Implementasi Prinsip MDL untuk Kompresi Graph Database Menggunakan Algoritma Greedy

Harris Febryantony Z[1], Kemas Rahmat Saleh W, ST., M.Eng[2], Alfian Akbar Gozali, ST., MT[3]
[1,2,3]Fakultas teknik Informatika – Universitas Telkom
[1]harryz@students.telkomuniversity.ac.id, [2]bagindokemas@tass.telkomuniversity.ac.id,
[3]alfian@tass.telkomuniversity.ac.id,

Abstrak
Graph secara konsep merupakan abstraksi yang secara fundamental telah lama dipakai, yang memungkinkan untuk memodelkan pada sistem di dunia nyata. Begitu pula pada data, data jenis apapun dapat dimodelkan relasi antar data tersebut menggunakan graph. Graph database diadopsi untuk memudahkan dan membantu dalam memahami, memodelkan, serta menganalisis suatu proses. Graph database sangat cocok digunakan pada data bersifat tidak terstruktur dan semi terstruktur dibanding *relational database* yang mana memiliki kelemahan jika data dan ukuran tabel bertambah menyebabkan kemungkinan *join* antar tabel sangat besar. Dalam aplikasinya jumlah data pada *graph database* semakin lama akan berkembang semakin besar menjadi jutaan bahkan miliaran *node* dan *edge*, sehingga *cost* untuk untuk melakukan analisa dan visualisasi *graph databse* menjadi sangat besar untuk kemampuan sistem saat ini.
Untuk menyelesaikan permasalahan tersebut maka diperlukan suatu metode untuk mengurangi ukuran dari *graph* tetapi tetap menyimpan informasi-informasi penting dari *graph*. Dengan menerapkan prinsip *Rissaenen's Minimum Description Length* (MDL) dan melakukan penggabungan secara *greedy* serta mengombinasikan dengan representasi *graph* G yang terdiri dari *Graph Summary* dan sebuah set *Correction*, maka dapat dihasilkan *graph database* yang dikompres dengan baik.

Kata Kunci: *graph database, graph summarization, graph representation, MDL principle, lossles, lossy, compression, greedy, Rissaenen's Minimum Description Length*

Abstract
Graph is an abstract concept that is fundamentally has been used, which allows to model the real world system. So did in data, any type of data can be modelled using the relationship of graph. Graph database was adopted to facilitate and assist in understanding, modeling, and analyzing a process. Graph database very suitable to use in unstructured and semi-structured data, compare to relational database which have a disadvantage that when increasing data that involving increasing possibility of joining between table. As well, the need to analyze and visualize large database. In application, the amount of data in graph database are rapidly growing to millions and event billions of nodes and edges so the computation cost to perform analysis and visualization of the graph can be very hard for current system capabilities.
To resolve these problem above, we need a method that can reduce the size of the graph without losing it important information, by implementing the Principle of Minimum Description Length Rissaenen's (MDL) with greedy to merge and combine it with the representation graph G which consists of Graph Summary and a set of Correction, then the resulting graph database is highly compressed.

***Keywords**: graph database, graph summarization, graph representation, MDL principle, lossles, lossy, compression, greedy, Rissaenen's Minimum Description Length*

1. PENDAHULUAN

1.1. Latar Belakang

Teori graph telah secara aktif diteliti dan ditingkatkan oleh matematikawan, sosiolog, antropolog, serta berbagai bidang keilmuan lainnya. Semenjak pelopori oleh Euler pada abad ke 18 hingga sampai pada masa sekarang ini. Tidak terkecuali, dunia teknologi informasi juga ikut berpartisipasi dalam penerapan, penelitian, serta penggunaan teori graph, salah satunya adalah pada graph database.

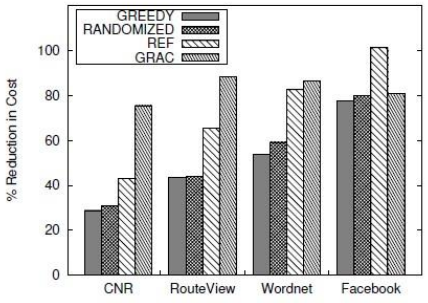
Graph database telah menjadi tren saat ini karena kemampuannya dalam memproses data-data yang bersifat struktural dan semi struktural dibanding RDBMS(Relational Database Management System). Yaitu dalam memodelkan permasalahan dunia nyata yang mencakup pada jejaring sosial, sistem rekomendasi, World Wide Web(WWW), pemantauan jaringan, manajemen data center, serta banyak lainnya. Hal ini juga digerakkan oleh kesuksesan secara komersial oleh perusahaan-perusahaan besar seperti Facebook, Google, dan Twitter [1] yang telah memusatkan bisnis model mereka pada teknologi graph.

Sejalan dengan perkembangannya, ukuran graph database ikut berkembang secara masif. Sehingga merupakan suatu tantangan untuk melakukan visualisasi serta analisis terhadap informasi pada database dengan milyaran bahkan triliunan node dan edge. Database tersebut terlalu besar untuk dimasukkan semuanya ke dalam memory [2] sistem saat ini. Oleh karena itu diperlukan representasi graph database tersebut dengan ukuran yang lebih kecil.

Ada berdapat beberapa algoritma kompresi graph database, seperti: Reference Encoding(REF) [3]-teknik ini sangat sukses dan populer untuk kompresi web-graph; Graclus(Grac) [4]-ini merupakan algoritma graph clustering yang membagi node dari graph berbobot menjadi cluster sehingga jumlah bobot antar-cluster edge diminimalisasi; Sampling(SAMP)-metode edge sampling sederhana.

Dalam penelitiannya S. Navlakha, R. Rastogi dan N. Shrivastava [2], mereka mengemukakan teknik untuk melakukan komputasi representasi graph yang menghasilkan hasil kompresi dengan baik, representasi tersebut terdiri atas dua bagian: pertama graph summary(lebih kecil dari graph input) yang menampung cluster penting dan hubungan di dalam graph input, bagian kedua adalah suatu set correction yang nantinya jika diperlukan digunakan untuk membentuk kembali graph input. Untuk membentuk representasi graph, digunakan algoritma greedy dengan menerapkan prinsip Rissaenen's Minimum Description Length (MDL).

Graph representation ini memungkinkan supernode untuk memiliki self-edges, yang sangat efektif untuk menyatukan kelompok yang berdekatan menjadi satu supernode. Kedua, di sini menyimpan superedge antara supernode A dan B hanya jika node di A secara sangat dek terhubung dengan B. Sehingga representasi ini lebih sedikit ter-cluster, ukuran kecil, dan lebih cocok untuk visual data mining [2].



