Analisis dan Implementasi Graph Indexing Pada Graph Database Menggunakan Algoritma GIndex

Analysis and Implementation of Graph Indexing for Graph Database Using GIndex Algorithm

Hadyan Arif[1], Kemas Rahmat Saleh[2], Dr. Adiwijaya[3]
[1.2.3] Fakultas Informatika, Telkom Engineering School, Telkom University
Jalan Telekomunikasi No.1, Dayeuh Kolot, Bandung 40257 hadyanarif93@gmail.com[1] ,
bagindok3m45@gmail.com[2] , kang.adiwijaya@gmail.com[3]

Abstrak

Penggunaan *graph* dalam memodelkan suatu struktur yang rumit saat ini berkembang secara pesat terutama dalam memodelkan struktur seperti susunan melekul, jaringan protein, dan jaringan sosial. Penggunaan *graph database* untuk menangani tipe data *graph* yang memiliki relasi yang kompleks dinilai lebih efektif daripada menggunakan *relational database*.

Dalam mempercepat pemrosesan *query* pada *graph database* dibutuhkan suatu metode yang dapat disebut *graph indexing* agar lebih cepat dan efisien. GIndex merupakan salah satu metode *graph indexing* yang mendukung pemrosesan *query* bertipe *subgraph query*. Pada metode GIndex menerapkan beberapa teknik seperti *size-increasing support constraint* untuk membangun *feature set database* dan pemilihan *discriminative fragments* dalam membangun *index*. Kemudian membandingkan data pada *index* dengan *feature set query* untuk mendapatkan *candidate set* yang nantinya akan dilakukan *subgraph matching* menggunakan algoritma Ullman untuk mendapatkan *answer set*.

Pada penelitian ini data yang akan dijadikan sebagai *dataset* merupakan susunan molekul. Berdasarkan pengerjaan tugas akhir yang telah dilakukan didapatkan hasil bahwa pada pengimplementasian algoritma GIndex, jika menggunakan nilai *maximal frequent fragment* yang cukup besar maka akan memakan waktu yang lebih lama dan memungkinkan jumlah *candidate set* yang didapatkan akan lebih sedikit, berlawanan dengan penggunaan nilai *minimal discriminative fragment*. Banyaknya jumlah *candidate set* yang didapatkan akan berpengaruh pada waktu *subgraph matching* yang dibutuhkan.

Kata kunci: *graph, graph database*, GIndex, *subgraph query, size-increasing support constraint, discriminative fragments, index, subgraph matching*

1. PENDAHULUAN

Penggunaan *graph* dalam memodelkan suatu struktur yang rumit saat ini berkembang secara pesat [16]. Contohnya dalam memodelkan susunan melekul, jaringan protein, dan jaringan social [11]. Secara konsep hampir semua data yang ada dapat direpresentasikan dengan menggunakan *graph* [15].

Penggunaan *graph database* saat ini pun mulai berkembang seiring dengan berkembangnya penggunaan *graph* dalam memodelkan suatu struktur [11]. *Graph database* dapat secara mudah menangani tipe data yang direpresentasikan dalam bentuk *graph*. Pengimplementasian *graph database* untuk menangani tipe data yang memiliki relasi/keterikatan yang kompleks dinilai lebih efektif daripada menggunakan *relational database* [13]. Hal ini dikarenakan dalam *graph database*, data direpresentasikan sebagai *node* dan *edge* yang saling terhubung satu sama lain, berbeda dengan *relational database* yang menggunakan tabel relasi yang berisi kolom dan baris.

Dalam mempercepat proses pencarian *query* pada *graph database*, dibutuhkan suatu metode yang dapat disebut *graph indexing*. Berdasarkan kinerjanya metode *graph indexing* dibagi menjadi 2 yaitu teknik *graph indexing* berbasis *non-mining* dan teknik *graph indexing* berbasis *mining* [16]. Beberapa metode yang termasuk dalam *non mining-based graph indexing techniques* seperti: GraphGrep [4], GraphRel [10], GString [6], dan Closure-Tree [5]. Dan beberapa metode yang ada pada metode *mining-based graph indexing techniques* seperti: GIndex [15], FGIndex [2], dan TreePi [17].

