**Implementasi VoG (Vocabulary based summarization of Graph) Pada Web Graph**

***Implementation of VoG (Vocabulary based summarization of Graph) on the Web Graph***

**Tugas Akhir**

**Kelompok Keahlian : SIDE**

**Satrio Adityo Hartomo**

**1103120029**

****

**Program Studi Teknik Informatika**

**Fakultas Informatika**

**Universitas Telkom**

**Bandung**

**2015**

# LEMBAR PERNYATAAN

Dengan ini saya menyatakan bahwa tugas akhir dengan judul **“Implementasi VoG (Vocabulary based summarization of Graph) pada Web Graph”** dan seluruh isinya benar-benar merupakan karya saya sendiri dan saya tidak melakukan penjiplakan atau pengutipan dengan cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan. Atas pernyataan ini, saya siap menanggung resiko/sanksi yang dijatuhkan kepada saya apabila kemudian ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya ini, atau ada klaim dari pihak lain terhadap karya keaslian saya ini.

Bandung, Desember 2015

Yang membuat pernyataan,

Satrio Adityo Hartomo

# LEMBAR PENGESAHAN

**Implementasi VoG (Vocabulary based summarization of Graph) pada Web Graph**

***Implementation of VoG (Vocabulary based summarization of Graph) on the Web Graph***

**Satrio Adityo Hartomo**

**1103120029**

Telah disetujui dan disahkan sebagai tugas akhir

Program Studi Teknik Informatika

Fakultas Informatika Universitas Telkom

Bandung, Desember 2015

Menyetujui

|  |  |  |
| --- | --- | --- |
| Pembimbing I |  | Pembimbing II |
|  |  |  |
|  |  |  |
| Kemas Rahmat Saleh W, S.T., M.Eng. NIP. 06830335-1 |  | Siti Sa'adah, S.T., M.T.  NIP. 13861143-1 |

|  |
| --- |
| Ketua Program Studi  Sarjana Teknik Informatika |
|  |
|  |
| Kemas Rahmat Saleh W, S.T., M.Eng. NIP. 06830335-1 |

# Abstrak

# Abstract

# Lembar Persembahan

# Kata Pengantar

# Daftar Isi

# Daftar Gambar

# Daftar Tabel

# Daftar Istilah

# Pendahuluan

## Latar Belakang

Dengan adanya teknik penyimpanan data yang baru yaitu *graph database* [1], cara untuk menyimpan data tidak lagi terbatas pada *relational database* yang struktural. Data yang mempunyai perbedaan atribut (semi terstruktur) dapat disimpan ke *database* dengan representasi *graph* [1], seperti Facebook dan Twitter yang juga menggunakan *graph database* untuk menyimpan data-datanya [2] [3]. *Website* yang lain pun memiliki data yang unik, berjumlah banyak dan semakin bertambah hingga saat ini [4]. *Website* yang dulunya hanya menampilkan data statis kini telah berevolusi menjadi *website* yang dapat menampilkan data secara dinamis dan dapat direpresentasikan sebagai *graph* (*web graph*) [1].

Pada referensi [4] disebutkan bahwa pada Juli 2000 terdapat dua milyar *web*, dan terus meningkat jumlahnya hingga empat milyar *web* pada tahun 2001. *Web graph* yang besar dengan ribuan *node* dan *edge* [5], akan sulit untuk diidentifikasi bagaimana bentuk strukturnya, apakah *random* atau tidak. Dengan ukuran yang besar tersebut akan sulit juga untuk memperoleh informasi. Oleh karena itu diperlukan metode untuk meringkas *web graph* agar ukurannya menjadi lebih kecil dari *web graph* aslinya, sehingga dapat dilakukan pengidentifikasian struktur dan analisis untuk mendapatkan informasi pada hasil ringkasan *web graph*.