Gambar 0-1 Perbandingan Algoritma Kompresi Graph [2]

Dari pengujian yang dapat dilihat padaGambar 0-1, greedy memberikan kompresi tertinggi pada graph representation dibanding metode kompresi yang ada, serta dengan cost melakukan komputasi yang minimum dan efektif.

2. LANDASAN TEORI

2.1. Teori Graph

Sebuah graph $G = (V, E)$ dibentuk oleh sekumpulan objek $V = \{v_1, v_2, v_3, \ldots\}$ yang disebut himpunan node (vertex, titik, atau simpul) dan sebuah himpunan objek $E = \{e_1, e_2, e_3, \ldots\}$ yang disebut himpunan edge(line atau sisi) sedemikian hingga tiap edge e_k menghubungkan pasangan node (v_i, v_j). Atau secara sederhana graph dibentuk dari node dan edge yang menghubungkan antar node. Cara umum untuk menggambarkan atau merepresentasikan graph adalah berupa diagram, yaitu dengan menggambar sebuah titik untuk setiap node dan menggabungkan kedua titik tersebut dengan sebuah garis(edge). Garis penghubung tersebut dapat berupa garis lurus atau garis lengkung, Demikian pula ukurannya, dalam penggambaran bukanlah satu yang diperhatikan. Hal terpenting yang perlu diperhatikan adalah incidece antara node dengan edge-nya, perhatikan contoh graph berikut:

Dalam graph memiliki himpunan node $V = \{v_1, v_2, v_3, v_4, v_5\}$ dan edge $E = \{(v_1, v_2), (v_1, v_5), (v_2, v_3), (v_2, v_5), (v_3, v_4)\}$ atau bisa dituliskan $E = \{e_1, e_2, e_3, e_4, e_5\}$

dengan jumlah node dan edge masing-masing lima.

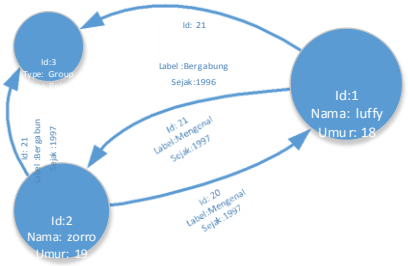
Dalam graph derajat sebuah node u adalah jumlah edge yang terhubung dengan node tersebut. Karena pada graph tak berarah sebuah edge bisa berarah keluar dari sebuah node atau masuk dari sebuah node, maka derajat sebuah node pada graph tak berarah dikalikan dengan 2. Derajat rata-rata sebuah graph merupakan jumlah edge dalam sebuah graph dibagi dengan jumlah nodenya. Perhitungan ini dapat digunakan untuk mengetahui kepadatan suatu graph. Pada graph tak berarah, derajat rata-ratanya dapat dihitung dengan rumus:

$$\frac{2 \times |\text{edge}|}{|\text{node}|}$$

2.2. Graph Database

Salah satu penerapan teori graph pada bidang teknologi informasi adalah graph database, yaitu struktur data yang menggunakan graph untuk merepresentasikan entitas satu objek sebagai node dan bagaimana entitas itu berhubungan dengan lingkungannya(antar objek), yang mana hubungan tersebut sebagai edge. Serta properti-properti yang merepresentasikan informasi terkait node dan edge.

Model graph yang paling banyak digunakan [1] pada graph database adalah model property graph. yaitu struktur graph yang terdiri atas node, edge, dan properties untuk merepresentasikan dan menyimpan data, seperti contoh pada Gambar 0-2 dibawah ini:



Gambar 0-2 Contoh Struktur Graph Databse

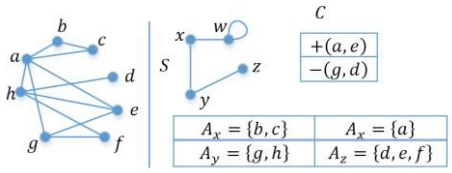
2.3. Representasi Umum *Graph*

Representasi dari sebuah graph merupakan gambaran gambaran terhadap graph terseut. tersebut. Secara umum, untuk merepresentasikan sebuah graph [2] dapat dijabarkan sebagai berikut, jika diberikan sebuah graph $G = (V, E)$, representasi untuk graph G adalah $R = (S, C)$ yang terdiri dari graph summary S dan sebuah himpunan edge corrections C. Graph summary tersebut merupakan struktur graph agregasi, di supernode yang berkorespondensi ke satu set V_i dari node dalam G, dan setiap edge $(V_i, V_j) \in S$ disebut dengan superedge yang merepresentasikan edge antara setiap pasang node di V_i dan V_j. Bagian kedua dari representation adalah sebuah himpunan correction C yang berisi koreksi dari superedge untuk edge pada graph G, yang memiliki notasi positif $(' + ')$ atau negatif $(' - ')$.

Penggunaan struktur summary S pada representasi graph adalah untuk memanfaatkan kesamaan struktur hubungan antar node yang secara umum banyak terdapat pada graph. Jika dua node memiliki edge ke sebuah himpunan node yang sama(atau set sangat mirip), maka kedua node tersebut dapat digabungkan menjadi satu edge yang kemudian disebut dengan supernode kemudian kedua edge menuju tetangga yang sama diganti dengan sebuah superedge. Jika kedua node yang digabungkan dihubungkan oleh sebuah edge, maka tambahkan self-edge pada supernode hasil penggabungan. Ini jelas akan mengurangi jumlah total edge yang akan disimpan secara signifikan.

Pada bagian correction C memiliki fungsi untuk menyimpan koreksi terhadap superedge $(V_i, V_j) \in S$ pada saat melakukan pembentukan kembali dari summary graph menjadi graph awal. C berisi entri dengan bentuk $' - (V_i, V_j)'$ untuk edge yang tidak terdapat pada G jika superedge yang sama tidak dimasukkan ke dalam S, akan berisi entri dengan bentuk $' + (V_i, V_j)'$ untuk edge yang terdapat dalam G. Rekonstruksi dari summary graph dilakukan dengan mengembangkan summary S sehingga supernode dipecah kembali menjadi node-node pembentuknya. Untuk setiap supernode $V_i \in S$, kemudian pada setiap superedge $(V_i, V_j) \in S$, tambahkan edge antar semua pasangan node (V_i, V_j) sehingga $V_i \in V_i$ dan $V_j \in V_j$. Dari pengembangan superedge tersebut, memungkinkan adanya edge hasil bentukan yang seharusnya tidak terdapat pada graph G mana setiap node $V_i \in V$, disebut dengan

dan sebaliknya. Untuk memperbaikinya maka digunakanlah himpunan corrections [illegible] Dilakukan dengan cara: didefinisikan fungsi [illegible] merekonstruksi keseluruhan graph. Berikut adalah gambaran graph beserta graph summary-nya:



Gambar 0-3 Kiri: Graph Original. Kanan Graph Representatio yang Berisi Graph Summary(s), Correction(c), dan Pemetaan Supernode

Pada Gambar di atas, memperlihatkan graph yang dikompres dari ukuran (jumlah edges) 11 menjadi 6 (4 pada graph summary dan 2 pada correction), untuk melakukan rekonstruksi dari graph di atas , dilakukan dengan memilih tetangga dari sebuah node (misal g) di dalam graph, langkahnya sebagai berikut: Pertama, temukan supernode(y) yang mengandung g. Kemudian tambahkan edges dari g ke semua node pada supernode yang bertetangga dengan y. Ini menghasilkan edges {(g,a),(g,d),(g,e),(g,f)}. Selanjutnya, lakukan koreksi ke himpunan edge, yaitu dengan menghapus semua edge dengan entri '-'(edge (g,d)), dan tambahkan edge dengan entri '+'(tidak ada pada contoh ini). Ini menghasilkan set edge pada tetangga g sebagai {(g,a),(g,e),(g,f)}, yang sama dengan graph original. Tahapan ini bisa dilakukan berulang pada semua node V_i untuk merekonstruksi graph G.

2.4. Representasi MDL

Prinsip Minimum Description Length (MDL) merupakan metode untuk inferensi(penarikan kesimpulan) induktif yang memberikan solusi umum pada model pemilihan masalah [8], MDL berdasarkan pada pemahaman berikut: setiap regularitas dalam data dapat digunakan untuk untuk mengompres data tersebut, misalnya untuk menggambarkan suatu data menggunakan simbol yang lebih sedikit dibanding penggunaan simbol yang diperlukan sebenarnya. Secara kasarnya [2], MDL menyatakan bahwa teori terbaik untuk menyimpulkan suatu bentuk set data adalah yang meminimalisasi jumlah dari (A) ukuran dari teori dan (B) ukuran data ketika di encode dengan teori.

Untuk penerapan dalam graph, data merupakan graph input G sedangkan teori adalah summary S dan correction C secara esensial merepresentasikan enkoding data sehubungan dengan teori. Didefinisikan cost dari representation $R = (S, C)$ menjadi jumlah storage cost dari dua komponennya sendiri, sehingga $cost(R) = |S| + |C|$. (hiraukan cost menyimpan V dan pemetaan M untuk supernode V karena ini secara umum lebih kecil dibanding cost penyimpanan himpunan edge E dan E. Pada persamaan cost kedua, sama dengan (B). Dengan demikian, jika $R = (S, C)$ menunjukkan cost minimal representation, maka prinsip MDL mengatakan bahwa R merupakan "kemungkinan terbaik" summary dari graph. Dengan kata lain R sebagai tambahan sebagai representasi yang terkompresi terbaik dari graph G, juga mengandung R yang mana merupakan graph summary terbaik. Sehingga cost minimal representasi R disebut dengan representasi MDL [2].

Himpunan edge E terbentuk hanya berdasarkan supernode yang terdapat dalam V. Cara menghitungnya dilakukan sebagai berikut, anggap supernode sembarang A dan B dalam V. Definisikan E_{AB} sebagai himpunan dari semua pasangan (x, y), sedemikian sehingga $x \in A$ dan $b \in B$; set ini merepresentasikan semua kemungkinan edge pada graph G yang mungkin ada antar kedua supernode tersebut. Selanjutnya, didefinisikan $E_{AB} \subseteq E_{AB}$ sebagai himpunan edge yang benar-benar ada dalam graph original $E_{AB} = E_{AB} \cap E$). Sekarang. Kita memiliki dua cara untuk enkoding edge pada E_{AB} menggunakan struktur summary dan corrections, Cara pertama adalah untuk menambahkan superedge (A, B) untuk A dan B sebagai correction positif untuk memasukkan semua edge E_{AB}. Cara kedua adalah menambahkan edge satu per satu sebagai correction positif untuk memasukkan semua edge E_{AB}. Jumlah bit yang dibutuhkan untuk kedua alternatif adalah $(1 + |E_{AB} - E_{AB}|)$ dan $(|E_{AB}|)$. Kemudian dipilih mana yang kebutuhan memorinya lebih kecil untuk enkoding edge E_{AB}[2]. Maka menghitung cost untuk merepresentasikan himpunan edge E_{AB} antara supernode A dan B dalam representasi adalah:

$$cost(E_{AB}) = \begin{cases} 1 + |E_{AB} - E_{AB}| & \text{jika dan hanya jika } |E_{AB}| > (|A| \cdot |B| + 1)/2, \\ |E_{AB}| & \text{lainnya} \end{cases}$$

Selanjutnya kita akan pada graph summary kemudian dipilih berdasarkan hasil perhitungan sebelumnya.

2.5. Algoritma *Greedy* dalam Kompresi *database*

Dalam kompresi graph database, greedy akan membentuk kandidat solusi dengan melakukan perhitungan terhadap nilai cost reduction pada sembarang pasangan node dam graph. Menurut pengamatan Saket Navlakha [2], sembarang pasangan node akan memiliki nilai cost reduction jika pasangan tersebut memiliki tetangga yang sama. Semakin besar jumlah tetangga yang sama semakin besar pula

pengaruhnya terhadap cost reduction pasangan tersebut. Berdasarkan observasi tersebut maka dapat definisikan ��� adalah cost reduction untuk sembarang pasangan supernode (��dengan persamaan sebagai berikut:

$$��� = (��+ ��− ��) / (��+ ��)$$

Definisikan $�∈ ℕ_S$ dan $v∈ ℕ_S$, dengan himpunan tetangganya masing-masing N_u dan N_v terdapat edge (��dalam graph ꝺdengan node $u∈ N_u$ dan $v∈ N_v$.Pada persamaan di atas, nilai ��� didapat dari perhitungan perbandingan atau rasio pengurangan dari: cost hasil penggabungan ꝺdan ꝺmenjadi supernode dan penjumlahan cost dari supernode yang didapat dengan menjumlahkan semua cost dari superedge (��, ꝺmerupakan tetangga dari nodesuper ꝺdengan $v∈ N_v$ lihat persamaan, maka nilai