Dari beberapa metode *graph indexing* yang ada, GIndex merupakan salah satu metode *graph indexing* yang pada kinerjanya menggunakan mekanisme *indexing frequent feature*. Kelebihan mekanisme *indexing* menggunakan *frequent feature* ini adalah struktur keaslian dari *dataset* tetap terjaga [18], sehingga tidak dimungkinkan adanya kehilangan struktur informasi dari *dataset*. Selain menggunakan mekanisme *indexing frequent feature*, GIndex menerapkan adanya pemilihan struktur *discriminative* untuk menekan jumlah struktur yang harus disimpan kedalam

index agar lebih efisien [18]. Dalam menerapkan mekanisme *indexing frequent feature* dan pemilihan struktur *discriminative,* pada GIndex digunakan beberapa teknik seperti teknik *size-increasing support constraint* untuk membangkitkan *feature set* dari *graph database* dan pemilihan *discriminative fragments* dari beberapa *fragment* pada *feature set* untuk disimpan kedalam *index*.

Setelah *index* dibangun dilakukan pencocokan antara *feature set query* dengan fragment yang terdapat pada *index* untuk mendapatkan *candidate set*. Kemudian dilakukan verifikasi dengan metode *subgraph matching* [14] dengan algoritma Ullman antara *candidate set* dengan *query* untuk mendapatkan *answer set* yang dicari. Pada pengerjaan tugas akhir ini data yang akan dijadikan sebagai *dataset* merupakan susunan molekul. Hal tersebut dikarenakan dalam pengimplementasian metode GIndex dianggap lebih cocok dan efektif dalam menangani tipe data *graph* yang memiliki relasi yang kompleks serta tidak berarah seperti susunan molekul.

2. LANDASAN TEORI

2.1 GIndex

GIndex merupakan salah satu metode *graph indexing* yang termasuk dalam kategori teknik *graph indexing* berbasis *mining* [16]. Pada GIndex menerapkan beberapa teknik seperti *size-increasing support constraint* dan pemilihan *discriminative fragment*. Pada umumnya GIndex terbagi menjadi 3 fase yaitu: *index construction*, *search*, dan *verification*.

2.1.1 Index Construction

Pada saat pembangunan index, pada algoritma GIndex menerapkan 3 tahap yaitu : *size increasing support constraint, discriminative fragment selection, save to index.*

2.1.1.1 Size-increasing Support Constraint

Size-increasing support constraint merupakan salah satu teknik mengeksplorasi *graph* menjadi beberapa kumpulan *frequent fragment* untuk dimasukkan ke dalam *index*. Dapat dikatakan bahwa dalam hal ini *fragment* merupakan *subgraph* dari *graph* tertentu. *Size-increasing support constraint* membangkitkan beberapa *frequent fragment* sebesar *maximal frequent fragment size* hingga *minSup* = 1 bertujuan untuk memastikan kelengkapan ketika dilakukan *indexing,* sehingga ketika *indexing* tidak akan melewatkan satu *fragment* yang berguna [15].

2.1.1.2 *Discriminative Fragment Selection*

Dengan menggunakan teknik s*ize-increasing support constraint*, tentunya akan menghasilkan banyak *frequent fragment* kecil yang memungkinkan dapat memperlambat kinerja ketika *indexing* jika seluruh hasil kumpulan dari *frequent fragment* yang ada harus dimasukkan ke dalam *index* seluruhnya. Dikarenakan hal tersebut, dalam implementasinya metode GIndex juga menerapkan adanya *discriminative fragment selection* untuk memilih *fragment* yang berguna saja untuk dimasukkan kedalam *index* [15]. *Discriminative fragment* yang nantinya akan dipilih untuk disimpan ke dalam *index* ditentukan oleh nilai *discriminative ratio* yang dimiliki oleh masing-masing *frequent fragment.*

2.1.1.3 *Save to Index*

Setelah *discriminative fragment* telah terpilih dari *feature* set, *discriminative fragment* tersebut disimpan ke dalam *index* beserta dengan *value set* yang dimiliki dari *discriminative fragment* tersebut.