VoG (Vocabulary based summarization of Graph) adalah metode untuk meringkas graph yang besar secara efisien [6]. Dalam metode VoG akan dilakukan graph decomposition menggunakan algoritma Slashburn. Algoritma Slashburn akan menghasilkan sekumpulan subgraph yang mana subgraph yang dihasilkan akan dilakukan proses identifikasi struktur. Dalam tugas akhir ini akan dilakukan implementasi web graph *summarization* menggunakan metode VoG untuk meringkas web graph menjadi subgraph-subgraph yang lebih kecil ukuran atau jumlah nodenya dan mendapatkan informasi dari subgraph-subgraph yang dihasilkan.

## Perumusan Masalah

Dari latar belakang yang telah diuraikan, permasalahan yang akan dibahas dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana mengimplementasikan VoG untuk *summarize* web graph?
2. Bagaimana pengaruh jumlah *node* dalam *Giant Connected Component (GCC)* pada struktur yang dihasilkan oleh VoG?
3. Bagaimana menganalisis struktur subgraph yang dihasilkan oleh VoG untuk memperoleh minimum satu kalimat informasi?

## Tujuan

Adapun tujuan dalam tugas akhir ini adalah sebagai berikut:

1. Mengimplementasikan metode VoG untuk men-*summarize* web graph.
2. Menganalisis pengaruh jumlah node dalam *Giant Connected Component (GCC)* pada struktur yang dihasilkan oleh VoG.
3. Menganalisis struktur subgraph yang dihasilkan oleh VoG untuk memperoleh minimum satu kalimat informasi.

## Batasan Masalah

Mengenai batasan permasalahan dalam tugas akhir ini adalah sebagai berikut:

1. Struktur yang dapat diidentifikasi oleh metode VoG dalam tugas akhir ini terdiri dari *Perfect Clique, Perfect Star, Perfect Bipartite, Perfect Chain, Near Clique, Near Star, Near Bipartite,* dan *Near Chain.*
2. Metode VoG hanya dapat melakukan identifikasi struktur setelah memiliki keluaran dari algoritma *graph decomposition* yang dalam tugas akhir ini adalah algoritma Slashburn.
3. Dataset yang digunakan didapat dari hasil crawling web menggunakan aplikasi buatan sendiri.
4. Dua *node* yang terhubung hanya memiliki satu *edge*.
5. Untuk pengolahan graph menggunakan bantuan library GraphStream.

## Metodologi Penyelesaian Masalah

Metode penyelesaian masalah yang diterapkan dalam tugas akhir ini adalah sebagai berikut:

1. Identifikasi permasalahan

Pada tahap ini dilakukan studi literatur dan diskusi untuk mengidentifikasi permasalahan yang akan dibahas pada tugas akhir ini. Identifikasi meliputi observasi terkait fenomena yang terjadi di bidang graph database, metode yang digunakan untuk solusi permasalahan, dan batasan masalah yang ada.

1. Studi literatur

Pada tahap ini dilakukan pengumpulan dan pemahaman terkait materi-materi yang dibutuhkan untuk menyelesaikan permasalahan yang tertera di perumusan masalah. Referensi yang dicari antara lain materi mengenai metode VoG, metode *graph decomposition* untuk menghasilkan graph summarization.

1. Perancangan dan pembuatan sistem

Pada tahap ini dilakukan pemodelan dataset ke dalam bentuk graph. Kemudian membuat rancangan sistem untuk memberikan solusi dari permasalahan. Dari rancangan sistem yang sudah dibuat dilakukan pembuatan sistem dengan menggunakan bahasa pemrograman Java.

1. Pengujian dan analisis

Pada tahap ini dilakukan pengujian terhadap dataset yang sudah dimodelkan dalam bentuk graph. Pengujian dilakukan sesuai dengan skenario yang ada pada bagian strategi pengujian. Kemudian dilakukan analisis terhadap subgraph-subgraph hasil ringkasan graph.

1. Penyusunan laporan

Pada tahap ini dilakukan dokumentasi dan pelaporan hasil sesuai dengan aturan dan sistematika penulisan yang ditetapkan institusi.

## Sistematika Penulisan

Sistematika penulisan buku tugas akhir ini adalah sebagai berikut:

* 1. Bab I Pendahuluan

Bab ini berisi latar belakang, perumusan masalah, tujuan, batasan masalah, metodologi penyelesaian masalah dan sistematika penulisan buku tugas akhir.

