Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN: 2355-7699
Vol. 13, No. 2, April 2026, hlm. 251- 260 e-ISSN: 2528-6579

SIMPLIFIKASI GRAF PEMANGGILAN FUNGSI: PENDEKATAN *COMMUNITY DETECTION* UNTUK MEMPERMUDAH PEMAHAMAN STRUKTUR KODE

Risa Tioria Marlini Purba[*1], Ari Sandy Putra Ari Tonang[2], Abdulah Karim[3], Gusti Ayu Putri Saptawati Soekidjo[4], Koyimatu Muhamad[5], Fitra Arifiansyah[6]

[1,2,3,4,5,6]Institut Teknologi Bandung, Bandung
Email: [1]23523302@mahasiswa.itb.ac.id, [2]23523315@mahasiswa.itb.ac.id, [3]323523314@mahasiswa.itb.ac.id, [4]putri@itb.ac.id, [5]koyimatu@itb.ac.id, [6]fitra@itb.ac.id
[*]Penulis Korespondensi

(Naskah masuk: 17 Desember 2024, diterima untuk diterbitkan: 24 Agustus 2025)

Abstrak

Dalam pengembangan perangkat lunak skala besar, pemahaman terhadap struktur kode sangat penting untuk menganalisis interaksi antar-fungsi dalam kode sumber. Graf pemanggilan fungsi *(function call graph)* merupakan kakas yang efektif untuk memetakan hubungan antar-fungsi, yang membantu pengembang dalam menelusuri jalur eksekusi dan memahami pola struktur kode modular yang kompleks. Namun, pada kode sumber yang rumit, graf pemanggilan fungsi sering kali menjadi sangat besar dan sulit diinterpretasi karena banyaknya *node* dan *edge* yang terlibat. Untuk mengatasi masalah ini, teknik simplifikasi graf melalui *community detection* diterapkan sebagai solusi untuk mengelompokkan fungsi-fungsi yang saling terkait dalam cluster, sehingga menghasilkan visualisasi yang lebih terstruktur dan mudah dipahami. Penelitian ini bertujuan untuk mengembangkan kakas berbasis Python yang mampu menyederhanakan graf pemanggilan fungsi menggunakan algoritma Girvan-Newman. Kakas ini memanfaatkan pustaka *networkx* untuk membentuk graf dan menerapkan deteksi komunitas, *ast* untuk parsing kode, serta *matplotlib* dan *streamlit* untuk visualisasi dan interaksi pengguna. Hasil eksperimen pada 10 program dengan ukuran 10-85 baris kode menunjukkan bahwa metode community detection mampu mereduksi jumlah node dan edge dalam graf pemanggilan fungsi hingga 60%, dengan skor modularitas tertinggi 0.6605. Evaluasi dengan 25 pengembang perangkat lunak menunjukkan tingkat kepuasan 80% dalam hal kemudahan penggunaan dan peningkatan produktivitas analisis kode.

Kata kunci: *graf pemanggilan fungsi, community detection, algoritma Girvan-Newman, peyederhanaan graf, Python.*

SIMPLIFICATION OF FUNCTION INVOCATION GRAPHS: A COMMUNITY DETECTION APPROACH TO MAKE CODE STRUCTURE EASIER TO UNDERSTAND

Abstract

*In large-scale software development, understanding the code structure is crucial for analyzing the interactions between functions in the sourc*e code*. A function call graph is an effective tool for mapping the relationships between functions, assisting developers in tracing execution paths and understanding object-oriented complex code structures. However, in complex sourc*e code*, the function call graph often becomes very large and complicated to interpret due to the many nodes and edges involved. To address this issue, graph simplification techniques, such as community detection, are applied as a solution to group related functions into clusters, thereby producing a more structured and easier-to-understand visualization. This study aims to develop a Python-based tool that simplifies function call graphs using the Girvan-Newman algorithm. The tool utilizes the networkx library to construct graphs and apply community detection, ast for code parsing, and matplotlib and streamlit for visualization and user interaction. The results of experiments on 10 programs, ranging in size from 10 to 85 LOC, showed that the community detection method was able to reduce the number of nodes and edges in the function invocation graph by up to 60%, achieving the highest modularity score of 0.6605. An evaluation of 25 software developers revealed an 80% satisfaction rate in terms of ease of use and increased productivity in code analysis.*

Keywords: *function call graph, community detection, Girvan-Newman algorithm, graph simplification, software development, Python.*

1. PENDAHULUAN

1.1. Latar Belakang

Dalam pengembangan perangkat lunak modern, pemahaman terhadap struktur dan interaksi antar-komponen dalam kode sumber menjadi sangat penting, terutama dalam proyek skala besar yang mencakup ribuan fungsi saling bergantung (Alanazi et al., 2021a). Salah satu pendekatan yang banyak digunakan untuk menganalisis hubungan dan ketergantungan antar-fungsi dalam perangkat lunak adalah melalui graf pemanggilan fungsi (*function call graph*) (Hazem & Gustafsson, 2023). Graf pemanggilan fungsi memberikan representasi visual dari hubungan pemanggilan antara fungsi-fungsi dalam kode sumber, memudahkan pengembang untuk memahami alur eksekusi, mengidentifikasi ketergantungan, dan mendeteksi masalah dalam struktur perangkat lunak (Alanazi et al., 2021a; Hazem & Gustafsson, 2023).