2.1.2 Search

Pada tahap sebelum pencarian, akan dibangkitkan *feature set query* dari *query* yang dimasukkan. Kemudian proses pencarian dilakukan dengan cara melihat atau memeriksa apakah terdapat *feature set query* yang terkandung pada *index* atau sama dengan *discriminative fragment* yang terdapat pada *index*. Jika pada saat proses pencarian *feature set query* tidak cocok dengan semua *discriminative fragment* pada *index*, maka pencarian tidak perlu dilanjutkan sampai ke *supergraph*-nya [15]. Setelah didapatkan *discriminative fragment* yang sama dengan beberapa *feature set query*, dilakukan pemilihan *maximum discriminative fragment* [15] dan juga *intersecting candidate set* dari *value set* yang dimiliki dari setiap *maximum discriminative* yang didapatkan untuk mendapatkan *candidate set* yang lebih optimal.

2.1.3 Verification

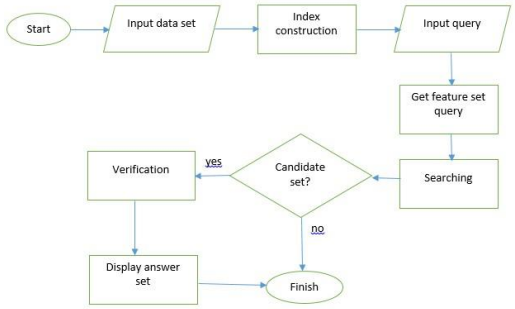
Setelah mendapatkan beberapa *candidate set* [15]. Dilakukan *verification* antara *candidate* set dengan *query*. Cara mudah dalam mencocokkan *candidate set* dengan *query* yaitu dengan menggunakan cara pengujian *subgraph isomorphism* [15] dengan menerapkan *subgraph matching* menggunakan algoritma Ullman.

2.2 Subgraph Matching

Setelah didapatkan *candidate set*, dilakukan *subgraph matching* antara *candidate set* dengan *query* untuk membuktikan apakah *query* tersebut merupakan *subgraph isomorphism* dari *candidate set. Subgraph matching* yang akan dilakukan menerapkan algoritma Ullman.

3. PERANCANGAN SISTEM

Gambaran umum sistem yang akan dibangun diilustrasikan seperti gambar dibawah ini:



Gambar 3-1 Tahapan proses

Berdasarkan Gambar 3-1, secara umum sistem akan menerima dua *data input* dengan format penulisan SMILES. *Data input* tersebut terdiri dari *data input* berupa *dataset* untuk disimpan kedalam *graph database* dan *data input* yang berupa *query graph*. Pada tahap awal proses pencariannya, setiap *dataset* yang telah tersimpan akan dibangkitkan dengan menggunakan teknik *size-increasing support constraint* menjadi beberapa kumpulan *frequent fragment* yang disebut *feature set*. Dari beberapa *fragment* pada *feature set* yang telah dibangkitkan dipilih *discriminative fragment* yang sesuai dengan nilai *minimal discriminative ratio* untuk disimpan kedalam *index*, setelah beberapa *discriminative fragment* terkumpul didalam *index* dilakukan proses *size-increasing support* terhadap *query* untuk mendapatkan *feature set query* yang selanjutnya akan dibandingkan dengan setiap *discriminative fragment* yang telah tersimpan pada *index*. Jika terdapat *discriminative fragment* yang sama dengan salah satu *fragment* dari *feature set query*, *discriminative fragment* tersebut akan disimpan sementara untuk proses pada tahap selanjutnya.

Setelah proses perbandingan antara *discriminative fragment* yang terdapat pada *index* dengan *feature set* dari *query* selesai, dari beberapa *discriminative fragment* yang telah tersimpan sementara akan dilakukan *discriminative fragment selection* untuk mendapatkan *maximum discriminative fragment* dan *intersecting value set* untuk mendapatkan *candidate set.* Setelah *candidate set* ditemukan dilakukan *verification* dengan cara pengujian *subgraph isomorphism* untuk mendapatkan answer set.