* 1. Bab II Landasan Teori

Bab ini berisi landasan teori yang menjadi dasar dilakukannya penelitian pada tugas akhir ini.

* 1. Bab III Perancangan Sistem

Bab ini berisi perancangan sistem yang akan diimplementasikan pada tugas akhir ini.

* 1. Bab IV Pengujian dan Analisis

Bab ini berisi pengujian dan analisis hasil dari sistem yang telah dibuat.

* 1. Bab V Kesimpulan dan Saran

Bab ini berisi kesimpulan yang didapat dari permasalahan, pengujian dan analisis hasil, serta saran untuk penelitian selanjutnya.

# Dasar Teori

## 2.1 Teori Graph

*Graph*, secara definisi adalah sekumpulan *nodes* () dan *edges* (). Sekumpulan *nodes* bisa saja tidak terbatas atau bisa disebut ***infinite graph***, sedangkan yang sekumpulan *nodes*nya terbatas disebut ***finite graph***. Dalam tugas akhir ini hanya akan fokus pada *finite graph*. *Graph* dapat dinotasikan sebagai . adalah sekumpulan *nodes* dan adalah sekumpulan *edges* yang menghubungkan *nodes* [7].

|  |  |  |  |
| --- | --- | --- | --- |
| Gambar 2‑1: Directed Graph | | Gambar 2‑2: Undirected Graph | |
| Gambar 2‑3: Simple Graph | Gambar 2‑4: Multigraph | | Gambar 2‑5: Pseudograph |
| Gambar 2‑6: Graph Berlabel | | Gambar 2‑7: Graph Tak Berlabel | |

Perbedaan jenis *graph* dapat dilihat berdasarkan ada atau tidaknya arah *edge*, *graph* jenis ini terdiri dari ***directed graph*** (*graph* yang mempunyai *edge* berarah) dan ***undirected graph* (***graph* yang mempunyai *edge* tidak berarah). Berdasarkan *edge* yang menghubungkan *nodes* terdiri dari ***simple graph*** (tidak memiliki *loop* dan *multiple edge*), ***multigraph*** (tidak memiliki *loop edge* tetapi memiliki *multiple edge*) dan ***pseudograph*** (memiliki *loop edge*). Berdasarkan ada atau tidaknya label terdiri dari ***graph* berlabel** dan ***graph* tak berlabel**. Contoh jenis-jenis *graph* dapat dilihat pada gambar 2-1 sampai 2-7 di bawah ini [7].

Sekumpulan *nodes* dan *edges* yang ada di *graph* dapat digunakan untuk memodelkan permasalahan yang ada pada dunia nyata, sebagai contoh *graph* jaringan komputer. Namun, dalam kondisi tertentu untuk menyelesaikan permasalahan yang ada, tidak seluruh *nodes* digunakan. Sehingga yang digunakan untuk menyelesaikan masalah adalah bagian kecil dari *graph*, atau bisa disebut dengan *subgraph*. Secara definisi, *subgraph* dari *graph* adalah *graph* dimana dan dan [7].

|  |  |
| --- | --- |
| Gambar 2‑8: Graph G | Gambar 2‑9: Graph H (subgraph G) |

Cara merepresentasikan *graph* ada beberapa cara, diantaranya yaitu menggunakan *Adjacency Matrix* dan *Adjacency List*. Berikut contoh cara merepresentasikan graph [7]:

Dari gambar 2-8 dapat direpresentasikan *adjacency list* seperti gambar 2-10 di bawah ini :



Gambar 2‑10 Representasi graph - adjacency list [8]

Atau sebagai *adjacency matrix* seperti di bawah ini :

Angka 1 merepresentasikan adanya *edge* antar*nodes* [7].

## 2.2 Graph Database

*Graph Database Management System* adalah *database management system* yang memiliki metode CRUD (*Create*, *Read*, *Update*, dan *Delete*) untuk memaparkan model data *graph*. Properti model *graph* dapat dideskripsikan sebagai berikut :

* Terdiri dari *nodes* dan *relationship/edges*.
* Setiap *nodes* terdiri dari pasangan *key-value*.
* *Edge* mempunyai nama dan arah, jika tidak memiliki arah disebut *undirected graph.*
* *Edge* dapat juga mempunyai properti.