Dengan graf pemanggilan fungsi, pengembang dapat menelusuri jalur pemanggilan antar-fungsi dan mengidentifikasi struktur modular dalam perangkat lunak, sehingga memperoleh pemahaman yang lebih baik mengenai relasi antar-komponen kode (Hazem & Gustafsson, 2023; Shi et al., 2024). Namun, tantangan utama dalam penerapannya adalah kompleksitas graf yang dihasilkan, terutama dalam kode sumber yang besar. Pada proyek perangkat lunak skala besar (Alanazi et al., 2021a; Xia et al., 2018a), graf pemanggilan fungsi dapat mencakup ribuan *node* dan *edge* yang saling berhubungan, menghasilkan visualisasi yang padat dan sulit diinterpretasi. Akibatnya, graf pemanggilan fungsi yang terlalu kompleks dapat memperlambat proses *debugging* dan analisis kode (Shi et al., 2024). (Alanazi et al., 2021a; Gousios & Proksch, 2024; Xia et al., 2018a).

1.2. *Community Detection* pada Simplifikasi Graf

Dalam mengatasi tantangan kompleksitas dalam graf pemanggilan fungsi, teknik simplifikasi graf menjadi sangat relevan (Aghamohammadi et al., 2020). Simplifikasi graf bertujuan untuk mengurangi jumlah *node* dan *edge* dalam graf tanpa menghilangkan informasi penting yang diperlukan untuk analisis (Aghamohammadi et al., 2020). Salah satu metode yang digunakan dalam simplifikasi graf adalah *community detection* (J. Li et al., 2024), yang memungkinkan pengelompokan fungsi-fungsi yang saling terkait dalam komunitas atau klaster. Penggunaan *community detection* dapat menghasilkan graf yang lebih terstruktur dan mudah diinterpretasi, membantu pengembang dalam mengidentifikasi modul atau subsistem utama dalam kode sumber (Gousios & Proksch, 2024; J. Li et al., 2024).

Community detection adalah teknik yang banyak digunakan dalam analisis jaringan sosial, biologi, dan kini semakin banyak diadaptasi dalam konteks perangkat lunak (J. Li et al., 2024; Zahiri et al., 2023a). Sementara itu, mengelompokkan *node* berdasarkan *betweenness centrality* pada *edge*, terbukti efektif dalam mendeteksi komunitas pada graf yang besar dan kompleks (Xia et al., 2018a; Zahiri et al., 2023a). Alternatif lainnya, seperti algoritma Louvain, juga menawarkan metode yang efisien dalam mengidentifikasi struktur komunitas pada jaringan berskala besar dengan tingkat modularitas yang tinggi (Abbas & Nawaf, 2020a; Alhajjar, 2022; Zhang et al., 2021).

Zahiri, Mohammadzadeh and Harifi (2023a) memperkenalkan algoritma yang menggunakan *edge betweenness centrality* untuk mengidentifikasi komunitas dalam graf. Algoritma ini membagi graf berdasarkan keterkaitan antar-*node*, sehingga fungsi-fungsi yang sering saling memanggil cenderung berada dalam komunitas yang sama (Newman, 2006). (Fortunato & Hric, 2016) mengulas berbagai algoritma *community detection* lainnya, seperti Zhang et al (2021), yang menawarkan pendekatan efisien dalam mengidentifikasi struktur komunitas dalam graf berskala besar (Fortunato & Hric, 2016)(Fortunato, 2010).

Dalam penelitian terbaru, teknik *community detection* telah diadaptasi untuk menyederhanakan analisis kode perangkat lunak, Alhajjar (Alhajjar, 2022) dalam Network Science menunjukkan bahwa deteksi komunitas dapat digunakan untuk mengidentifikasi subsistem atau modul dalam perangkat lunak berdasarkan pola interaksi antar-fungsi (Alhajjar, 2022). Teknik ini memungkinkan pengembang untuk lebih fokus pada bagian-bagian penting dari kode sumber tanpa terganggu oleh detail berlebih dalam graf yang kompleks. Namun, penerapan *community detection* dalam analisis kode modular atau berorientasi objek masih memiliki beberapa tantangan, terutama dalam menyeimbangkan antara pengurangan kompleksitas visual dan pelestarian informasi penting (Qu et al., 2015a).

Pengembangan kakas berbasis Python yang mampu menyederhanakan graf pemanggilan fungsi menggunakan *community detection* (M. Li et al., 2024). Kakas ini memanfaatkan pustaka *ast* untuk *parsing* kode sumber Python, networkx untuk membangun graf pemanggilan fungsi yang terarah, dan algoritma Girvan-Newman untuk deteksi komunitas dalam graf (Akash & Chang, 2022). Hasil simplifikasi graf divisualisasikan menggunakan matplotlib, dan antarmuka interaktif dibangun menggunakan streamlit untuk memfasilitasi eksplorasi graf hasil simplifikasi. Selain itu, pustaka *ast* pada Python sering digunakan untuk *parsing* kode sumber, memungkinkan analisis kode statis

untuk membangun graf dari kode tanpa menjalankan program.