$$= ��= ���|N_{uv} − N_{vv}| + 1, |N_{vv}|\}$$

Alasan memilih bagian kecil daripada cost reduction absolut adalah bahwa akhir secara inheren terbias ke arah node dengan derajat lebih tinggi. Dikarenakan pada dasarnya memilih pasangan node dengan jumlah tetangga sama yang tertinggi. Node tersebut juga bisa memiliki jumlah tetangga tak sama yang tertinggi. Yang secara tidak langsung menyatakan bahwa node tersebut harus memiliki prioritas lebih rendah dibanding dua node dengan derajat lebih rendah dengan set tangga yang identik. Fractional cost reduction menjamin bahwa kasus tersebut tidak terjadi dengan menormalisasikan cost reduction dengan cost sebenarnya. Sangat penting untuk diamati bahwa normalisasi tersebut tidak membuat cit dari pasangan manapun berganti dari negatif menjadi positif(atau kosong) atau sebaliknya. Dengan kata lain. Jika menggabungkan sepasang memberi cost reduction. Maka ꝺ·) tidak akan menghalangi untuk memilihnya dengan cepat. (tapi hanya bisa mengubah urutan). Perhatikan nilai maksimum yang dapat diterima ��� adalah .5, ketika set tetangga dari dua mode identik [2].

Dalam prosesnya, algoritma greedy secara iteratif menggabungkan pasangan (�� pada graph dengan nilai maksimum ꝺ(pasangan terbaik).

3. PERANCANGAN SISTEM

3.1. Deskripsi Sistem

Dalam Tugas Akhir ini akan dibangun dan dilakukan pengujian terhadap sistem yang mengimplementasikan prinsip MDL untuk melakukan komputasi representasi graph dengan algoritma *greedy* terhadap *graph input*. *Dataset* input yang digunakan dalam pengujian sistem adalah *dataset* SNAP [9]. Kemudian dari *graph input* tersebut dilakukan proses komputasi sedemikian sehingga menghasilkan representasi *graph* yang terkompres dan *summary*-nya yang dapat dijadikan untuk analisis lebih lanjut. Untuk mengetahui validitas hasil representasi graph dilakukan rekosntruksi terhadap graph representasi untuk dilakukan perbandingan hasil *query* dengan graph input. Berikut gambaran umum dari sistem yang akan dibangun:

3.2. Graph Summaryzation

Graph summaryzation berfungsi untuk melakukan komputasi terhadap graph masukan, kemudian menghasilkan keluaran berupa representasi *graph* dalam bentuk *summary* dan *correction* dengan kompresi yang baik.

Library Graphstream [10] digunakan untuk memudahkan perhitungan, pemodelan struktur data, visualisasi *graph*, serta fungsionalitas lainnya. Pada *graphstream* terdapat objek *graph, node,* dan *edge* yang dapat ditambahkan atribut dengan tipe apa saja baik itu tipe primitif (string, integer, array, dst) atau tipe bentukan, Ini digunakan pada *supernode*, yang merupakan objek *node* dengan tambahan atribut "*node*" untuk menyimpan *node-node* pembentuknya. Sedangkan untuk kebutuhan visualisasi *graph* pada sistem digunakan objek *viewer*.

a. Fase inisialisasi

Padap fase ini yang proses pertama adalah membentuk *supernode* $ℕ_S$ dari setiap *node* $ℕ_G$ graph masukan, kemudian lakukan perhitungan *cost reduction* ��� terhadap semua kombinasi pasangan *supernode* yang berjarak dua *hop*.

Perhitungan *cost reduction* menggunakan persamaan memerlukan penggabungan melakukan perhitungan ꝺ, namun pada proses ini tidak dilakukan penghapusan *node* ꝺdan ꝺtetapi *node* ꝺ akan dihapus setelah perhitungan selesai.

Pemilihan pasangan *node* yang berjarak dua *hop* dilakukan karena berdasarkan pengamatan pada *paper* rujukan [2] yang mana pasangan *node* akan mempunyai nilai *cost reduction* positif jika memiliki tetangga yang sama. Jika memiliki tetangga yang sama, tentu tetangga tersebut menjadi penghubung antar kedua *node* tersebut yang memberikan jarak dua hop,

Selanjutnya, dari hasil perhitungan didapat nilai *cost reduction* untuk setiap pasangan yang di masukkan (ꝺ) jika tersebut bernilai positif (ꝺ>0). *Heap* ꝺ merupakan struktur data *Max-heap* yang selalu menempatkan nilai maksimum di posisi *head*-nya.Ini digunakan akan selalu memberikan nilai *cost reduction* maksimum

pada setiap tahap iteratif untuk memilih pasangan *node* pada fase selanjutnya.

b. Fase Penggabungan Iteratif

Setelah dilakukan fase inisiasi maka diperoleh nilai *heap* dari setiap pasangan *node* yang mungkin untuk digabungkan. Penggabungan pasangan *node* menjadi *supernode* dilakukan pada fase ini secara berulang hingga isi *heap* menjadi kosong.

Langkah pertama, ambil satu pasangan (�� dari posisi *head* (dengan nilai *cost reduction* maksimum) dari *heap*. Kemudian gabungkan pasangan *node* tersebut menjadi *supernode* (�= ��∪ ��. Pada proses penggabungan, setiap *node* anggota yang berada dalam atribut "*node*" pada ��dan ��dimasukkan ke dalam atribut "*node*" pada ��Selanjutnya tambahkan *superedge* untuk setiap tetangga ��dan ��terhadap ��Terdapat beberapa kondisi penambahan *superedge* pada penggabungan *node*, yaitu:

- Jika pada salah satu pasangan antar *node* anggota ��dengan ��terdapat *edge*, maka *supernode* akan memiliki *selfedge*.
- Jika ��dan ��memiliki tetangga yang sama, *maka superedge* ditambahkan dari ��ke tetangga tersebut.

Setelah proses penggabungan selesai, maka hapus kedua *node* ��dan ��dari *heap* yang mengandung ��atau ��Kemudian lakukan perhitungan *cost reduction* kembali dengan mengganti ��atau ��dengan ��untuk pasangan yang dihapus tersebut. Selanjutnya tambahkan ke dalam *heap* pasangan tersebut jika memiliki nilai *cost reduction* positif.

Misalnya:

- (��→hapus dari *heap*.
  (��→tambahkan ke dalam *heap* jika *cost reduction*-nya positif.
- (��→hapus dari *heap*.
  (��→tambahkan ke dalam *heap* jika *cost reduction*-nya positif.

Dengan adanya penggabungan ��dan ��tidak hanya akan mempengaruhi nilai *cost reduction* pasangan yang mengandung ��atau ��tetapi juga akan mempengaruhi nilai *cost reduction* pasangan yang mengandung *node* (misalkan ��yang sebelumnya bertetangga dengan ��dan ��. Maka, perlu dilakukan perhitungan kembali untuk memperbarui nilai *heap* pasangan yang mengandung ∈ ��tersebut, seperti langkah sebelumnya(penghapusan dan penambahan *heap*).

