Analisis dan Implementasi Graph Indexing Pada Graph Database Menggunakan Algoritma GraphGrep

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

Emir Septian Sori Dongoran[1], Kemas Rahmat Saleh[2], Alfian Akbar Ghozali[3]

[1.2.3] Fakultas Informatika, Telkom Engineering School, Telkom University

Jalan Telekomunikasi No.1, Dayeuh Kolot, Bandung 40257 emirkipang@gmail.com[1], bagindok3m45@gmail.com[2] alfian@telkomuniversity.ac.id[3]

Abstrak

Graph database adalah basis data yang menggunakan struktur graf untuk merepresentasikan dan mengelola data. Sebagian besar basis data yang digunakan basis data relasional karena penggunaannya yang relatif mudah dan mendukung banyak tipe data. Namun, untuk tipe data tertentu seperti tipe data molekul yang memiliki ciri vertex berlabel dan edge yang tidak berarah, basis data relasional kurang begitu efektif digunakan karena tipe data tersebut memiliki keterkaitan secara independen. Untuk menangani hal tersebut, basis data graf atau biasa disebut graph database adalah solusi yang paling tepat. Pada tugas akhir ini akan mengaplikasikan graph indexing menggunakan algoritma GraphGrep. GraphGrep adalah metode yang paling tepat untuk studi kasus data bertipe molekul. Karena GraphGrep menganggap setiap node yang ada di graph database mempunyai nomor (id-node) dan label (label-node) Sehingga sangat cocok untuk tipe data molekul. GraphGrep menggunakan hash table (fingerprint) sebagai index, membandingkan fingerprint database dengan fingerprint query untuk mem-filter database dan menggunakan algoritma Ullman untuk melakukan subgraph matching. Dari penelitian ini diharapkan mampu menerapkan algoritma GraphGrep pada graph indexing dengan menggunakan dataset bertipe molekul serta menganalisis performansi yang dihasilkan.

Kata kunci: *graph, graph database, graph indexing, GraphGrep, subgraph matching, backtrack, Ullman*

Abstract

Graph database is a database that uses a graph structure to represent and manage the data. As we know, most of the database commonly used relational database as a relatively easy to use and support many types of data. However, for certain data types such as molecular data type that have characteristic labeled vertices and edges are not trending, relational database is less effective because of the type of data used are interrelated independently. To handle this, the graph database is the most approriate solution. In a graph database, it also have indexing method. For molecular data type study case GraphGrep is the most approriate method because it assume each node in the graph database has a unique number (id-node) and label (label- node). So it is suitable for molecular data type. GraphGrep using a hash table (fingerprint) as an index, comparing the graph database fingerprint with graph query fingerrint to filter the database and use Ullman algorithm to perform subgraph matching. From this study are expected to implement the GraphGrep algorithm in graph indexing using molecular data type study case and analyze the resulting performance.

Keywords: *graph, graph database, graph indexing, GraphGrep, subgraph matching, backtrack, Ullman*

1. Pendahuluan

Pertumbuhan data yang setiap saat selalu bertambah, memaksa penggunaan basis data hampir di segala bidang. Baik di bidang akademik, finansial, farmasi, dan sebagainya. Sebagian besar basis data yang digunakan untuk menangani hal tersebut adalah basis data relasional, karena penggunaannya yang mudah dan mendukung banyak tipe data, seperti *string*, *integer*, *float*, dan *binary*.

Namun, untuk tipe data tertentu seperti tipe data molekul yang memiliki ciri vertex berlabel dan edge yang tidak berarah, basis data relasional kurang begitu efektif digunakan karena tipe data tersebut memiliki keterkaitan secara independen. Untuk menangani hal tersebut, basis data graf atau biasa disebut *graph database* adalah solusi yang paling tepat. Namun terdapat perbedaan dalam melakukan pencarian query antara basis data relasional dan basis data graf dikarenakan data yang terdapat dalam basis data graf adalah data berbentuk graf.

Untuk menangani hal tersebut, maka dibutuhkan metode pencarian (*indexing)* yang baru. Sedangkan tipe data graf yang akan digunakan kali ini adalah memiliki vertex berlabel dan edge yang tidak berarah. Dari sekian banyak metode indexing seperti Gindex[4], FGIndex[2], C-tree[3], dan Graphgrep[1], GraphGrep adalah metode yang dianggap paling tepat. Karena tahapan-tahapannya yang umum digunakan seperti menggunakan *hash table* (*fingerprint)* untuk *indexing* dan membandingkan *fingerprint database* dengan *fingerprint query* untuk mem-*filter database* dan menggunakan algoritma Ullman untuk melakukan *subgraph matching*[6].