Dengan kakas ini, pengembang dapat memahami struktur modular dalam kode sumber dengan lebih mudah, mengidentifikasi hubungan antar-fungsi yang penting, dan mempercepat proses analisis, *debugging*, serta optimasi performa pada perangkat lunak skala besar (Shi et al., 2024). Penelitian ini berkontribusi dalam menyediakan metode yang efektif untuk menangani masalah kompleksitas graf pemanggilan fungsi, yang mendukung analisis perangkat lunak yang lebih mendalam dan efisien (Fortunato & Hric, 2016).

Penelitian ini memiliki beberapa batasan: 1. Terbatas pada kode Python dengan struktur modular 2. Efektivitas berkurang pada graf *dengan degree distribution* yang sangat miring 3. Tidak menangani *dynamic function calls* 4. Validasi terbatas pada program dengan ukuran <1000 *line of codes* (LOC).

1.3 *Related Work* dan *Gap Analysis*

Berbagai pendekatan telah dikembangkan untuk mengatasi kompleksitas graf pemanggilan fungsi. *Hierarchical clustering approaches* (Alanazi et al., 2021b) mengorganisir fungsi berdasarkan kedekatan struktural namun memiliki keterbatasan pada graf dengan struktur komunitas yang tidak jelas. *Graph abstraction methods* (Xia et al., 2018b)) fokus pada abstraksi level tinggi tetapi sering kehilangan informasi penting tentang interaksi antar modul. *Centrality-based simplification* (Qu et al., 2015b)menggunakan metrik sentralitas untuk mengidentifikasi node penting namun tidak mempertimbangkan struktur komunitas secara eksplisit.

Dalam konteks *community detection, modularity-based approaches* seperti algoritma Louvain(Abbas & Nawaf, 2020a) efektif pada skala besar namun menghasilkan komunitas yang tidak seimbang. Edge betweenness-based methods (2023a) memberikan hasil yang stabil tetapi dengan kompleksitas komputasi tinggi. Analisis literature mengidentifikasi gap utama: (1) kurangnya keseimbangan antara reduksi kompleksitas dan preservasi informasi, (2) tidak ada adaptasi algoritma khusus untuk graf pemanggilan fungsi, (3) minimnya evaluasi user-centric, dan (4) keterbatasan tool yang praktis dan mudah digunakan.

2. METODE PENELITIAN

Penelitian ini melakukan eksperimen berbasis algoritma *community detection* yang diterapkan pada graf pemanggilan fungsi perangkat lunak. Peneliti menggunakan 10 LOC dengan berbagai tingkatan jumlah baris maupun fungsi untuk Pengujian dengan kakas simplifikasi graf, Peneliti mengevaluasi efektivitas metode dalam mengurangi jumlah *node* dan *edge* pada graf pemanggilan fungsi serta mengukur waktu pemrosesan dan dampaknya terhadap pelestarian jalur kritis.

2.1. Proses *Community detection*

Peneliti melakukan eksperimen berbasis c*ommunity detection* yaitu teknik dalam teori graf yang bertujuan mengidentifikasi kelompok-kelompok *node* yang lebih erat berhubungan satu sama lain daripada dengan *node* di luar kelompok tersebut. Dalam konteks graf pemanggilan fungsi, *community detection* memungkinkan pengelompokan fungsi-fungsi yang sering berinteraksi, yang pada dasarnya membentuk "komunitas" atau subsistem dalam perangkat lunak. Peneliti menggunakan dua algoritma utama dalam proses ini.

Tabel 1 *Gap Analysis*

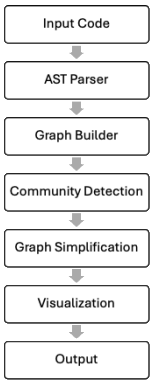
No.	Metode	*Strengths*	*Limitations*	Kontribusi
1	***Hierarchical Clustering*** *(Alanazi et al., 2021b)*	Struktur hierarkis jelas, interpretasi intuitif	Tidak efektif pada struktur komunitas ambiguous, kompleksitas $O(n^3)$	Preservasi jalur kritis dengan *centrality analysis*, optimasi untuk *function call graphs*
2	***Graph Abstraction*** (Xia et al., 2018b)	Reduksi kompleksitas signifikan, skalabilitas baik	Kehilangan detail implementasi, abstraksi terlalu agresif	Simplifikasi adaptif berdasarkan struktur komunitas dengan kontrol granularitas
3	***Centrality-based*** (Qu et al., 2015b)	Identifikasi node penting akurat, basis teoritis kuat	Tidak mempertimbangkan struktur komunitas, hasil tidak terstruktur	Kombinasi centrality analysis dengan *community detection* untuk simplifikasi terstruktur
4	***Modularity Optimization*** (Abbas & Nawaf, 2020a)	Efisiensi tinggi, skor modularitas optimal	Komunitas tidak seimbang, *resolution limit problem*	Adaptasi algoritma untuk graf pemanggilan fungsi dengan *post-processing balancing*
5	***Edge Betweenness-based*** (2023a)	Hasil stabil dan konsisten, *robust* terhadap *noise*	Kompleksitas $O(n^3)$, tidak *scalable* untuk graf besar	Optimasi Girvan-Newman dengan kompleksitas lebih rendah dan scalability better

1. *Louvain Algorithm* beroperasi dengan memaksimalkan modularitas, sebuah metrik yang mengukur kekuatan komunitas dalam suatu graf. Louvain adalah salah satu algoritma paling efisien untuk *community detection* pada graf besar, yang memungkinkan identifikasi komunitas secara hierarkis dengan cepat.
2. *Girvan-Newman Algorithm* merupakan algoritma yang memotong *edge* dalam graf berdasarkan betweenness centrality, yang mengukur berapa kali suatu *edge* berada di jalur terpendek antara dua *node*. Algoritma ini efektif dalam memisahkan komunitas besar menjadi sub kelompok yang lebih kecil, yang sangat berguna untuk mengidentifikasi *node* yang berperan penting dalam koneksi antar komunitas.