Dengan abstraksi sederhana dari *nodes* dan *edges* kedalam struktur yang saling terhubung, *graph database* memungkinkan untuk dibangunnya model yang mirip dengan permasalahan yang ada di dunia nyata [1].



Gambar 2‑11 Model graph database

## 2.3 Web Graph

*Webpages* dan *hyperlinks* di *World Wide Web* dapat direpresentasikan sebagai *nodes* dan *edge* di *directed graph*. Saat ini, *graph* tersebut memiliki milyaran *nodes* dan *edge* dan bertambah banyak seiring berjalannya waktu. Ada beberapa alasan untuk mempelajari *web graph*, diantaranya adalah bagaimana algoritma yang bekerja pada *web graph*, bagaimana *web search*nya, dan bagaimana pengklasifikasiannya [9].

Tidak hanya itu, ada alasan lain untuk mengembangkan pemahaman dari *web graph* ini, diantaranya dapat :

1. Mendesain strategi *crawling* pada web.
2. Memahami sosiologi pembuatan konten pada web.
3. Menganalisis *behavior* dari algoritma web.
4. Memprediksi evolusi struktur web.
5. Memprediksi munculnya fenomena baru di web [10].

Berikut contoh *web graph*:



Gambar 2‑12: Contoh Web Graph [11]

## 2.4 Graph Compression

Jika dalam suatu *graph* dapat ditemukan *good community*, maka *graph* tersebut dapat dikompres yang mana dapat membantu untuk menyusutkan ukuran data [12]. Rincian proses *graph compression* dalam tugas akhir ini dijelaskan pada bagian 2.4.1 dan 2.4.2.

### Graph Shattering

Untuk menyelesaikan permasalahan dalam *graph compression*, diperlukan proses untuk mengurai *graph* dengan cara mengeksploitasi *hubs* (*node* dengan degree paling tinggi). Dengan hasil eksploitasi *hubs* didapatkan *alternative community* yang berbeda dari *clique community* [12].

Dari referensi [12] disebutkan bahwa berdasarkan observasi, *real world graph* dapat dengan mudah dipecah dengan menghapus *hub node* dari *graph*. Dengan penghapusan *hub node*, *graph* terurai menjadi banyak *subgraph-subgraph*.

### 2.4.2 Slash and Burn

Baris 1 dan 2 dari algoritma 1 menghapus *nodes* dengan *degree* paling tinggi dan *edge* yang berasosiasi, dengan demikian *graph* terurai dalam tiga kelompok:

* -hubset: *nodes* dengan *degree* paling tinggi.
* GCC: *node* yang dimiliki oleh *Giant Conncected Component* dari .
* Spokes: *node* yang dimiliki oleh *non-giant connected component* dari [12].



Gambar 2‑13: Algoritma Slashburn



Gambar 2‑14: (a) adalah graph G, (b) adalah graph G' setelah satu kali eksekusi slashburn [6]

## 2.5 VoG

*Vocabulary based summarization of Graph* (VoG) merupakan metode yang efektif dan efisien untuk meringkas graph yang besar. *Vocabulary* (Ω) adalah sekumpulan struktur *graph* yang terdiri dari struktur *full-clique, near-clique, full-bipartite, near-bipartite, perfect-star, near-star, perfect-chains* dan *near-chains*. Masih ada struktur *graph* yang lain, akan tetapi dari referensi [6] disebutkan bahwa struktur *graph* yang sering muncul dalam *real world* *graph* adalah struktur yang ada pada *Vocabulary* tersebut. Oleh karena itu, dalam tugas akhir ini dibangun sistem yang dapat mengidentifikasi struktur dalam *vocabulary*. Penjelasan mengenai struktur *graph* dapat dilihat di bagian 2.5.1 - 2.5.4 [6].