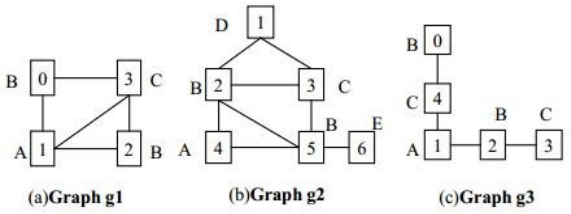
2. Landasan Teori

2.1 Graph Database

Graph database adalah basis data yang menggunakan struktur graf untuk merepresentasikan dan mengelola data. Graf adalah kumpulan dari vertex (*vertices*) dan sisi (*edges*). Setiap vertex menyatakan entitas (seperti orang, produk, dll) dan setiap sisi menyatakan hubungan atau relasi antara dua vertex. Pada tugas akhir ini setiap vertex pada graf diidentifikasikan dengan *id* dan *label*. Selain itu tipe sisi yang digunakan adalah tidak berarah.

2.2 GraphGrep

GraphGrep merupakan salah satu teknik *graph indexing*[1] yang memanfaatkan *path* dari graf. GraphGrep menganggap setiap node yang ada di graph database mempunyai nomor (*id-node*) dan *label* (*label-node*). Sisi yang digunakan adalah tidak berarah dan tidak berlabel. Sebagai contoh di gambar 2.6, (C,A) adalah *label-path* di g1 dan (3,1) adalah *id-path*-nya[1][2].



Gambar 2-1 : Graph database yang terdiri dari 3 graf[7]

Algoritma GraphGrep secara umum terbagi menjadi beberapa tahapan yaitu *index construction*, *database filtering*, dan *subgraph matching.*

2.3.1 Index Construction

Untuk setiap graf dan masing-masing vertexnya, akan dicari semua *path* yang bisa dibentuk, dimulai dari panjang *path* dengan l_p = 1 hingga l_p yang ditentukan. Karena beberapa path kemungkinan berada pada urutan *label* (*label-path)* yang sama, maka *id-paths* dengan *label* yang sama tersebut akan dijadikan satu kelompok. *Path-representation* dari sebuah graf adalah kumpulan dari *label-path* yang setiap *label-path* memiliki kumpulan *id-path*. *Keys* yang terdapat di tabel *hash* adalah *hash values* dari *label-paths*. Setiap baris mengandung jumlah *id-paths* yang terkait dengan key (*hash value*) dalam setiap graf. *Hash table* yang digunakan bisa juga disebut *fingerprint*[1][2].

2.3.2 Database Filtering

Mem-filter pencarian untuk mendapatkan candidate set dengan cara membandingkan fingerprint graph database dengan fingerprint graph query untuk membuang setiap graf di database jika value fingerprint graph database lebih besar atau sama dengan value fingerprint graph query.

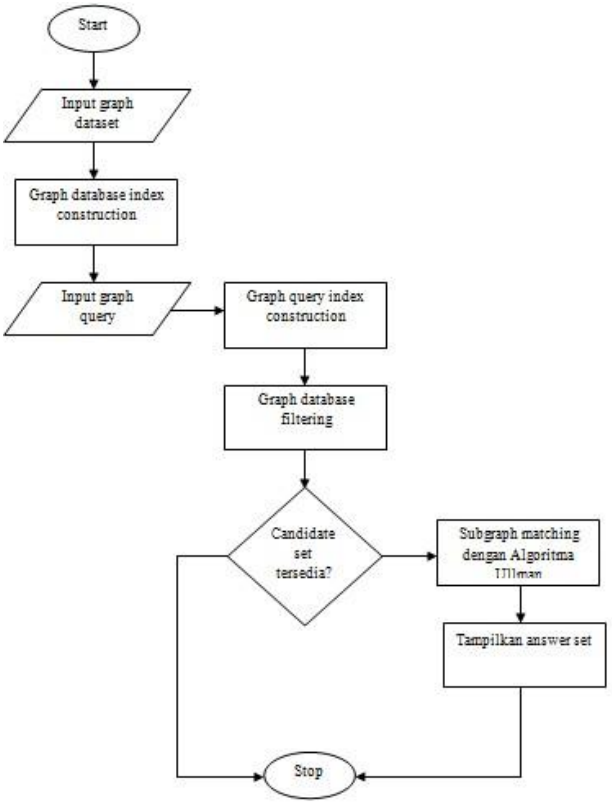
2.3.3 Subgraph Matching

Setelah melakukan filtering, kita akan melakukan subgraf isomorfisme antara graf query dengan setiap graf dari candidate set. Kemudian kita menggunakan algoritma Ullman yang menggunakan adjacency