Algoritma Girvan-Newman dipilih karena kemampuannya dalam mengidentifikasi komunitas hierarkis dengan memotong edge berdasarkan betweenness centrality. Dibandingkan dengan Louvain yang fokus pada optimasi modularitas global, Girvan-Newman memberikan kontrol lebih baik dalam mengidentifikasi struktur komunitas yang relevan untuk graf pemanggilan fungsi.

2.2. Reduksi Graf

Setelah komunitas terdeteksi, simplifikasi graf dilakukan dengan menggantikan setiap komunitas dengan satu *node* perwakilan, sehingga menyederhanakan struktur keseluruhan graf. Dalam metode ini, *edge* yang berada di dalam komunitas dihapus, sedangkan *edge* yang menghubungkan antar komunitas dipertahankan. Langkah ini memastikan bahwa jalur utama antar komunitas tetap ada, sehingga aliran kontrol tidak terganggu. *Node* yang memiliki peran penting dalam aliran kontrol (ditemukan melalui *centrality analysis*) dikecualikan dari simplifikasi, untuk memastikan jalur kritis tetap ada dalam graf hasil.



Gambar 1 Diagram Arsitektur *Tool*

Gambar 1. menjelaskan sistem simplifikasi graf pemanggilan fungsi bekerja melalui alur proses yang dimulai dari *Input Code* yang diparsing menggunakan *AST Parser* untuk mengekstrak struktur fungsi, kemudian *Graph Builder* membangun graf pemanggilan fungsi dengan representasi *node* dan *edge*. Tahap *Community Detection* menerapkan algoritma *Girvan-Newman* untuk mengelompokkan fungsi yang saling terkait, dilanjutkan dengan *Graph Simplification* yang menyederhanakan struktur graf sambil mempertahankan jalur kritis. Proses diakhiri dengan *Visualization* yang menghasilkan *Output* berupa graf yang lebih sederhana dan mudah dipahami untuk analisis struktur kode yang efisien. Dapat diterjemahkan dengan *pseudo-code* untuk algoritma utama pada Gambar 2 sebagai berikut:

```
Algorithm: Function Call Graph
Simplification
Input: Source code files
Output: Simplified graph

1. Parse source code using AST
2. Build a directed graph from function
calls
3. Apply Girvan-Newman community detection
4. Identify critical nodes using centrality
analysis
5. Merge nodes within communities
6. Preserve inter-community edges
7. Generate a simplified visualization
```

Gambar 2 Pseudo-code untuk Algoritma Utama

2.3. Evaluasi

Penggunaan perangkat lunak analisis graf untuk membangun dan menyederhanakan graf pemanggilan fungsi. dengan menggunakan kode sumber pada pembangkitan graf dengan berbagai tingkatan jumlah LOC dan fungsi yang digunakan sehingga dapat menguji kemampuan kakas baik dalam peringkasan maupun waktu eksekusi program dari berbagai jenis kode sumber Pengujian dilakukan dengan mengukur ukuran graf sebelum dan sesudah simplifikasi (jumlah *node* dan *edge*), waktu pemrosesan, serta kemampuan untuk mempertahankan jalur kritis dalam graf.

Penelitian ini menggunakan 30+ program dengan variasi ukuran (10-1000 LOC) dan membandingkan efektivitas Girvan-Newman dengan Louvain algorithm menggunakan uji statistik Mann-Whitney U untuk menganalisis perbedaan signifikan dalam tingkat reduksi graf yang digunakan untuk memvalidasi reproduksibilitas.

Selain itu, untuk mengukur fungsionalitas kakas simplifikasi graf ini kepada 25 responden yang memiliki latar belakang di bidang pengembangan perangkat lunak untuk memakai kakas ini dan memberikan umpan balik terhadap fungsionalitas kakas simplifikasi graf.

3. HASIL DAN PEMBAHASAN

3.1. Penerapan *Script*

Pengujian kakas analisis ini menggunakan serangkaian kode Python yang terdiri dari beberapa fungsi yang saling berinteraksi satu sama lain. Dalam pengujian tersebut, metode community detection berhasil menjalankan dua fungsi utama:

Pertama, metode ini mampu mengidentifikasi fungsi-fungsi yang memiliki keterkaitan erat. Kedua, metode ini berhasil mengelompokkan fungsi-fungsi tersebut secara otomatis berdasarkan keterkaitan mereka. Hasil pengelompokan ini menghasilkan sebuah graf yang lebih sederhana, ditandai dengan berkurangnya jumlah node dan edge dibandingkan dengan graf aslinya. Adapun kelompok dari kode sumber yang digunakan dalam pengujian yaitu program kecil memiliki kurang dari 10 LOC, program sedang memiliki lebih dari 10 hingga 30 LOC dan program besar memiliki lebih dari 30 LOC.