*Graph* yang ada di dunia sangatlah banyak dan besar, orang-orang sulit untuk memahami dengan mudah *graph* yang besar dan mungkin berantakan strukturnya. Dengan adanya VoG, *graph* dapat diringkas dan orang-orang dapat lebih mudah memahaminya [6].

### 2.5.1 Full Clique

*Full Clique* adalah *complete subgraph* dari suatu *graph* atau semua *node* didalam *subgraph* saling terhubung dengan *node* yang lain di dalam *subgraph* yang sama. Misalnya diberikan *graph* seperti gambar 2-8. *Subgraph* dengan struktur *Full clique* yang ada beberapa diantarnya adalah [13]:



Gambar 2‑15: Beberapa contoh full-clique subgraph

### 2.5.2 Near Clique

*Near Clique* adalah *subgraph* yang hampir *complete* dari suatu *graph*. Misalnya diberikan *graph* seperti gambar 2-8. *Near clique* yang ada beberapa diantarnya adalah [6]:



Gambar 2‑16: Beberapa contoh near-clique subgraph

### 2.5.3 Full Bipartite

*Full bipartite* adalah *non-empty*, *non-intersecting* sekumpulan *nodes* dan yang mana semua *node* di terhubung ke semua *node* di , namun tidak ada *edge* dalam kumpulan *nodes* atau . Contoh *full bipartite* [6]:



Gambar 2‑17: full-bipartite subgraph

### 2.5.4 Near Bipartite

*Near bipartite* adalah non-empty, *non-intersecting* *nodes* dan yang mana **hampir** semua *node* di terhubung ke semua *node* di , namun **ada** *node* di yang tidak terhubung ke salah satu *node* di , namun tidak ada *edge* dalam kumpulan *nodes* atau . Contoh *near bipartite* [6]:



Gambar 2‑18: near-bipartite subgraph

### 2.5.5 Star

*Star* adalah kasus spesifik dari *bipartite graph*, yang mana hanya ada *single node* terhubung ke semua *node* yang mana paling tidak terdiri minimal 2 *nodes* [6].



Gambar 2‑19: Star subgraph

### 2.5.6 Chain

*Chain* adalah *graph* yang setiap *node*nya memiliki edge ke *node* selanjutnya [6].



Gambar 2‑20: Chain subgraph

### 2.5.7 Algoritma VoG

Pseudocode dari VoG adalah sebagai berikut :



Gambar 2‑21: Pseudocode Algoritma VoG [6].

# Perancangan Sistem

## Deskripsi Sistem

Sistem yang dibangun dalam tugas akhir ini adalah sistem yang dapat meringkas *graph* yang besar menjadi sekumpulan *subgraph* yang berukuran lebih kecil. Dari masing-masing *subgraph* akan diidentifikasi struktur dan dianalisis informasinya. Dataset yang digunakan dalam pengujian sistem adalah dataset yang didapat melalui *crawling* *web* quora.com. Data yang di-*crawl* adalah data *username* dalam suatu *link question*. Dari data yang sudah diperoleh kemudian direpresentasikan sebagai *graph*, yang mana *node* merepresentasikan *username*, dan *edge* merepresentasikan dua *username* menjawab pertanyaan di *link* yang sama. Berikut ini adalah *flow chart* dari sistem yang dibangun:



Gambar 3‑1:Flowchart sistem secara umum

Gambar 3-1 adalah gambaran secara umum proses-proses yang ada dalam sistem yang dibangun. Proses yang lebih rinci mengenai proses *slashburn* dan *subgraph labeling*, dapat dilihat dalam gambar 3-2 dan 3-3 di bawah ini:



Gambar 3‑2: Flowchart proses slashburn untuk decomposing graph menjadi sekumpulan subgraph



Gambar 3‑3: Flowchart subgraph labeling untuk mengidentifikasi struktur subgraph

## Perancangan Sistem

Alur dari sistem yang dibangun dalam tugas akhir ini terdiri dari tiga tahap:

* 1. *Crawling* dataset dari *web* quora.com, dataset disimpan dalam bentuk file dengan format .txt.
  2. Representasi dataset ke bentuk *graph* dengan bantuan *library* GraphStream Java.
  3. Penerapan VoG, yaitu *graph compression* untuk menghasilkan *subgraph-subgraph* dari *graph input*, lalu *subgraph labeling* untuk mengidentifikasi struktur masing-masing *subgraph*.