Kemudian lakukan perulangan ke langkah pertama hingga didapat hasil optimum, di mana tidak ada lagi pasangan *node* yang memiliki nilai *cost reduction* positif.

c. Fase Output

Fase terakhir merupakan proses pembentukan *edge summary* dan *correction* untuk setiap pasangan *superedge* (�, hasil penggabungan fase proses sebelumnya dengan ��kemudian lakukan perhitungan *cost* untuk *superedge* (��, yaitu dengan menghitung jumlah *edge* yang mungkin (��dan jumlah edge yang benar benar ada(�� di antara kedua *supernode* Dapat dilihat jika ��> (��+ 1) / 2 maka *superedge* akan terdapat dalam *graph*.

Dari perhitungan di atas kemudian lakukan pembentukan data *correction* dengan melihat kondisi: jika ��≥ (��+ 1) / 2 maka tambahkan (��ke *correction* (��sebaliknya tambahkan *correction* (��untuk semua (∈ ��

3.3. Rekonstruksi *Graph*

Pembentukan kembali g*raph summary* menjadi seperti *graph* masukan awal menggunakan *summary* dan *correction*, di sini penulis melakukan cara yang sedikit berbeda dengan cara pada *paper* rujukan [2], yaitu dilakukan dengan mengembangkan *node* anggota semua *supernode* dan kemudian lakukan perbaikan dengan menggunakan data dari koreksi, *be*rikut proses yang berlangsung pada proses rekonstruksi:

1. Untuk semua *supernode* pada (, bentuk *node* baru berdasarkan *node-node* anggota dari atribut '*node*'pada *supernode*.(semua atribut *node* anggota disalin ke dalam *node* baru)
2. Untuk semua superedge(�� lakukan penambahan *edge* untuk setiap pasangan *node* anggota ��dengan ��
3. Tambahkan edge(�� untuk setiap entri correction ′ + (��′.
4. Hapus edge(�� untuk setiap entri correction ′ − (��′

Setelah proses di atas dilakukan, akan didapat *graph* rekonstruksi yang sesuai dengan *graph* awal.

4. PENGUJIAN DAN ANALISIS

4.1. Tujuan Pengujian

Tujuan pengujian yang dilakukan pada penelitian ini adalah:

1. Untuk melakukan analisis terhadap hasil implementasi kompresi *graph database* dengan mengimplementasikan algoritma *greedy*.
2. Untuk melakukan analis terhadap tingkat kompresi *graph database* awal dan hasil kompresi oleh sistem.

3. Untuk melakukan analisis terhadap performasi representasi *graph* hasil.
4. Untuk melakukan pengecekan *lossless/lossy compression* terhadap *graph* rekonstruksi dari *graph* yang dikompres.

4.2. Data Pengujian

Untuk melakukan pengujian terhadap sistem yang dibangun digunakan *graph* masukan yang diambil dari beberapa *dataset* dan juga menggunakan *graph* masukan yang di*generate*/dibentuk oleh sistem sendiri menggunakan beberapa metode pembentukan graph. Berikut rincian data yang digunakan:

a. Dataset Graph

Dataset untuk *graph* masukan merupakan *dataset* dengan jumlah *node* dan *edge* bervariasi, diambil dari beberapa *dataset* g*ephi* yang tersedia pada [5].

- **Les Miserables**: Merupakan data hubungan karakter dalam novel Les Miserables, D. E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, Addison-Wesley, Reading, MA (1993).
- **Facebook**: Merupakan *dataset graph* tidak berarah hubungan pertemanan facebook yang di ekstrak dari akun facebook penulis sendiri menggunakan NetGet [11]. NetGet merupakan versi modifikasi dari aplikasi Netvizz oleh Bernhard Rieder,
- **Net Science**: Merupakan *dataset* jaringan tulisan ilmuwan yang meneliti dan bereksperimen dengan teori jaringan, yang disusun oleh M. Newman pada May 2006.
- **Power**: Merupakan *graph undirected* yang merepresentasikan topologi dari *Power Grid* di Amerika Serikat. Data dirangkum oleh D. Watts dan S. H. Stronggatz. D. J. Watts and S. H. Strogatz, Nature 393, 440-442 (1998).
- **Internet Routers**: Gambaran struktur internet pada level sistem autonomous, oleh Mark Newman pada 22 july 2006.

b. Graph yang di-generate

Pada pembentukan *graph* masukan dengan cara *generate*, jumlah *node* dapat ditentukan dan disesuaikan dengan kebutuhan pengujian, sedangkan jumlah *edge* yang dihasilkan bergantung kepada variabel-variabel masukan untuk membentuk *graph* dari jenis generator itu sendiri, Berikut generator yang digunakan:

- Generator Graph Random
- Generator Barabàsi-Albert
- Generator Graph Dorogovtsev-Mendes

4.3. Skenario Pengujian

Untuk mengamati serta mengevaluasi performa sistem dalam membentuk representasi *graph* dilakukan dengan mengujikan pada masukan *dataset* yang bervariasi. Setiap *graph* masukan akan diproses masing-masing dengan menggunakan algoritma *greedy* yang menghasilkan representasi *graph* masukan, kemudian dibandingkan dan dihitung parameter-parameter seperti waktu, derajat rata-rata, rasio kompresi, serta *cost reduction*- nya.

4.4. Analisis Hasil Pengujian

Pada bagian ini akan dikemukakan hasil pengujian dari skenario di atas dan kemudian dilakukan analisis.

a. Analisis Performa Implementasi Sistem

1. Analisis Graph Generator Random

Percobaan *graph* random di atas dilakukan pembentukan *graph* dengan menaikkan jumlah edge pada masing-masing *graph* dengan *node* 100 dan 1000. Untuk mengetahui seberapa besar pengaruh jumlah *edge* terhadap besar *edge reduction*. Dari tabel hasil di atas dapat dilihat bahwa sistem dapat mengurangi jumlah *edge* maksimum yaitu pada *graph* Rand1000_b dengan *edge reduction* sebesar 84% dan pengurangan jumlah *edge* terkecil adalah pada *graph* rand100_a sebesar 66,96% yang mana derajat rata-rata dari *graph* tersebut adalah 1,13, ini menunjukkan bahwa setiap *node* memiliki tetangga rata-rata 1.13 dengan demikian pada saat penggabungan dua *node,* kedua *node* tersebut tidak memiliki tetangga yang sama yang mengakibatkan *superedge* menuju tetangga salah satu *node* tidak ada. Semakin banyak *superedge* yang tidak terbentuk makan *graph* akan semakin menjadi ter-cluster, perhatikan pada gambar di bawah dapat dilihat bagaimana *graph original* rand100_a saat dibentuk *summary*-nya terpecah menjadi *sub-graph*

Pada percobaan yang dilakukan, *graph* masukan tetap memberikan *summary* yang ter-*cluster* sampai pada *graph* rad100_c dengan nilai derajat rata-ratanya 8.60. Pada *graph* rand1000_b memberikan *summary* yang ter-cluster walau dengan derajat rata-rata 11,04, *graph* yang dihasilkan berikutnya sudah tidak ter-*cluster*.