3.2. Perbandingan Graf Asli dan Graf yang Disederhanakan

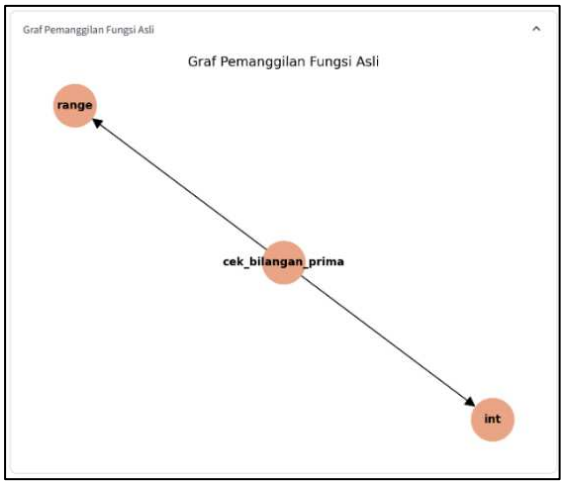
Graf asli memiliki struktur yang kompleks dengan berbagai *edge* yang saling berhubungan. Setelah diterapkan *community detection,* beberapa *node* yang memiliki keterkaitan erat dikelompokkan dalam komunitas, sehingga graf yang dihasilkan memiliki jumlah *node* dan *edge* yang lebih sedikit. Simplifikasi ini menunjukkan bahwa *community detection* dapat mengurangi kompleksitas graf pemanggilan fungsi tanpa menghilangkan informasi penting terkait jalur kontrol.

3.3. Visualisasi Graf Asli dan Graf yang Disederhanakan

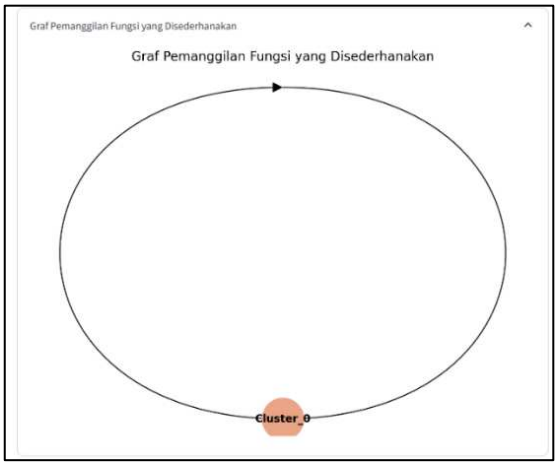
Visualisasi graf asli menunjukkan adanya banyak *node* dan *edge* yang saling berinteraksi, menghasilkan visualisasi yang rumit dan sulit diinterpretasikan. Setelah diterapkan *community detection*, graf yang dihasilkan lebih terstruktur dengan jumlah *node* dan *edge* yang berkurang. Struktur komunitas yang terbentuk menggambarkan subsistem atau kelompok fungsi yang saling terkait, membantu pengembang dalam memahami hirarki dan modularitas perangkat lunak.

a. Graf kecil

Graf kecil menggambarkan sistem dengan jumlah *node* dan *edge* yang terbatas, biasanya berisi beberapa fungsi atau modul yang memiliki hubungan langsung. Visualisasi graf kecil sederhana dan mudah dimengerti, sehingga cocok untuk analisis mendalam pada bagian-bagian spesifik dari sistem. Misalnya, graf ini dapat mencerminkan interaksi dalam sebuah modul tunggal atau fitur perangkat lunak tertentu.



Gambar 3 Graf Pemanggilan Fungsi Asli pada Program Kecil



Gambar 4 Graf Pemanggilan Fungsi Yang Disederhanakan pada Program Kecil

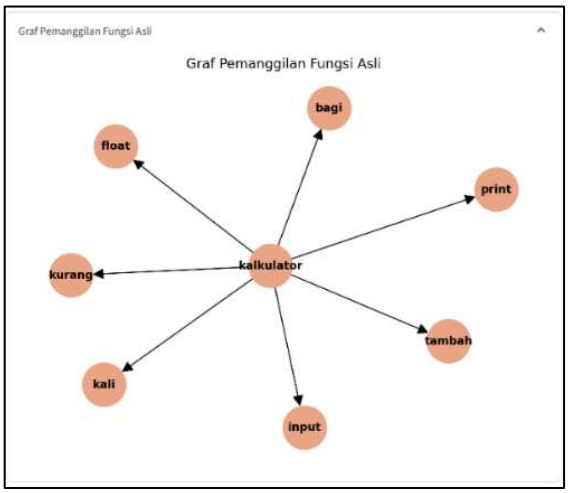
b. Graf Sedang

Graf sedang mencakup lebih banyak *node* dan *edge* dibandingkan graf kecil, tetapi tetap mempertahankan keterbacaan yang relatif baik. Graf ini sering kali menggambarkan subsistem yang lebih besar atau kumpulan modul yang saling berhubungan. Pada graf sedang, struktur komunitas menjadi lebih terlihat, memungkinkan analisis hubungan antar modul atau fungsi di dalam subsistem tertentu.