4. PENGUJIAN DAN ANALISIS

Berdasarkan tujuan yang ingin dicapai, berikut adalah beberapa pengujian yang akan dilakukan:

4.1 Pengujian Pengaruh Ukuran Nilai *Maximum Fragment* dan *Minimal Discriminative Ratio* Terhadap Waktu Pembentukan *Index*

Pengujian akan dilakukan dengan mengukur waktu yang dibutuhkan untuk setiap ukuran *maximum fragment* yang telah ditentukan (4, 6, dan 8) dan ukuran *minimal discriminative ratio* yang telah ditentukan (2 dan 5) terhadap beberapa jumlah *dataset* yaitu 500, 750, 1000, 1250, dan 1500 dalam proses pembangunan *index construction*.

4.2 Pengujian Pengaruh Ukuran Nilai *Maximum Fragment* dan *Minimal Discriminative Ratio* Terhadap Ukuran *Index Size*

Pengujian akan dilakukan dengan mengukur *size index* (ukuran *index*) yang dihasilkan untuk setiap ukuran *maximum fragment* yang telah ditentukan (4, 6, dan 8) dan ukuran *minimal discriminative ratio* yang telah ditentukan (2 dan 5) terhadap beberapa jumlah *dataset* yaitu 500, 750, 1000, 1250, dan 1500 dalam proses *index construction*.

4.3 Pengujian Pengaruh Ukuran Nilai *Maximum Fragment* dan *Minimal Discriminative Ratio* Terhadap Jumlah *Candidate Set*

Pengujian akan dilakukan dengan menghitung jumlah *candidate set* yang didapatkan dari proses *searching* menggunakan 2 jenis *query* yang akan dibangkitkan untuk mendapatkan *feature set query* dengan nilai *maximum fragment* sesuai dengan *fragment* pada saat *index* dibangun (4, 6, dan 8). 2 *query* tersebut adalah sebagai berikut :

Tabel 4-1 Tabel query

Nomor	*Query*	Siklik	Jumlah *Node*
1	O=C(OCC)CNNCC	Tidak	10
2	O=C(OC)C1=NNC(C)=C1	Ya	10

Pengujian dilakukan untuk setiap *index* yang telah dibangun dari setiap ukuran *maximum fragment* yang telah ditentukan (4,6, dan 8) dan ukuran *minimal discriminative ratio* yang telah ditentukan (2 dan 5) terhadap beberapa jumlah *dataset* yaitu 500, 750, 1000, 1250, dan 1500.

4.4 Pengujian Waktu *Subgraph Matching* dan Penemuan Solusi *Subgraph Matching* pada Algoritma Ullman dengan Beberapa *Candidate Set*

Pengujian yang akan dilakukan pada skenario ini pada dasarnya memiliki konsep yang sama dengan pengujian yang dilakukan pada referensi [3]. Pengujian dilakukan dengan menghitung waktu serta melihat hasil solusi yang dibangkitkan pada saat proses pengujian *subgraph isomorphism* antara *candidate set* tertentu dengan 2 *query* pada Tabel 4-1. *Candidate set* yang digunakan sebagai pengujian merupakan beberapa *candidate set* yang dihasilkan dari proses *searching* antara 2 *query* sebelumnya dengan *index* yang telah dibangun. *Candidate set* yang digunakan untuk pengujian berjumlah 3 *graph* molekul dan akan berbeda untuk setiap *query*. Berikut beberapa *candidate set* yang dipilih untuk pengujian :

Tabel 4-2 Tabel candidate set

Candidate Set Query 1	Jumlah *Node*
O=CNN(C)c1ccccc1	11
O=C1N(CCCCC1)c1ccccc1	14
O=C1OC(=CCN=[N+]=[N-])C(OC)=C1OC	15
Candidate Set Query 2	Jumlah *Node*
O=CNN(C)c1ccccc1	11
O=C(OC)C=1C(=NNC=1C)C	11
BrC=1C(=O)=NN=C(OCC)C=1Br	12

4.5 Analisis Hasil Pengujian Pengaruh Ukuran Nilai *Maximum Fragment* dan *Minimal Discriminative Ratio* Terhadap Waktu Pembentukan *Index*