Perbandingan jumlah *node* antara *graph* masukan dengan *graph* hasil kompresi di mana tidak terjadi pengurangan yang cukup signifikan untuk jumlah *edge* yang berbeda yaitu dengan rata-rata 38%. Sedangkan untuk perbandingan pengurangan *edge* terjadi

peningkatan pengurangan jumlah *edge* dari dari 66,96% dari jumlah *edge* 115 pada rand100_a menjadi 79,24% dari jumlah *edge* 819 pada rand100_d, kemudian terjadi penurunan pada *graph* dengan jumlah *edge* selanjutnya. Dengan rata-rata pengurangan *edge*-nya adalah 67%.

2. Analisis Graph Barabàsi-Albert

Graph generator Barabàsi-Albert memberikan hasil *summary* yang tidak jauh berbeda dengan generator random. Berikut potongan hasil pengamatan (Lebih Lengkap dapat dilihat pada lampiran) terhadap *graph* Barabàsi-Albert:

Percobaan *graph Barabàsi-Albert* di atas dilakukan pembentukan *graph* dengan menaikkan jumlah edge pada masing-masing *graph* dengan *node* 100 dan 1000. Untuk mengetahui seberapa besar pengaruh jumlah *edge* terhadap besar *edge reduction*. Dari tabel hasil di atas dapat dilihat bahwa sistem dapat mengurangi jumlah *edge* maksimum yaitu pada *graph* Bara1000_e dengan *edge reduction* sebesar 79,64% dan pengurangan jumlah *edge* terkecil adalah pada *graph* rand1000_a sebesar 68,67% yang mana derajat rata-rata dari *graph* tersebut adalah 1,29, ini menunjukkan bahwa setiap *node* memiliki tetangga rata-rata 1.29 dengan demikian pada saat penggabungan dua *node*, kedua *node* tersebut tidak memiliki tetangga yang sama yang mengakibatkan *superedge* menuju tetangga salah satu *node* tidak ada. Semakin banyak *superedge* yang tidak terbentuk makan *graph* akan semakin menjadi ter-cluster, perhatikan pada gambar di bawah dapat dilihat bagaimana *graph original* bara1000_a saat dibentuk *summary*-nya terpecah menjadi *sub-graph*.

Dari hasil pengujian ini dapat dilihat secara visual tidak terjadi peng-clusteran mulai dari *graph* Bara100_d dengan derajat rata-rata 15,28 dan pada *graph* Bara1000_e dengan derajat rata-rata 31,73.

Pengurangan *node* untuk *graph* Bara100 tidak dipengaruhi secara signifikan oleh penambahan jumlah *edge*. Terjadi penurunan jumlah pengurangan pada *graph* Bara100 dari awal hingga *graph* Bara100_e kemudian naik pada *graph* Bara100_f. Pengurangan *node* terkecil sebesar 40% pada *graph* Bara100_d dan Bara100_e dan yang terbesar pada *graph* Bara100_a sebesar 52% dan dengan rata-ratanya adalah sebesar 43,33%, sedangkan untuk pengurangan *edge* terbesar terjadi pada *graph* Bara100_f sebesar 78,55% dan terkecil pada *graph* Bara100_c sebesar 69,49% dengan pengurangan *edge* rata-rata sebesar 73.44%.

Pengurangan *node* terbesar terjadi pada *graph* Bara100_e sebesar 38,90% dan terbesar pada *graph* Bara1000_b 60,60% dengan pengurangan *node* rata-rata 49,96%, sedang pengurangan *edge* paling besar pada Bara1000_e sebesar 79,64% dan terkecil pada Bara1000_a sebesar 68,67%, dengan rata rata pengurangan *edge* sebesar 75,61%.

Dari pengujian yang dilakukan pada *graph Barabàsi-Albert* dapat dihasilkan *graph output* dengan rasio kompresi rata rata yaitu 69,47%, serta *edge reductionnya* di bawah random dengan rata-rata 74,43%

3. Analisis Graph Generator Dorogovtsev-Mendes

Pengujian pada *graph Dorogovtsev-Mendes* menghasilkan *edge reduction* yang bernilai rata-rata 55.11% dan dikarenakan pada generator ini selalu menghasilkan *graph* dengan kisaran derajat rata-rata yang rendah, maka menghasilkan *graph* yang sedikit tar-*cluster*. berikut data hasil kompresi *graph Dorogovtsev-Mendes*:

Karena proses pembentukan *graph input Dorogovtsev-Mendes* sama untuk variabel masukan berbeda, graph yang dihasilkan memiliki derajat rata rata relatif sama, yaitu 4. Sehingga cost reduction yang didapat juga relatif sama yaitu pada nilai rata rata 55.11%, tidak terlalu terjadi perubahan hasil kompresi *graph* dengan adanya variasi jumlah *node* dan *edge*,

Pengurangan jumlah node tertinggi terjadi pada *graph* Dorogov_b dan Dorogov_c sebesar 45,50%, dan yang terkecil pada Dorogov_a sebesar 43%, sedangkan pengurangan *edge* terbesar ada pada Dorogov_a sebesar 64,47% dan terkecil pada Dorogov_e dengan nilai 56,96%.

4. Analisis Graph Dataset.

Dari data di atas dapat dilihat bahwa *graph* masukan dari *dataset* menghasilkan cost *reduction* dengan rata-rata 69.66% dengan *cost reduction* terbesar pada *graph* Facebook sebesar 78,02% dan terkecil pada graph Power 58,38%.

Pengurangan jumlah *node* untuk setiap masing-masing *graph input* dengan rata-rata sebesar 49.05% yang terbesar adalah pada *graph* masukan LesMiserables sebesar 59.74% dan yang terkecil pada *graph* Facebook sebesar 41.09%, sedangkan untuk pengurangan *edge* didapat hasil rata-rata 76,93% dengan pengurangan *edge* terbesar pada *graph* LesMiserables sebesar 79.92%, dan yang terkecil pada *graph* Power sebesar 71,16%.