matrix dan metode backtrack untuk melakukan subgraf isomorfisme.

3. Perancangan dan Implementasi

Berikut ini adalah gambaran umum sistem yang akan dibangun pada Tugas Akhir ini.



Gambar 3-1 : Alur proses sistem

Berdasarkan gambar 3-1, untuk pertama kali *user* akan memasukkan *dataset graph database* yang berformat SMILES. Setelah itu sistem akan memproses data tersebut untuk membangun index dengan cara membuat *path representation* untuk menghasilkan *fingerprint* dari setiap graf yang ada di dataset. Setelah itu *user* akan memasukkan *graph query* dan sistem juga akan membangun index dari *graph query* tersebut. Pada tahap *graph database filtering*, sistem akan membandingkan antara *value* setiap *fingerprint graph database* dengan *value* setiap *fingerprint graph query* untuk mendapatkan *candidate set.* Jika *candidate set* tidak ditemukan maka sistem akan langsung selesai dan menyatakan bahwa *graph query* tidak terdapat di *graph database* tersebut, Tapi jika terdapat *candidate set*, maka sistem akan masuk ke tahap selanjutnya yaitu melakukan *subgraph matching* dengan cara melakukan subgraf isomorfisme antara *graph query* dengan setiap graf dari *candidate set* untuk mendapatkan *answer set.* Setelah itu sistem akan menampilkan *answer set.*

4. Pengujian dan Analisis

4.1 Skenario Pengujian

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

1. Pengujian Pengaruh l_p terhadap Waktu Pembentukan *Index*
2. Pengujian Pengaruh l_p terhadap Hasil *Filtering*
3. Pengujian Waktu dan Penemuan Solusi *Subgraph Matching*

4.2.1 Pengujian Pengaruh l_p terhadap Waktu Pembentukan *Index*

Pada pengujian ini akan dilakukan pengujian pengaruh l_p terhadap waktu pembentukan *index* pada *graph database* yang digunakan. Pengujian pembentukan *index* dilakukan pada setiap l_p yang telah ditentukan yaitu 4, 7, dan 10 untuk setiap dataset dengan data yang berjumlah 250, 500, 750, 1000, 1250, dan 1500.

4.2.2 Pengujian Pengaruh lp terhadap Hasil Filtering

Pada pengujian ini akan dilakukan pengujian pengaruh l_p terhadap hasil *filtering* dengan menggunakan 3 jenis *query*. Berikut ini adalah detil jenis *query* pada pengujian yang dilakukan :

Tabel 4-1 : Tabel *query* pengujian

No	Query	SMILES	Jumlah Vertex	Atom Ion	Siklik	Depth
1	Q1	O=C(N)CC#N	6	Tidak	Tidak	5
2	Q2	ON=Cc1ccc(O)cc1	10	Tidak	Ya	8
3	Q3	O=[N+]([O-])c1ccc(cc1)CC#N	12	Ya	Ya	10

4.2.3 Pengujian *Subgraph Matching* dengan Algoritma Ullman

Pengujian ini dilakukan untuk mengetahui waktu *subgraph matching* antara *candidate set* yang dihasilkan dari setiap dataset pada *graph query* Q1 (diambil dari pengujian sebelumnya) dengan l_p = 5 yaitu menyesuaikan panjang maksimal *path/depth* pada Q1. Teknik s*ubgraph matching* yang digunakan menggunakan algortima Ullman. Selain itu pengujian ini dilakukan untuk mengetahui *answer set* berdasarkan *candidate set* yang dihasilkan.