Sesuai pengujian dengan mengukur waktu yang dibutuhkan didapatkan hasil seperti dibawah :

1. Hasil pengujian dengan ukuran *minimal discriminative ratio* sama dengan 2.

Tabel 4-3 Tabel hasil pengujian skenario 1 dengan minimal discriminative ratio 2

Dataset	*Minimal Discriminative Ratio*	*Max. Fragment* 4	*Max. Fragment* 6	*Max. Fragment* 8
		Time Build Index (detik)	*Time Build Index* (detik)	*Time Build Index* (detik)
500	2	2.900449	41.61541729	2505.918
750	2	3.390424	42.23747179	4383.725
1000	2	4.760179	90.58875468	5695.526
1250	2	6.294568	133.1043441	6760.201
1500	2	7.478914	197.7429111	8846.672

2. Hasil pengujian dengan ukuran *minimal discriminative ratio* sama dengan 5.

Tabel 4-4 Tabel hasil pengujian skenario 1 dengan minimal discriminative ratio 5

Dataset	*Minimal Discriminative Ratio*	*Max. Fragment* 4	*Max. Fragment* 6	*Max. Fragment* 8
		Time Build Index (detik)	*Time Build Index*(detik)	*Time Build Index* (detik)
500	5	2.687137	41.38732111	2505.524
750	5	3.094148	42.01157234	4383.269
1000	5	4.474257	90.13287749	5695.122
1250	5	6.118395	132.823416	6760.053
1500	5	7.231922	197.3349165	8846.653

Berdasarkan data hasil pengujian pada Tabel 4-3 dan Tabel 4-4 dapat disimpulkan bahwa, apabila ukuran nilai *maximum fragment* semakin besar maka hasil dari waktu yang dibutuhkan untuk membangun index akan semakin besar juga. Hal ini dikarenakan jika semakin besar nilai *maximum fragment* maka akan semakin banyak proses yang akan dilakukan untuk membangkitkan beberapa *frequent fragment* dari dataset hingga berukuran sama dengan nilai *maximum fragment*. Sedangkan untuk ukuran nilai *minimum discriminative ratio*, apabila nilai semakin besar maka akan semakin kecil waktu yang dibutuhkan jika dibandingkan dengan memakai nilai *minimum discriminative ratio* yang lebih kecil. Hal ini dikarenakan semakin besar nilai *minimum discriminative ratio* maka akan semakin sedikit *discriminative fragment* yang disimpan kedalam index. Nilai *discriminative ratio* yang dimiliki oleh setiap *discriminative fragment* pada umumnya memiliki nilai yang kecil. Sehingga jika nilai *minimal discriminative ratio lebih besar* maka waktu proses yang dilakukan untuk menyimpan kedalam *index* akan lebih sedikit.

4.6 Analisis Hasil Pengujian Pengaruh Ukuran Nilai *Maximum Fragment* dan *Minimal Discriminative Ratio* Terhadap Ukuran *Index Size*

Sesuai pengujian dengan mengukur *index size* yang didapatkan hasil seperti dibawah :

1. Hasil pengujian dengan ukuran *minimal discriminative ratio* sama dengan 2.

Tabel 4-5 Tabel hasil pengujian skenario 2 dengan minimal discriminative ratio 2

Dataset	*Minimal Discriminative Ratio*	*Max. Fragment* 4	*Max. Fragment* 6	*Max. Fragment* 8
		Index Size	*Index Size*	*Index Size*
500	2	56	76	93
750	2	58	78	97
1000	2	58	78	97
1250	2	60	84	106
1500	2	69	99	125

2. Hasil pengujian dengan ukuran *minimal discriminative ratio* sama dengan 5.

Tabel 4-6 Tabel hasil pengujian skenario 2 dengan minimal discriminative ratio 5

Dataset	*Minimal Discriminative Ratio*	*Max. Fragment* 4	*Max. Fragment* 6	*Max. Fragment* 8
		Index Size	*Index Size*	*Index Size*
500	5	39	52	65
750	5	42	54	69
1000	5	41	53	68
1250	5	46	62	77
1500	5	53	74	93

Berdasarkan data hasil pengujian pada Tabel 4-5 dan Tabel 4-6 dapat disimpulkan bahwa, apabila ukuran nilai *maximum fragment* semakin besar maka hasil dari *index size* yang dihasilkan semakin besar juga. Hal ini dikarenakan jika semakin besar nilai *maximum fragment* maka akan semakin banyak *frequent fragment* yang dibangkitkan dari dataset hingga berukuran sama dengan nilai *maximum fragment*, maka tidak akan menutup kemungkinan bahwa *frequent fragment* yang termasuk *discriminative fragment* akan semakin banyak.