4.5. Analisis Waktu Kompresi.

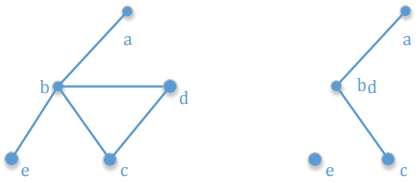
Dari beberapa *dataset* yang telah diujikan, dapat dilihat bahwa jumlah *edge* sangat mempengaruhi lama waktu eksekusi yang diperlukan oleh sistem untuk membentuk *summary*. Berikut adalah waktu yang diperlukan oleh system dalam milidetik:

Waktu yang diperlukan untuk membentuk *graph summary* semakin besar sebanding dengan jumlah *edge* pada *graph* tersebut, dengan jumlah yang besar tentu derajat rata-rata *graph* tersebut menjadi tinggi dan membuat iterasi penentuan kandidat solusi pasangan yang akan digabungkan menjadi semakin lama, karena kemungkinan sepasang *node* berjarak 2-Hop semakin banyak maka kandidat pasangan yang dimasukkan ke dalam *heap* untuk diproses semakin besar, dan tentu perhitungan *cost reduction* untuk masing-masing kandidat menambah waktu proses. Dapat diperhatikan pada data masukan Facebook, walaupun jumlah *node graph* masukan etscience lebih besar dari *node* Facebook, tetapi waktu yang dibutuhkan untuk melakukan *summary* sangat jauh lebih besar pada *graph* masukan Facebook daripada Netscience.

4.6. Analisis Performa Graph Kompresi

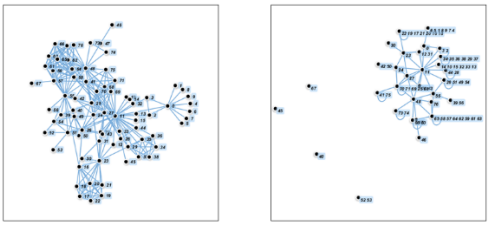
Pengujian dilakukan dengan menggunakan skenario yang telah dijelaskan sebelumnya menggunakan query terhadap tetangga node yang dipilih secara random pada masing-masing graph masukan. Untuk detail hasil yang didapat dapat dilihat pada lampiran.

Berdasarkan pengujian yang dilakukan, hasil yang diperoleh untuk *query* pada *node* random terdapat perbedaan *node* tetangga antara *graph* masukan dengan *graph* hasil pada setiap level kedalamannya, hal ini terjadi dikarenakan adanya informasi yang hilang atau *edge* yang terputus dari satu *supernode* ke tetangga *node* pembentuknya pada saat penggabungan dua node. Serta, ini diakibatkan penggabungan dua buah *node* yang awalnya jika dilakukan *query* dari suatu *node*. Contoh kasus misalkan dari *node* pada Gambar 0-4 yang pasangan *node* dengan digabungkan, maka pada hasil *query* pada *graph* masukan pada *depth* ke-1 adalah *node* sedangkan pada *graph* hasil adalah node pada hasil adalah *node* node tidak masuk ke dalam hasil query karena tidak terhubung oleh *edge*. *Edge* tersebut masuk ke dalam *correction (+)*. Jika dilakukan pembacaan *correction*, ini akan meminimalisasi perbedaan hasil *query* akibat penggabungan pasangan *node*.



Gambar 0-4 Query Error pada graph Kiri: Graph Masukan, Kanan: Graph hasil

Berikut pengamatan yang dilakukan pada *graph dataset* LesMiserables, dipilih karena ukuran yang kecil untuk memudahkan pengamatan.



Gambar 0-5Kompresi Graph LesMiserables

Dari kompresi diatas, kemudian dilakukan *query* terhadap graph masukan dan *summary* dengan ID *node* awal "0" yang memberikan hasil query pada Tabel 0-1.

Tabel 0-1 Hasil Query Node 0 Graph Les Miserables

Depth	Input	Summary		Reconstruct		error
		Node	Jumlah	Node	Jumlah	
1	10	[6, 11, 1, 5, 8, 3, 7, 9, 2, 4]	10	[11, 6, 5, 1, 8, 9, 7, 3, 4, 2]	10	0
2	33	[14, 44, 71, 55, 28, 37, 15, 23, 26, 51, 48, 64, 43, 24, 34, 70, 72, 69, 32, 33, 25, 13, 27, 29, 10, 58, 12, 35, 36, 68, 31, 38, 49]	32	[14, 44, 71, 55, 28, 37, 54, 15, 23, 26, 51, 48, 43, 24, 34, 72, 70, 69, 32, 33, 25, 13, 27, 29, 10, 35, 12, 36, 68, 38, 31, 49]	33	3
3	31	[42, 17, 45, 21, 50, 59, 61, 56, 54, 65, 57, 66, 16, 18, 47, 22, 41, 52, 19, 76, 30, 73, 39, 74, 75, 20, 53, 60, 62, 40, 63]	28	[42, 17, 21, 50, 59, 61, 56, 65, 57, 66, 64, 16, 47, 18, 22, 41, 19, 76, 30, 58, 73, 74, 39, 75, 20, 60, 62, 63]	31	7
4	2	[46, 67]	1	[46]	2	1
5	0	[]	0	[]	0	0

Pada Tabel 0-1, dapat dilihat bahwa terdapat perbedaan hasil query antara *graph* awal dengan *graph summary*, pada depth ke-1 tidak terjadi eror, sedangkan pada depth ke-2 terjadi eror pada 3 *node*, yaitu *node* dengan ID 64 dan 58 yang seharusnya ada pada *graph summary*, dan node dengan ID 54 seharusnya tidak ada pada *graph summary*. Untuk menelusuri apa penyebab eror tersebut maka dilakukan penelusuran terhadap *graph input*, *summary* dan *correction*.

Tabel 0-2 Correction

+(42,25)	+(26,27)	+(55,25)	+(53,51)	-(76,57)
+(43,26)	+(27,28)	+(31,27)	+(42,41)	-(57,66)
+(11,58)	+(29,27)	+(67,57)	+(31,30)	-(76,59)
+(39,52)	+(55,41)	+(33,27)	+(58,70)	-(76,61)
+(25,23)	+(11,64)	+(56,49)	+(41,24)	-(57,60)
+(55,16)	+(76,66)	+(58,27)	+(40,25)	-(71,24)
+(24,26)	+(72,26)	+(39,25)		-(70,48)
+(25,26)	+(41,57)	+(16,26)		-(11,54)
+(45,28)	+(50,49)	+(41,62)		
+(29,23)	+(75,48)	+(52,51)		

Tabel 0-1

Pertama lakukan penelusuran *node* 64, dari *node* 64 masuk ke dalam hasil *query* pada *depth* ke-3 dalam *graph summary*, ini terjadi karena penghapusan *edge* (11,64) dan masuk ke dalam *correction*, begitu juga untuk *node* dengan ID 58. Untuk *node* 54 yang masuk ke dalam hasil *query depth* ke-2, yang seharusnya masuk ke dalam *depth* ke-3, ini terjadi karena penggabungan antara *node* 54 dengan *node* 26, 51, dan 49. Dimana *node* 49 dan 26 merupakan tetangga pada *depth* ke-2 dari *node* 0. Untuk *node* eror dan *correction* terkait diberi warna merah pada tabel diatas.