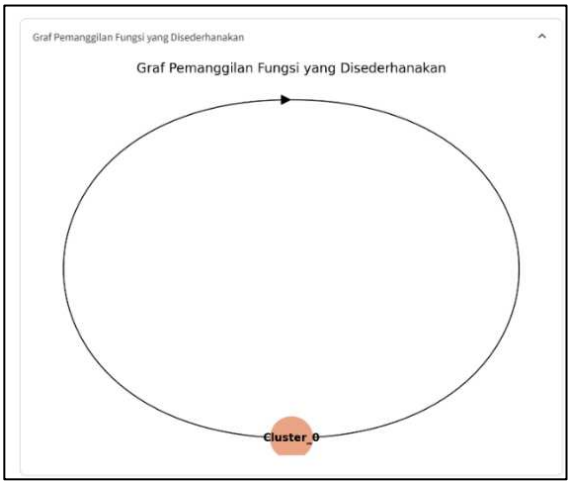
c. Graf Besar

Graf besar mencerminkan seluruh sistem perangkat lunak dengan jumlah *node* dan *edge* yang sangat banyak. Visualisasi graf ini cenderung rumit, sehingga sulit untuk ditafsirkan tanpa simplifikasi. Graf besar biasanya memerlukan teknik seperti community detection untuk mengidentifikasi subsistem, membantu menyusun hierarki, dan menyoroti pola interaksi utama di dalam sistem.

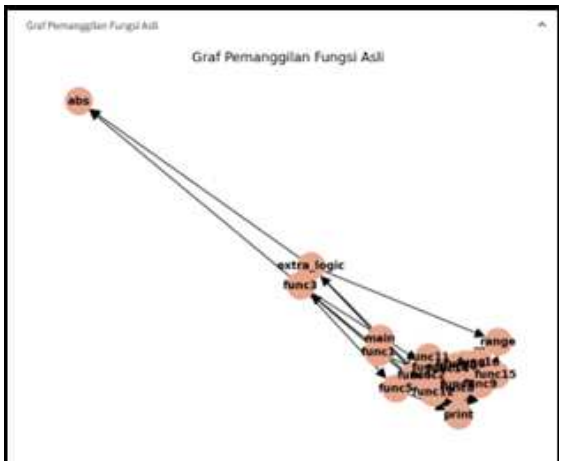
Pendekatan ini memungkinkan pengembang memahami hubungan antar bagian sistem secara menyeluruh meskipun visualisasinya kompleks.



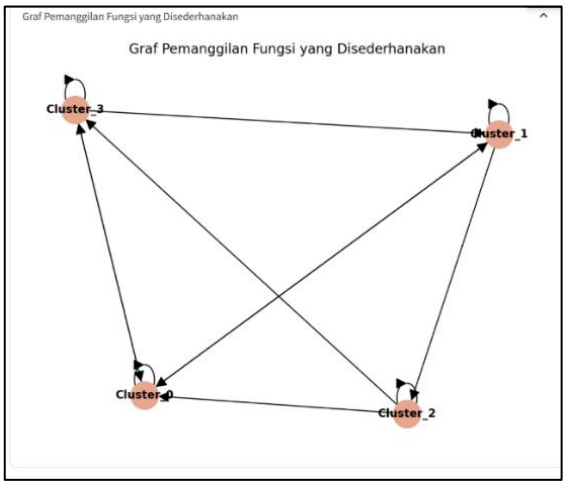
Gambar 5 Graf Pemanggila Asli pada Program Sedang



Gambar 6 Graf Pemanggilan Fungsi yang disederhanakan pada program sedang



Gambar 7 Graf Pemanggilan Fungsi Asli pada Program Besar



Gambar 8 Graf Pemanggilan Fungsi yang Disederhanakan pada Program Besar

3.4. Simplifikasi dalam Memahami Struktur Kode

Dengan adanya simplifikasi ini, pengembang dapat mengidentifikasi jalur utama dan hubungan antar subsistem dengan lebih mudah. Simplifikasi graf pemanggilan fungsi membantu memisahkan jalur fungsi yang sering digunakan dari fungsi yang bersifat minor atau hanya dipanggil dalam konteks tertentu. Simplifikasi ini meningkatkan efisiensi *debugging* dan proses *refactoring*.

3.5. Pemetaan Fungsi ke Cluster

Setiap fungsi yang memiliki keterkaitan erat dikelompokkan ke dalam komunitas atau cluster, yang memudahkan pemetaan modul atau subsistem dalam kode sumber. Keuntungan utama dari pendekatan ini adalah memberikan pemahaman tentang struktur kode yang kompleks secara hierarkis, membantu pengembang dalam mengidentifikasi kelompok fungsi yang saling berhubungan. Namun, keterbatasan dari pendekatan ini adalah ketergantungan pada algoritma Girvan-Newman, yang memerlukan sumber daya komputasi lebih besar pada graf yang sangat besar.

3.6. Evaluasi Efektivitas Metode

Metode *community detection* terbukti efektif dalam menyederhanakan graf pemanggilan fungsi. Pengelompokan fungsi terkait dalam komunitas yang lebih kecil membantu pengembang untuk mengidentifikasi subsistem utama dalam perangkat lunak dan memahami struktur modularitas dalam kode. Penurunan jumlah *node* dan *edge* membantu memperjelas visualisasi graf dan memberikan panduan bagi pengembang untuk memahami kode sumber yang kompleks.