Sedangkan untuk ukuran nilai *minimum discriminative ratio*, apabila nilai semakin besar maka akan semakin kecil *index size* yang dihasilkan jika dibandingkan dengan memakai nilai *minimum discriminative ratio* yang lebih kecil. Hal ini dikarenakan semakin besar nilai *minimum discriminative ratio* maka akan semakin sedikit *discriminative fragment* yang disimpan kedalam index.

4.7 Analisis Hasil Pengujian Pengaruh Ukuran Nilai *Maximum Fragment* dan *Minimal Discriminative Ratio* Terhadap Jumlah *Candidate Set*

Sesuai pengujian dengan menghitung jumlah *candidate set* yang dihasilkan, didapatkan hasil seperti dibawah :

1. Hasil pengujian menggunakan *query* 1.

Tabel 4-7 Tabel hasil pengujian skenario 3 dengan query 1

Dataset	*Candidate Set* (*Max. Fragment* 4 dan *Min. Discriminative ratio* 2)	*Candidate Set* (*Max. Fragment* 6 dan *Min. Discriminative ratio* 2)	*Candidate Set* (*Max. Fragment* 8 dan *Min. Discriminative ratio* 2)
500	4	4	4
750	4	4	4
1000	8	8	8
1250	9	9	9
1500	13	12	9
Dataset	*Candidate Set* (*Max. Fragment* 4 dan *Min. Discriminative ratio* 5)	*Candidate Set* (*Max. Fragment* 6 dan *Min. Discriminative ratio* 5)	*Candidate Set* (*Max. Fragment* 8 dan *Min. Discriminative ratio* 5)
500	6	5	5
750	5	5	5
1000	13	13	11
1250	14	14	12
1500	19	19	12

2. Hasil pengujian menggunakan *query* 2.

Tabel 4-8 Tabel hasil pengujian skenario 3 dengan query 2

Dataset	*Candidate Set* (*Max. Fragment* 4 dan *Min. Discriminative ratio* 2)	*Candidate Set* (*Max. Fragment* 6 dan *Min. Discriminative ratio* 2)	*Candidate Set* (*Max. Fragment* 8 dan *Min. Discriminative ratio* 2)
500	4	3	3
750	4	3	3
1000	8	6	6
1250	9	6	6
1500	12	8	8
Dataset	*Candidate Set* (*Max. Fragment* 4 dan *Min. Discriminative ratio* 5)	*Candidate Set* (*Max. Fragment* 6 dan *Min. Discriminative ratio* 5)	*Candidate Set* (*Max. Fragment* 8 dan *Min. Discriminative ratio* 5)
500	6	5	5
750	5	8	5
1000	13	13	11

1250	14	14	12
1500	19	19	15

Berdasarkan data hasil pengujian pada Tabel 4-7 dan Tabel 4-8 dapat disimpulkan bahwa, apabila ukuran nilai *maximum fragment* semakin besar maka jumlah *candidate set* yang didapatkan akan memiliki kemungkinan lebih sedikit. Hal ini dikarenakan jika semakin besar nilai *maximum fragment* maka akan semakin banyak jumlah *frequent fragment* yang menyebabkan adanya kemungkinan suatu *discriminative fragment* yang memiliki *value set* yang lebih banyak mempunyai *maximum discriminative fragment* yang memiliki *value set* yang lebih sedikit. Sehingga jumlah *candidate set* yang didapatkan akan berkurang.