### Web Crawling

Untuk mendapatkan dataset yang digunakan dalam tugas akhir ini, dilakukan *web crawling* pada web quora.com. Data yang diambil adalah *username* dan *link question*. Berikut ini contoh *web* yang di-*crawl*:



Gambar 3‑4: link question dari quora.com

Setelah *web crawler* selesai melakukan *crawling*, akan diperoleh dataset. Berikut ini adalah contoh dataset yang didapatkan:



Gambar 3‑5: satu link question hasil crawl untuk dijadikan dataset

### Representasi Graph dari Dataset

Dari dataset yang sudah didapatkan, data setiap baris akan dipecah untuk mendapatkan *url*, dan *username*. Dari *url* dan *username* tersebut akan dibuat representasi *graph* menggunakan bantuan *library* GraphStream Java. Berikut ini adalah contoh visualisasi *graph* di aplikasi:



Gambar 3‑6: Visualisasi graph P dari dataset

### Implementasi VoG

Untuk proses implementasi VoG, harus ditentukan terlebih dahulu berapa jumlah *node* maksimal dalam GCC (). Jumlah *node* maksimal ini akan menjadi batas untuk iterasi proses *graph compression* menggunakan algoritma Slashburn.

#### 3.2.3.1 Proses Graph Compression menggunakan Algoritma Slashburn

Algoritma Slashburn akan berhenti dieksekusi jika . Ketika proses slashburn berjalan, *graph* diproses secara iteratif dengan batas . Graph akan dicari *hub node* di dalam GCC untuk dihapus. Penghapusan *hub node* ini disertai dengan *edge*-nya dan menghasilkan *subgraph*. Jika proses penghapusan *node* ini menghasilkan satelit *node* yang *isolated*, maka *hub node* akan dibuat kembali dan satelit *node* akan dihubungkan dengan *hub node*. Hal ini dilakukan agar tidak terbentuk *subgraph* dengan satu *node*. Setelah proses *graph compression* selesai, akan dihasilkan *subgraph-subgraph*. Berikut ini adalah *subgraph* yang terbentuk dari pada gambar 3-6 dengan :



Gambar 3‑7: Visualisasi subgraph dari Graph P setelah proses slashburn selesai

#### 3.2.3.2 Proses Subgraph Labeling

Setelah proses *graph compression* menggunakan algoritma slashburn selesai, didapatkan *subgraph*. Dari *subgraph-subgraph* tersebut akan diidentifikasi strukturnya. Untuk proses identifikasi struktur, yang pertama dilakukan adalah mendapatkan jumlah *node* dalam *subgraph* dan *adjacency matrix*-nya. Setelah jumlah *node* didapat, maka dibuatlah empat model yaitu *clique, star, chain,* dan *bipartite* model. Model-model inilah yang akan digunakan untuk proses mendapatkan *error matrix*. Berikut ini adalah contoh proses untuk mendapatkan *error matrix*:

(1)

Contoh operasi matematika (1) diatas adalah operasi *adjacency matrix subgraph* XOR dengan *matrix* model clique, yang menghasilkan *error matrix* untuk model *clique*.

(2)

Contoh operasi matematika (2) diatas adalah operasi *adjacency matrix subgraph* XOR dengan *matrix* model star, yang menghasilkan *error matrix* untuk model *star*.

Dari dua contoh *error matrix* diatas, *error matrix* kemudian di-*encode* menjadi *error score* untuk masing-masing model. Dalam kasus di atas, *error score* untuk *error matrix* model *clique* = 2. Sedangkan *error score* untuk *error matrix star* adalah 0. Setelah didapatkan *error score* untuk masing-masing model, akan dipilih *error score* yang paling kecil nilainya, sehingga *subgraph* akan diidentifikasi sebagai **Subgraph dengan Struktur Star**.