4.3 Analisis Hasil Pengujian

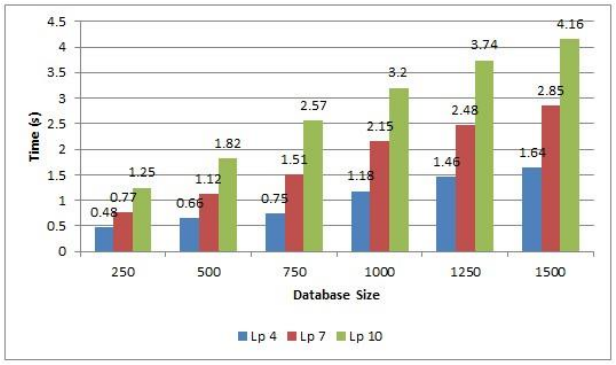
Setelah dilakukan pengimplementasian sistem, skenario pengujian yang sudah dirancang dapat terlaksanakan dengan baik. Pada bagian ini akan dipaparkan hasil pengujian yang telah dilaksanakan.

4.3.1 Analisis Hasil Pengaruh l_p terhadap Waktu Pembentukan *Index*

Berdasarkan pengujian pengaruh l_p terhadap waktu pembentukan *index* pada *graph database* yang digunakan, berikut adalah hasil yang didapat:

Tabel 4-2 : Tabel pengaruh l_p terhadap waktu pembentukan *index*

	250	500	750	1000	1250	1500
Lp 10	1.25	1.82	2.57	3.20	3.74	4.16
Lp 7	0.77	1.12	1.51	2.15	2.48	2.85
Lp 4	0.48	0.66	0.75	1.18	1.46	1.64



Gambar 4-1 : Grafik pengaruh l_p terhadap waktu pembentukan *index*

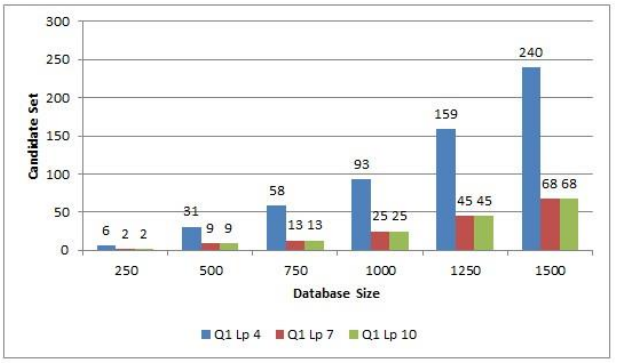
Berdasarkan grafik pada gambar 4-1, maka dapat disimpulkan bahwa semakin tinggi l_p yang digunakan maka akan semakin lama pula waktu yang dibutuhkan untuk membangun *index* pada *graph database*.

4.3.2 Analisis Hasil Pengaruh l_p terhadap Hasil Filtering

Berdasarkan pengujian proses *filtering* yang dilakukan terhadap 3 jenis *query* yang berbeda terhadap l_p yang berbeda juga pada ukuran *database* tertentu. Berikut adalah hasil yang didapat:

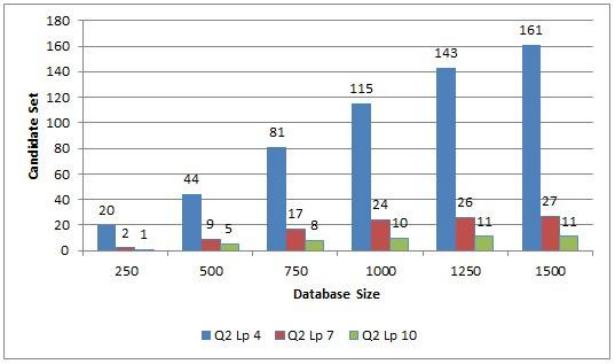
Tabel 4-3 : Tabel pengaruh l_p *pada* graph query *Q1, Q2, dan Q3 terhadap hasil* filtering

	250	500	750	1000	1250	1500
Q1 Lp 10	2	9	13	25	45	68
Q1 Lp 7	2	9	13	25	45	68
Q1 Lp 4	6	31	58	93	159	240
Q2 Lp 10	1	5	8	10	11	11
Q2 Lp 7	2	9	17	24	26	27
Q2 Lp 4	20	44	81	115	143	161
Q3 Lp 10	0	2	3	3	3	3
Q3 Lp 7	1	8	16	25	35	56
Q3 Lp 4	14	30	65	92	125	155