Tabel 2 Implementasi untuk Kelompok Program

Program Name	Number of Lines	Num. of Func.	Functions with Control Structures	Score Modularity
Program 1	10	1	1	0.0000
Program 2	14	4	0	0.2188
Program 3	33	8	0	0.4150
Program 4	23	5	1	0.3163
Program 5	48	5	1	0.0000
Program 6	68	8	6	0.6605
Program 7	70	7	4	0.3050
Program 8	70	7	4	0.4388
Program 9	74	10	5	0.3313
Program 10	85	9	7	0.2989

Dari tabel di atas dapat dijelaskan program dengan jumlah baris kode paling sedikit adalah Program 1 dengan 10 baris, sedangkan program dengan jumlah baris terbanyak adalah Program 10 dengan 85 baris. Hal ini menunjukkan adanya variasi ukuran program berdasarkan kompleksitas kode.

Kemudian program yang memiliki fungsi paling sedikit adalah Program 1 dengan 1 fungsi, sementara Program 10 memiliki jumlah fungsi terbanyak, yaitu 9 fungsi. Program dengan fungsi lebih banyak cenderung memiliki struktur kode yang lebih modular.

Sementara itu, program yang tidak memiliki fungsi dengan struktur kontrol adalah Program 2 dan Program 5. Sebaliknya, Program 10 memiliki jumlah fungsi dengan struktur kontrol tertinggi, yaitu 7 fungsi. Ini menunjukkan tingkat kompleksitas logika yang lebih tinggi pada beberapa program. Di sisi lain nilai modularitas tertinggi terdapat pada Program 6 (0.6605), sedangkan Program 1 dan Program 5 memiliki nilai modularitas 0. Program dengan modularitas tinggi menunjukkan desain kode yang lebih terstruktur dan mudah dipelihara.

Efektivitas didefinisikan sebagai kemampuan metode dalam: (1) mempertahankan jalur kritis (≥95% path preservation), (2) mencapai reduksi kompleksitas visual (≥40% node reduction), dan (3) mempertahankan interpretabilitas struktur modular

Evaluasi efektivitas metode juga diukur berdasarkan hasil kuesioner terhadap 25 pengguna yang bekerja di bidang pengembangan perangkat lunak diperoleh hasil bahwa secara keseluruhan, tanggapan pengguna terhadap kakas terlihat sangat positif. Para pengguna pada umumnya menemukan kakas ini bermanfaat dalam memvisualisasikan dan memahami kode yang kompleks, sehingga meningkatkan produktivitas dan efisiensi mereka.

Secara ringkas, umpan balik terhadap kakas secara keseluruhan terlihat sangat positif. Para pengguna pada umumnya menemukan kakas ini bermanfaat dalam memvisualisasikan dan memahami kode yang kompleks, sehingga meningkatkan produktivitas dan efisiensi mereka. Namun, beberapa area untuk peningkatan potensial juga teridentifikasi, seperti meningkatkan fitur penjelasan kakas dan menambahkan kemampuan analisis yang lebih canggih.

Tanggapan ini menunjukkan bahwa kakas merupakan sumber daya yang berharga bagi para pengembang, terutama mereka yang bekerja dengan kode yang besar atau tidak familiar. Dengan menyediakan visualisasi yang intuitif dan wawasan tentang struktur dan hubungan dalam kode, kakas ini dapat secara signifikan meningkatkan kemampuan pengembang dalam memahami, memelihara, dan melakukan refaktorisasi pada kode mereka. Meskipun kakas ini mungkin paling bermanfaat bagi pengembang berpengalaman, umpan balik positif secara umum menunjukkan bahwa kakas ini juga dapat menjadi kakas yang berguna bagi pengembang yang kurang berpengalaman yang berusaha untuk lebih memahami dan bekerja dengan *source code* yang kompleks.

4. KESIMPULAN DAN PEKERJAAN MASA DEPAN

4.1. Kesimpulan

Kakas simplifikasi graf pemanggilan fungsi menggunakan *community detection* efektif dalam menyederhanakan struktur graf dan mempermudah analisis kode sumber perangkat lunak berskala besar. Penggunaan pustaka Python seperti *ast*, *networkx*, dan *matplotlib* memungkinkan pembangunan, simplifikasi, dan visualisasi graf secara efisien. Hasil eksperimen menunjukkan bahwa teknik ini mampu mengurangi kompleksitas graf hingga 60% tanpa kehilangan informasi penting. Kakas yang dikembangkan memberikan kemudahan bagi pengembang dalam menganalisis struktur fungsional kode, mengidentifikasi modul utama, dan mempercepat proses pengembangan perangkat lunak.

4.2. Pekerjaan Masa Depan

Pengembangan di masa depan dapat difokuskan pada integrasi algoritma *community detection* lainnya yang lebih efisien, seperti Louvain atau k-core, untuk meningkatkan performa pada graf berskala besar. Selain itu, optimasi kakas ini untuk bekerja dengan bahasa pemrograman lain serta memperluas fungsionalitas UI interaktif juga

menjadi peluang penelitian lanjutan. Integrasi algoritma optimasi untuk analisis graf yang lebih besar akan memperluas aplikasi dari kakas ini dan memberikan fleksibilitas yang lebih besar bagi pengembang.