Sedangkan untuk ukuran nilai *minimum discriminative ratio*, apabila nilai semakin besar maka akan semakin banyak jumlah *candidate set* yang dihasilkan jika dibandingkan dengan memakai nilai *minimum discriminative ratio* yang lebih kecil. Hal ini dikarenakan pada umumnya setiap *frequent fragment* yang memiliki nilai *discriminative ratio* yang besar dimiliki oleh *frequent fragment* yang memiliki jumlah *value set* yang lebih besar.

4.8 Analisis Hasil Pengujian Waktu *Subgraph Matching* dan Penemuan Solusi *Subgraph Matching* pada Algoritma Ullman dengan Beberapa *Candidate Set*

Sesuai pengujian dengan menghitung waktu yang diperlukan dan jumlah penemuan solusi pada saat melakukan *subgraph matching* untuk beberapa *candidate set*, didapatakan hasil seperti dibawah :

1. Hasil pengujian *candidate set.*

Tabel 4-9 Tabel hasil pengujian skenario 4 dengan query 1

Candidate Set	Solusi	Waktu
1	0	0.031438
2	0	0.002157
3	1451519	3931.103732

Tabel 4-10 Tabel hasil pengujian skenario 4 dengan query 2

Candidate Set	Solusi	Waktu
1	0	0.003639
2	20159	32.36308
3	2879	3.981805

Berdasarkan data hasil pengujian pada Tabel 4-9 dan Tabel 4-10 dapat disimpulkan bahwa, jika jumlah penemuan solusi semakin banyak maka akan membutuhkan waktu semakin banyak juga. Penemuan jumlah solusi pada setiap *candidate set* ditentukan oleh *query* dan *candidate set* itu sendiri, hal ini dikarenakan pada saat melakukan proses *subgraph matching* dilakukan pembangunan *adjacency matrix* M yang dimana *matrix* M merupakan *intersect* dari *graph query* dan *graph candidate set* yang nantinya akan ditelusuri berapa jumlah solusi yang dapat dibangkitkan. Hasil pengujian pada referensi [3] dan berdasarkan hasil pengujian pada skenario ini menyebutkan bahwa, penemuan jumlah solusi juga dapat dimungkinkan tergantung dari banyaknya *node* yang dimiliki oleh *query* dan *candidate set*, karena jika ukuran *matrix* M yang dibangun lebih besar maka akan dimungkinkan ditemukannya jumlah solusi yang lebih banyak.

Berikut juga terdapat pengujian terhadap waktu *subgraph matching* yang diperlukan dengan menggunakan *query* 2 dengan *maximum fragment* = 6 dan *minimal discriminative ratio* = 5.

Tabel 4-11 Tabel hasil pengujian waktu subgraph matching

Dataset	Candidate Set	Answer Set	Waktu Subgraph Matching
500	5	3	32.14099
750	5	3	31.44869
1000	11	6	77.96482
1250	14	6	133.9259
1500	19	8	207.9932

Berdasarkan Tabel 4-11 dapat disimpulkan bahwa, jumlah *candidate set* yang dihasilkan juga dapat mempengaruhi waktu dari proses *subgraph matching.* Semakin banyak jumlah *candidate set* maka akan semakin banyak juga waktu yang dibutuhkan.

5. PENUTUP

5.1 Kesimpulan

Berikut beberapa kesimpulan yang dapat diambil dari beberapa pengujian yang telah dilakukan pada bab sebelumnya. Kesimpulan tersebut yaitu :

1. Semakin besar nilai *maximum fragment* yang digunakan dan semakin kecil nilai *discriminative ratio*, maka akan semakin besar waktu yang dibutuhkan untuk membangun Index.
2. Semakin besar nilai *maximum fragment* yang digunakan dan semakin kecil nilai *discriminative ratio*, maka akan semakin besar ukuran *index size* yang didapatkan.
3. Semakin kecil nilai *maximum fragment* yang digunakan dan semakin besar nilai *discriminative ratio*, maka akan memungkinkan semakin banyak jumlah *candidate set* yang didapatkan.
4. Jumlah penemuan solusi dan waktu yang diperlukan dalam proses *subgraph matching* dengan algoritma Ullman bergantung pada *adjacency matrix* M yang dibangun.
5. Semakin besar jumlah node pada *query* dan *candidate set* dapat memungkinkan waktu yang diperlukan untuk *subgraph matching* akan lebih besar.
6. Semakin banyak jumlah *candidate set* maka akan memungkinkan semakin besar waktu yang akan diperlukan.