Jika *correction* digunakan untuk pemrosesan *query* maka hasil query yang didapat akan semakin besar erornya, ini disebabkan oleh *node* yang digabung menjadi *supernode* akan ikut masuk ke dalam hasil *query* pada *depth* tertentu, sehingga *node* hasil query bertambah dengan *node* yang tidak seharusnya. Pada contoh kasus di atas adalah misalnya untuk *node* 64 kemudian mengikutsertakan correction +(11,64) maka node 65, 58, 57, 62, 59, 61, dan 63 akan ikut termasuk ke dalam *query* pada depth ke-2, yang seharusnya *node* tersebut masuk ke dalam hasil query depth ke-3.

Waktu eksekusi *query* ketetanggaan *graph* pada masing-masing kedalaman antara *graph* masukan dengan *graph summary*. Bergantung kepada berapa tetangga yang diproses oleh *query* tersebut.

4.7. Analisis Kompresi Lossless/Lossy

Dari pengamatan terhadap pengujian yang dilakukan berdasarkan skenario di atas, hasil dapat dilihat dari hasil *query* tetangga untuk setiap *depth* dalam *graph summary* memberikan hasil yang berbeda dengan *graph* input namun dengan yang eror relatif kecil, semakin dalam kedalaman yang digunakan eror yang didapat semakin besar. Maka *graph summary* yang dihasilkan bersifat lossy. sedangkan *graph* yang direkonstruksi kembali menggunakan *correction* memberikan hasil *query* yang sama dengan *graph* masukan *graph* rekonstruksi dari *summary* bersifat lossless.

5. PENUTUP

5.1. Kesimpulan

Berdasarkan hasil pengujian dan analisis yang telah dilakukan, maka diperoleh kesimpulan sebagai berikut:

1. Sebuah *graph* yang tidak berarah dapat dihitung dan dibentuk representasinya menggunakan algoritma *greedy* dengan menerapkan prinsip *Minimum Description Length* (MDL) dapat menghasilkan sebuah representasi yang dikompres dengan baik. Serta bisa dibentuk kembali menjadi menjadi *graph* yang sama dengan graph awal(*losless*) dengan menggunakan *Corrections*.
2. *Graph* masukan dengan jumlah *edge* mendekati jumlah *edge* maksimum yang dapat dibentuk oleh *graph* tersebut akan memberikan *compression ratio* yang semakin besar. Jika graph berbentuk graph sempurna, maka representasi yang dihasilkan berupa satu *node* dan satu *selfedge*, dengan compression ratio sebesar 0.99%.
3. Dari analis pengujian menunjukkan bahwa *graph* dapat dikompres dengan besar rata rata *compression rate* yang dihasilkan pada dataset yang digenerate menggunakan generator Random, Barabàsi-Albert, dan Dorogovtsev – Mendes berturut turut adalah 75.35%, 69,47%, dan 55.11%, serta pada graph dataset dengan *compression rate* pada masing-masing graph LesMiserables, Facebook, Netscience, Poewer, dan InternetRouters berturut-turut sebesar 75.23%, 78.02%, 67.61%, 58.38%, dan 69.08%
4. *Compression rate* terbesar terjadi pada graph masukan rand1000e yaitu sebesar 79,51%, dikarenakan graph tersebut memiliki edge sebesar 40345 yang membuat graph tersebut memiliki derajat rata-rata sebesar 80.69.
5. kemungkinan representasi yang dihasilkan menjadi ter-*cluster*. sebaliknya, nilai dengan derajat rata-ratanya yang besar akan menghasilkan representasi yang dengan kemungkinan ter-*cluster* semakin kecil.
6. Dengan adanya data yang *loss* atau *error* pada graph representasi, maka metode ini lebih cocok digunakan jika kebutuhan untuk kompresi *graph* yang bisa mentolerir *data loss*, contohnya: visualisi *graph*, *visual Mining*, dan lain lain.

5.2. Saran

Berikut saran yang dapat diberikan untuk pengembangan lebih lanjut terhadap *graph compression:*

1. Sistem selanjutnya agar dapat melakukan pengujian terhadap *dataset* maupun data generate *graph* yang berarah.
2. Penanganan terhadap data *loss* atau *error* pada saat penggabungan *edge*, sehingga *summary* yang dihasilkan bisa bersifat *lossless* dan efisien.
3. Pembacaan *correction* pada saat melakukan transversal atau *query* terhadap *graph summary*s sehingga didapat hasil *query* yang relevan dan sesuai dengan *graph* masukan.

Daftar Pustaka

[1] Bernhard Rieder. *NetGet*. http://snacourse.com/getnet. Diakses Pada November 2014.

[2] Gephi Dataset. https://wiki.gephi.org/index.php/Datasets. Diakses Pada April 2014.

[3] Paolo Boldi and Sebastiano Vigna, "*The Webgraph Framework I:Compression Techniques*.," *WWW '04 Proceedings of the 13th international conference on World Wide Web*, pp. 595–602, 2004.

[4] Ian Robinson, Jim Webber, and Emil Eifrem, *Graph Databases*.: O'Reilly Media, June 17, 2013.

[5] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis, "*A Fast Kernel-Based Multilevel Algorithm for Graph Clustering*.," *KDD*, pp. 629–634, 2005.

[6] Jorma Rissanen, "*An Introduction to The MDL Principle*".

[7] Peter Grunwald, *A Tutorial Introduction to the Minimum Description Length Principle*., 2004.

[8] Rio Eduardo Bangkit Gideon, *Transformasi Relational Database ke Graph Database pada Website Sosial Network*., 2013.

[9] Ruohonen Keijo, *Graph Theory*.,2013.

[10] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava, "*Graph Summarization with Bounded Error*," *SiGMOD*, 2008.

[11] Stefan Balev, Julien Baudry, Antoine Dutot, Yoann Pigné, and Guilhelm Savin. *GraphStream - A Dynamic Graph Library*. http://graphstream-project.org/