dan (3) mengembalikan k-produk teratas sebagai hasil dari kueri pencarian sebagaimana yang ditunjukkan oleh *pseudocode* pada

Algorithm 4 Reverse Skyline Computation

Input : product as query point p_q, active products PA, active customers CA, dimension of data d
Output : $RSL(p)$

procedure COMPUTERSL(p_q)
  call $DefineOrthant(d)$
  call $FindMSL(p_q, PA)$
  return $FindRSL(CA)$
procedure DEFINEORTHANT(d)
  for $i \leftarrow 0$ to 2^d **do**
    $id \leftarrow$ binary of i
    $o_{id} \leftarrow \emptyset$
procedure CALCMIDPOINT(p_q, p)
  for each $i \in d$ **do**
    $m \leftarrow \frac{(p_q^i + p^i)}{2}$
  return m
procedure FINDMSL(p_q, PA)
  for all $p \in PA$ **do**
    if $p \neq p_q$ **then**
      $m \leftarrow CalcMidpoint(p_q, p)$
      $id \leftarrow GetOrthantId(p)$
      if o_{id} is empty **then** $o_{id} \leftarrow m$
      else
        for each $mc \in MSL(o_{id})$ **do**
          if $m \prec_{p_q} mc$ **then**
            $MSL(o_{id}) \leftarrow$ delete mc
          else if $mc \prec_{p_q} m$ **then**
            break
        if $\forall mc \in MSL(o_{id}) \not\prec_{p_q} m$ **then**
          $MSL(o) \leftarrow$ insert m
procedure FINDRSL(p_q, CA)
  for $c \in CA$ **do**
    $id \leftarrow GetOrthantId(c)$
    if o_{id} is empty **then** $RSL(p) \leftarrow$ insert c
    else
      if $\forall m \in MSL(o_{id}) \not\prec_{p_q} c$ **then**
        $RSL(p) \leftarrow$ insert c
  return $RSL(p)$
procedure GETORTHANTID(D)
  for each $i \in d$ **do**
    if $D^i \leq p_q^i$ **then** $id \leftarrow$ append 0
    else $id \leftarrow$ append 1
  return id

Algorithm 5 Pandora Box

Input : $DSL(c)$, timestamp ts, number of products k, time interval (time init t_i, time end t_e)
Output : filled pandora box PB, total market contribution MC_p

// Data Precomputing
procedure CALCPROBABILITY($DSL(c)$) *// calculate probability*
  for all $p \in DSL(c)$ **do**
    $pr \leftarrow \frac{1}{DSL(c)}$
procedure UPDATEPB($DSL(c)$)
  for all $p \in DSL(c)$ **do**
    if $ts > lastts$ **then**
      $UpdateScore(p, ts, pr, lastts, lastpr)$
    $PB(p, ts) \leftarrow PB(p, ts) + pr$
procedure UPDATESCORE(p, ts, pr, $lastts$, $lastpr$)
  for $i \leftarrow lastts + 1$ to ts **do**
    $PB(p, i) \leftarrow PB(p, i) + lastpr$

// Query Processing
procedure GETSCORE(p, t_i, t_e)
  $MC \leftarrow 0$
  for $i \leftarrow t_i$ to $t_e + 1$ **do**
    $MC \leftarrow MC + PB(p, i)$
  return MC

Algorithm 6 Query Processing

Input : Pandora Box PB, number of products k, time interval (time init t_i, time end t_e)
Output : k products

$Q \leftarrow \emptyset$
for all $p \in P$ **do**
  $MC_p \leftarrow$ call $GetScore(p, t_i, t_e)$
sort MC in ascending order based on market contribution score
$Q \leftarrow$ get top-k MC

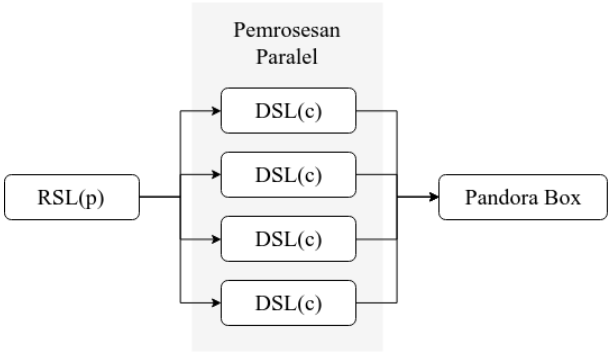
Algoritme 6.