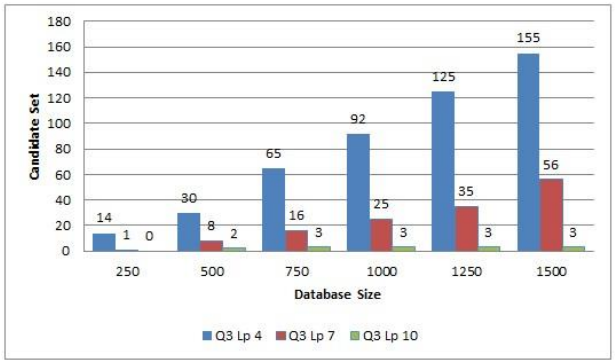
Gambar 4-2 : Grafik pengaruh l_p pada *graph query* Q1 terhadap hasil *filtering*

Berdasarkan gambar 4-2 di atas, semakin tinggi l_p yang digunakan maka *candidate set* yang dihasilkan akan semakin sedikit. Hal ini mengakibatkan semakin banyak *label-path* yang dihasilkan saat pembangunan *index* pada *graph database* dan *graph query*. Lalu menghasilkan baris *hash table* yang semakin banyak sehingga proses *filtering* akan semakin spesifik. Namun pada l_p = 7 dan l_p = 10 dihasilkan jumlah *candidate set* yang sama. Hal ini dikarenakan panjang *path* maksimal/*depth* yang dihasilkan oleh Q1 bernilai 5, sehingga dapat disimpulkan bahwa l_p yang digunakan akan bergantung kepada *depth* suatu *graph query*.



Gambar 4-3 : Grafik pengaruh l_p pada *graph query* Q2 terhadap hasil *filtering*

Berdasarkan gambar 4-3 di atas, maka pernyataan yang terkait gambar 4-2 sebelumnya terbukti benar untuk semakin tinggi nilai l_p, maka *candidate set* yang dihasilkan semakin sedikit.



Gambar 4-4 : Grafik pengaruh l_p pada *graph query* Q3 terhadap hasil *filtering*

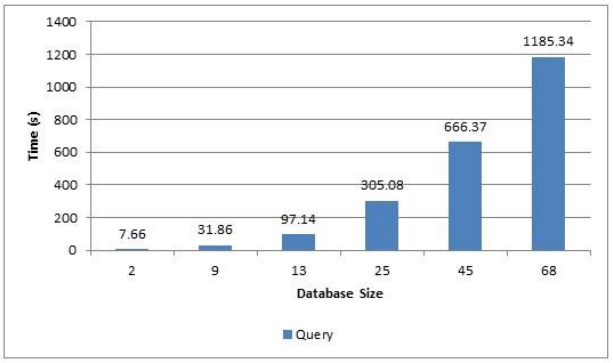
Berdasarkan gambar 4-4 di atas, saat pengujian l_p = 10 pada jumlah dataset sebanyak 250 graf molekul tidak menghasilkan *candidate set*. Hal ini menunjukkan bahwa tidak ada satupun graf pada dataset tersebut merupakan subgraf isomorfik terhadap Q3. Sehingga hal ini menunjukkan bahwa penggunaan l_p yang paling efektif pada *graph database* maupun *graph query* adalah ketika nilainya sama dengan *depth* suatu *graph query*.

4.3.3 Analisis Hasil Waktu dan Penemuan Solusi Subgraph Matching

Berdasarkan pengujian waktu *subgraph matching* yang dilakukan terhadap *candidate set* pada *graph query* Q1 menggunakan l_p = 5, berikut adalah hasil yang didapat:

Tabel 4-4 : Tabel pengujian waktu *subgraph matching* antara *candidate set* dengan *graph query* Q1

	2	9	13	25	45	68
Query	7.66	31.86	97.14	305.08	666.37	1185.34

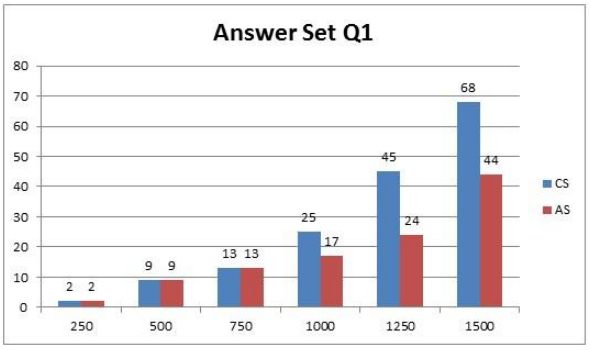


Gambar 4-5 : Grafik pengujian *subgraph matching* pada *candidate set* Q1

Berdasarkan tabel 4-4 di atas dan tabel 4-5 di bawah, maka semakin banyak *candidate set*, semakin banyak juga waktu yang dibutuhkan untuk melakukan *subgraph matching*. Selain itu juga *candidate set* yang didapatkan kemungkinan berupa *answer set.*