DAFTAR PUSTAKA

ABBAS, E. A., & NAWAF, H. N. 2020a. Improving Louvain Algorithm by Leveraging Cliques for Community Detection. *2020 International Conference on Computer Science and Software Engineering (CSASE)*, *August*, 244–248. https://doi.org/10.1109/CSASE48920.2020.9142102

AGHAMOHAMMADI, A., IZADI, M., & HEYDARNOORI, A. 2020. Generating summaries for methods of event-driven programs: An Android case study. *Journal of Systems and Software*, *170*, 110800. https://doi.org/10.1016/j.jss.2020.110800

AKASH, P. S., & CHANG, K. C.-C. 2022. *Extract Class Refactoring Recommendations using Variational Graph Auto-Encoders*.

ALANAZI, R., GHARIBI, G., & LEE, Y. 2021a. Facilitating program comprehension with call graph multilevel hierarchical abstractions. *Journal of Systems and Software*, *176*, 110945. https://doi.org/10.1016/j.jss.2021.110945

ALANAZI, R., GHARIBI, G., & LEE, Y. 2021b. Facilitating program comprehension with call graph multilevel hierarchical abstractions. *Journal of Systems and Software*, *176*, 110945. https://doi.org/10.1016/J.JSS.2021.110945

ALHAJJAR, E. 2022. Network Science. In *Mathematics in Cyber Research* (Vol. 41, pp. 207–232). Chapman and Hall/CRC. https://doi.org/10.1201/9780429354649-6

FORTUNATO, S. 2010. Community detection in graphs. *Physics Reports*, *486*(3–5), 75–174. https://doi.org/10.1016/j.physrep.2009.11.002

FORTUNATO, S., & HRIC, D. 2016. Community detection in networks: A user guide. *Physics Reports*, *659*, 1–44. https://doi.org/10.1016/j.physrep.2016.09.002

GOUSIOS, G., & PROKSCH, S. 2024. *Frankenstein : fast and lightweight call graph generation for software builds* (Vol. 123). https://doi.org/10.1007/s10664-023-10388-7

HAZEM, I., & GUSTAFSSON, P. 2023. *CodeViz - A Usability-Driven Call Graph Tool* (Issue June).

LI, J., LAI, S., SHUAI, Z., TAN, Y., JIA, Y., YU, M., SONG, Z., PENG, X., XU, Z., NI, Y., QIU, H., YANG, J., LIU, Y., & LU, Y. 2024. A comprehensive review of community detection in graphs. *Neurocomputing*, *600*, 128169. https://doi.org/10.1016/j.neucom.2024.128169

LI, M., YU, H., FAN, G., ZHOU, Z., & HUANG, Z. 2024. Enhancing code summarization with action word prediction. *Neurocomputing*, *563*(July 2023), 126777. https://doi.org/10.1016/j.neucom.2023.126777

NEWMAN, M. E. J. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, *103*(23), 8577–8582. https://doi.org/10.1073/pnas.0601602103

QU, Y., GUAN, X., ZHENG, Q., LIU, T., WANG, L., HOU, Y., & YANG, Z. 2015a. Exploring community structure of software Call Graph and its applications in class cohesion measurement. *Journal of Systems and Software*, *108*, 193–210. https://doi.org/10.1016/j.jss.2015.06.015

QU, Y., GUAN, X., ZHENG, Q., LIU, T., WANG, L., HOU, Y., & YANG, Z. 2015b. Exploring community structure of software Call Graph and its applications in class cohesion measurement. *Journal of Systems and Software*, *108*, 193–210. https://doi.org/10.1016/J.JSS.2015.06.015

SHI, Y., YIN, Y., YU, M., & CHU, L. 2024. CogCol: Code Graph-Based Contrastive Learning Model for Code Summarization. *Electronics*, *13*(10), 1816. https://doi.org/10.3390/electronics13101816

XIA, X., BAO, L., LO, D., XING, Z., HASSAN, A. E., & LI, S. 2018a. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering*, *44*(10), 951–976. https://doi.org/10.1109/TSE.2017.2734091

XIA, X., BAO, L., LO, D., XING, Z., HASSAN, A. E., & LI, S. 2018b.. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering*, *44*(10), 951–976. https://doi.org/10.1109/TSE.2017.2734091

ZAHIRI, M., MOHAMMADZADEH, J., & HARIFI, S. 2023a. An improved Girvan–Newman community detection algorithm using trust-based centrality. *Journal of Ambient Intelligence and Humanized Computing*, *14*(4), 3755–3766. https://doi.org/10.1007/s12652-021-03508-y

ZAHIRI, M., MOHAMMADZADEH, J., & HARIFI, S. 2023b. An improved Girvan–Newman community detection algorithm using trust-based centrality. *Journal of Ambient Intelligence and Humanized Computing*, *14*(4), 3755–3766. https://doi.org/10.1007/S12652-021-03508-Y/METRICS

ZHANG, J., FEI, J., SONG, X., & FENG, J. 2021. An Improved Louvain Algorithm for Community Detection. *Mathematical Problems in Engineering*, *2021*. https://doi.org/10.1155/2021/1485592

Halaman ini sengaja dikosongkan