C. Algoritme Tandingan

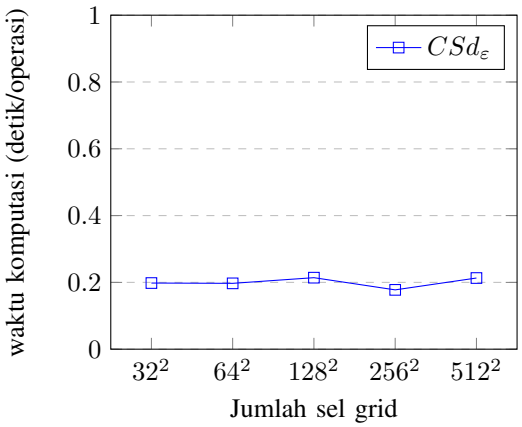
Algoritme tandingan dibuat untuk membandingkan performa antar algoritme. Ada dua jenis algoritme tandingan yang dibuat, yaitu k-MPPTI NoRSL dan k-MPPTI Paralel.

1) ***k-MPPTI NoRSL****:* Algoritme k-MPPTI yang tidak melalui proses komputasi *reverse skyline*.

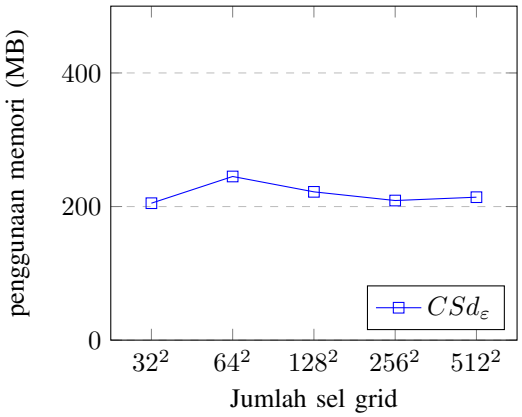
2) ***k-MPPTI Paralel****:* Algoritme k-MPPTI yang mengimplementasikan teknik pemrosesan paralel, yaitu suatu bentuk komputasi dua atau lebih tugas yang dilakukan secara bersamaan dan beroperasi dengan prinsip bahwa masalah besar seringkali dapat dibagi dan dipecah menjadi masalah yang lebih kecil, kemudian dipecahkan secara bersamaan (paralel) [9].



Gambar 5: Pemrosesan paralel



Gambar 6: Pengaruh jumlah sel terhadap waktu komputasi tiap operasi dalam satuan detik



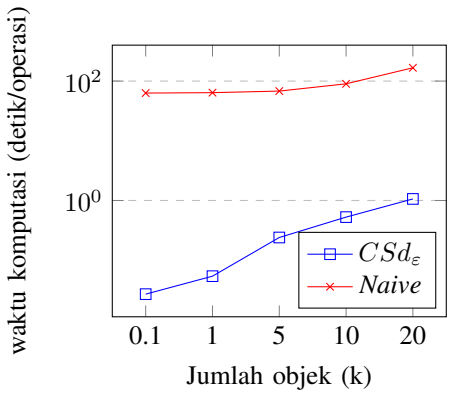
Gambar 7: Pengaruh jumlah sel grid terhadap penggunaan memori dalam satuan megabita

IV. Uji Coba

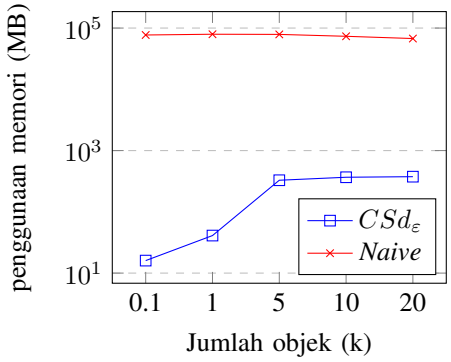
Uji coba dilakukan pada jaringan jalan raya California [**?**] dengan *node* sebanyak 8716 dan *edge* sebanyak 9077. Algoritma diimplementasikan menggunakan bahasa Scala dengan *memory heap* sejumlah 4 GB. Pengujian dilakukan pada komputer dengan *Processor* Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz x 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 IV: 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