Tabel 4-5 : Tabel perbandingan *candidate set* dan *answer set* pada graph *query* Q1

No	Candidate Set	Answer Set
1	2	2
2	9	9
3	13	13
4	25	17
5	45	24
6	68	44



Gambar 4-6 : Grafik perbandingan candidate set dan answer set pada graph query Q1

Berdasarkan grafik pada gambar 4-6 diatas *subgraph matching* dengan algoritma Ullman berhasil dilakukan dan menunjukkan bahwa pada *dataset* berukuran 250, 500, dan 750 *candidate set* yang dihasilkan merupakan *answer set*. Sedangkan pada ukuran 1000, 1250, dan 1500 terdapat pengurangan, sehingga *answer set* yang dihasilkan lebih sedikit dibandingkan *candidate set*.

5. Penutup

5.1 Kesimpulan

Berdasarkan pada pengujian yang telah dilakukan pada bab 4 dari buku tugas akhir ini, diperoleh kesimpulan sebagai berikut:

1. Semakin besar nilai l_p yang digunakan, semakin besar pula waktu yang dibutuhkan untuk pembangunan *index*.
2. Semakin besar nilai l_p yang digunakan untuk melakukan *filter database*, maka *candidate set* yang dihasilkan semakin sedikit.
3. Nilai l_p yang digunakan bergantung kepada *depth graph query* yang digunakan.
4. Implementasi *subgraph matching* dengan algoritma Ullman berhasil dilakukan dan dapat menghasilkan *answer set* yang sama atau lebih sedikit dari *candidate set*.

5.2 Saran

Setelah proses pengerjaan tugas akhir ini, berikut ini adalah beberapa saran yang dapat dijadikan acuan untuk penelitia kedepannya, khususnya untuk ruang lingkup *graph indexing* :

1. Tipe data yang digunakan pada penelitian ini adalah tipe data molekul yang memiliki struktur graf dengan vertex berlabel dan sisi yang tidak berarah juga tidak berbobot, untuk kedepannya dapat dilakukan dengan tipe data lainnya yang memiliki struktur graf berbeda.
2. Untuk selanjutnya dapat menggunakan dataset yang memiliki rata-rata vertex yang jauh lebih besar dari sekarang, misalnya rata-rata 50 vertex keatas.
3. Diharapkan untuk penelitian selanjutnya dapat menerapkan algoritma *subgraph matching* yang berbeda, seperti VF.

Daftar Pustaka

[1] Shasha, D., J.T-L Wang & R. Giugno. 2002. *Algorithmics and applications of tree and graph searching*. In Proc. 21th ACM Symp. Principles of Database Systems(PODS'02) : 39.

[2] Cheng, J., Y. Ke, W. Ng & A. Lu. 2007. *Fg-index: Towards verication-free query processing on graph databases*. In SIGMOD.

[3] He, H. & A. K. Singh. 2006. *Closure-tree: An index structure for graph queries*. In ICDE : 38.

[4] Yan, X., P. S. Yu, & J. Han. 2004. *Graph indexing: A frequent structure-based approach*. In SIGMOD.

[5] Williams, J. H. D.W. & W. Wang. 2007. *Graph database indexing using structured graph decom-position*. InICDE.

[6] Putra, D. M., O. O. Sardjito & C. Lawrence. *Penerapan dan Implementasi Algoritma Backtracking.*

[7] Giugno, R. & D. Shasha. 2002. *GraphGrep: A Fast and Universal Method for Querying Graphs*.

[8] Ian, R., W. Jim & E. Emil. 2013. *Graph databases*. O'Reilly Media.

[9] Balaban, A. T. 1985. *Applications of Graph Theory in Chemistry*. Bucharest : Department of Organic Chemistry, Polytefhnic Institute.

[10] Keijo, R.. 2013. *Graph Theory*.

[11] Silberschatz, A., Korth & Sudarshan. (2007). Database System Concepts. New York : McGraw-Hill.