5.2 Saran

Berikut ini beberapa saran yang dapat dijadikan acuan untuk penelitian kedepannya, khususnya untuk ruang lingkup *graph indexing* :

1. Untuk penelitian kedepannya dapat dilakukan dengan menggunakan tipe data lainnya yang memiliki struktur *graph* berbeda dari struktur *graph* molekul, terutama dengan tipe data *graph* yang memiliki label pada *edge*-nya.
2. Untuk penelitian selanjutnya diharapkan dapat menerapkan data *graph* yang memiliki *cycle*/siklik lebih dari 1.
3. Diharapkan untuk penelitian selanjutnya pada proses pengujian *subgraph isomorphism* dapat dilakukan dengan cara *subgraph matching* yang menerapkan algoritma lain.
4. Untuk penelitian kedepannya dapat dilakukan pengujian dengan menggunakan metode *indexing* yang menggunakan mekanisme *indexing frequent feature* yang lain seperti FGIndex untuk dilakukan perbandingan.

Daftar Pustaka

[1] Cattel, R., 2010. Scalable SQL and NoSQL data stores. In: *ACM SIGMOD Record.* New York: ACM, pp. 12-27.

[2] Cheng, J., Ke, Y., Ng, W. & Lu, A., 2007. *Fg-index: towards verion graph databases.* s.l., ACM, pp. 857-872.

[3] Dongoran, E. S. S., 2015. *Analisis dan Implementasi Graph Indexing Pada Graph Database Menggunakan Algoritma GraphGrep,* Bandung: Universitas Telkom.

[4] Giugno, R. & Shasha, D., 2002. *Graphgrep: A fast and universal method for querying graphs.* s.l., IEEE, pp. 112-115.

[5] He, H. & Singh, A. K., 2006. *Closure-Tree: An Index Structure for Graph Queries.* s.l., IEEE, p. 38.

[6] Jiang, H., Wang, H., Yu, P. & Zhou, S., 2007. *Gstring: A novel approach for efficient search in graph databases.* s.l., IEEE, pp. 566-575.

[7] Konagaya, M., Otachi, Y. & Uehara, R., 2014. *Plynomial-Time Algorithms for Subgraph Isomorphism in Small Graph Classes of Perfect Graphs*,* Ishikawa, Japan: School of Information Science, Japan Advanced Institute of Science and Technology.

[8] L. Ravi, K. S., 2012. Graph Matching Algorithm-Through Isomorphism. *International Journal of Web Technology,* 01(02).

[9] Rouhonen, K., 2008. *Graph theory.* s.l.:Tampere University of Technology.

[10] Sakr, S., 2009. GraphREL: A decomposition-based and selectivity-aware relational framework for processing sub-graph queries. In: *Database Systems for Advanced Applications.* s.l.:Springer Berlin Heidelberg, pp. 123-137.

[11] Sakr, S. & Al-Naymat, G., 2012. An overview of graph indexing and querying techniques.

[12] Singh, H. & Sharma, R., 2012. Role of Adjacency Matrix & Adjacency List in Graph Theory. *International Journal of Computers & Technology,* Volume 3.

[13] Vicknair, C. et al., 2010. *A Comparison of a Graph Database and a Relational.* New York, ACM.

[14] Vogl, W.-D., 2011. *Graph Matching - Algorithms,* Vienna: University Of Technology.

[15] Yan, X., Yu, P. S. & Han, J., 2004. *Graph Indexing: A Frequent Structurebased Approach.* s.l., ACM, pp. 335-346.

[16] Yan, X., Yu, P. S. & Han, J., 2005. Graph Indexing Based on Discriminative Frequent. *ACM Transactions on Database Systems (TODS),* pp. 960-993.

[17] Zhang, S., Hu, M. & Yang, J., 2007. *TreePi: A Novel Graph Indexing Method.* s.l., IEEE, pp. 966-975.

[18] Zhe, F., 2011. *Survey on Graph Query Processing on Graph Database,* s.l.: s.n.