# Pengujian dan Analisis

Sistem dibuat dengan bahasa pemrograman Java dengan bantuan library GraphStream untuk mengelola graph. Dataset yang digunakan dapat berbentuk file .txt dalam hal ini adalah dataset hasil crawling web quora.com. Dapat juga berbentuk file .dgs hasil penyimpanan graph yang mana file .dgs tersebut dapat dimuat kembali ke sistem.

## 4.1 Tujuan Pengujian

Dari pengujian yang dilakukan, tujuan yang ingin dicapai adalah sebagai berikut:

1. Menganalisis subgraph yang dihasilkan oleh algoritma graph decomposition apakah subgraph yang dihasilkan sesuai dengan vocabulary dalam VoG.
2. Menganalisis pengaruh jumlah nodes dalam GCC terhadap subgraph yang dihasilkan.
3. Menganalisis informasi yang terkandung dalam struktur subgraph.

## Skenario Pengujian

Dalam pengujian sistem ini, terdapat beberapa scenario pengujian. Skenario yang pertama adalah pengujian algoritma graph decomposition (Slashburn) untuk menghasilkan subgraph yang ada pada vocabulary. Pengujian graph decomposition menggunakan contoh dataset yang dibuat sendiri. Terdapat 4 dataset yang digunakan. Berikut adalah tampilan graph dari 4 dataset yang digunakan untuk menguji algoritma graph decomposition:

[gambar dataset clique]

[gambar dataset star]

[gambar dataset chain]

[gambar dataset bipartite]

# Daftar Pustaka

|  |  |
| --- | --- |
| [1] | I. Robinson, J. Webber dan E. Eifrem, Graph Databases, Sebastopol: O'Reilly Media, Inc, 2013. |
| [2] | P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang dan R. Zadeh, “WTF: The Who to Follow Service at Twitter,” dalam *International World Wide Web Conference Committee (IW3C2)*, Rio de Janeiro, 2013. |
| [3] | M. Curtiss, I. Becker, T. Bosman, S. Doroshenko, L. Grijincu, T. Jackson, S. Kunnatur, S. Lassen, P. Pronin, S. Sankar, G. Shen, G. Woss, C. Yang dan N. Zhang, “Unicorn: A System for Searching the Social Graph,” dalam *Proceedings of the VLDB Endowment*, Trento, 2013. |
| [4] | B. H. Murray dan A. Moore, “Sizing The Internet,” Cyveillance, 2000. |
| [5] | N. Shervashidze, S. Vishwanathan, T. H. Petri, K. Mehlhorn dan K. M. Borgwardt, “Efficient graphlet kernels for large graph comparison,” dalam *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Florida, 2009. |
| [6] | D. Koutra, U. Kang, J. Vreeken dan C. Faloutsos, “VoG : Summarizing and Understanding Large Graphs,” dalam *Proceedings of the 2014 SIAM International Conference on Data Mining*, 2014. |
| [7] | K. H. Rosen, Discrete Mathematics and Its Applications, New York: The McGraw-Hill Companies, Inc., 2012. |
| [8] | M. A. Kolosovskiy, Data structure for representing a graph:, Altai State Technical University, Russia. |
| [9] | R. Kumar, P. Raghavan, S. Rajagopalan, D. Shivakumar, A. S. Thomkins dan E. Upfal, “The Web as a Graph,” dalam *ACM*, Texas, 2000. |
| [10] | A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Thomkins dan J. Wiener, “Graph Structure in the Web,” *Computer Nerworks Elsevier,* vol. 33, pp. 309-320, 2000. |
| [11] | Dhia, “Finding the Patterns in a Mysterious New DGA,” [Online]. Available: https://labs.opendns.com/2013/10/24/mysterious-dga-lets-investigate-sgraph/. [Diakses 18 March 2015]. |
| [12] | Y. Lim, U. Kang dan C. Faloutsos, “SlashBurn: Graph Compression and Mining beyond Caveman Communities,” dalam *IEEE Transaction On Knowledge and Data Engineering*, 2014. |
| [13] | R. Balakhisnan dan K. Ranganathan, A Textbook of Graph Theory, Springer Science & Business Media, 2012. |