Gambar 8: Pengaruh jumlah objek terhadap waktu komputasi tiap operasi dalam satuan detik

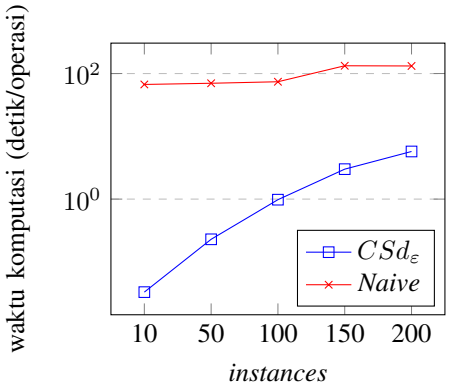


Gambar 9: Pengaruh jumlah objek terhadap penggunaan memori dalam satuan megabita

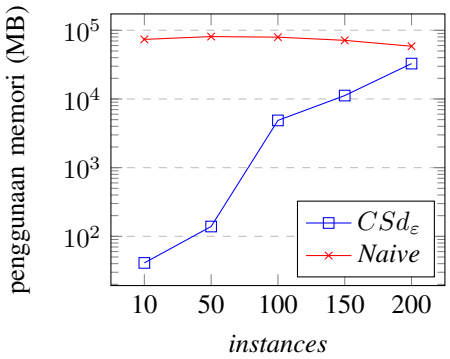
Jumlah sel tidak banyak mempengaruhi penggunaan memori dan waktu komputasi. Hal ini dikarenakan adanya *trade-off* antara proses memuat data dengan komputasi. Grid indeks yang memiliki sel sedikit menjadikan data yang dimuat lebih banyak sehingga menjadikan data yang diproses labih banyak. Tetapi di sisi lain, sistem tidak banyak mencari data secara berulang-ulang karena setiap sel sudah mengaver area yang besar. Sedangkan grid indeks yang memiliki sel yang banyak menjadikan proses komputasi lebih efisien karena melibatkan data yang lebih sedikit. Tetapi di sisi lain, sistem harus melakukan pencarian data berulang-ulang karena sedikitnya data yang didapat pada setiap sel.

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

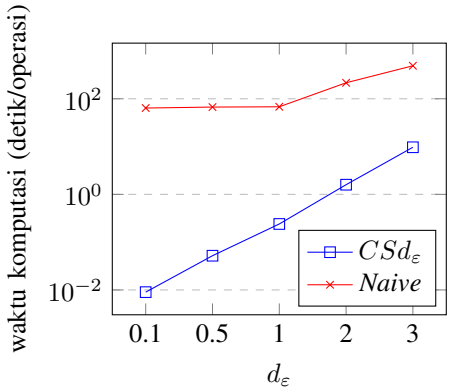


Gambar 11: Pengaruh jumlah *instance* terhadap penggunaan memori dalam satuan megabita

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.

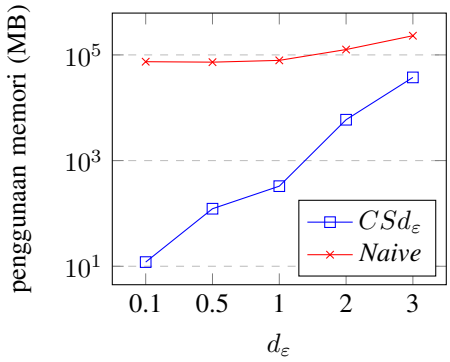
Pada metode *naive*, objek perubahan waktu komputasi terlihat ketika jumlah *instance* diatas 100. Hal ini dikarenakan waktu komputasi lebih banyak digunakan untuk penghitungan jarak terpendek dari setiap *node* ke objek, sehingga jumlah *instance* yang sedikit tidak berpengaruh banyak terhadap waktu 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 12: 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 13: 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].