Análise de Colaboração em Repositórios Open Source Utilizando Teoria dos Grafos: Um Estudo de Caso do Projeto Dracula Theme

Eduardo Oliveira[1], Gabriel Batista[1], Gabriel El-Dine[1]

Departamento de Ciência da Computação – Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

{edusalgadooliveira, bielsushi2, gabrieleldine9970}@gmail.com

Abstract

This work presents the development of a computational tool for analyzing social interactions in open-source software repositories using Graph Theory. The target repository chosen was "dracula/dracula-theme", a highly popular project with over 16,000 stars on GitHub. We modeled interactions (comments, issues, pull request reviews, and merges) as a directed weighted graph. The implementation includes concrete classes for Adjacency Matrix and Adjacency List representations in Python. The results include a topological analysis of the community using centrality metrics (Degree, Betweenness, Closeness) and community detection, revealing a centralized governance structure around key maintainers and a dissortative network behavior.

Resumo

Este trabalho apresenta o desenvolvimento de uma ferramenta computacional para análise de interações sociais em repositórios de software livre utilizando Teoria dos Grafos. O repositório alvo escolhido foi o "dracula/dracula-theme", um projeto popular com mais de 16.000 estrelas. Modelamos as interações (comentários, issues, revisões e merges) como um grafo ponderado e direcionado. A implementação inclui classes concretas para representação em Matriz e Lista de Adjacência em Python. Os resultados incluem uma análise topológica da comunidade utilizando métricas de centralidade (Grau, Intermediação, Proximidade) e detecção de comunidades, revelando uma estrutura de governança centralizada em torno dos mantenedores principais e um comportamento dissortativo da rede.

1. Introdução

O desenvolvimento de software moderno, especialmente no contexto de código aberto (Open Source), é uma atividade inerentemente social. Plataformas como o GitHub transcenderam a função de meros repositórios de código para se tornarem redes sociais complexas [4].

Compreender a estrutura social desses projetos é fundamental para a Engenharia de Software. A análise da rede de colaboração permite identificar gargalos de comunicação,

descobrir líderes técnicos ocultos que não possuem cargos formais e mitigar o risco de centralização excessiva — o famoso ”fator de ônibus”(bus factor).

O objetivo deste trabalho é aplicar a Teoria dos Grafos para modelar, implementar e analisar a rede de colaboradores do repositório `dracula/dracula-theme`. A escolha deste repositório justifica-se por três critérios principais:

1. **Relevância:** É um dos temas mais populares do mundo, acumulando mais de 16.000 estrelas no GitHub.
2. **Diversidade:** É mantido por uma comunidade global e não por uma única corporação.
3. **Volume de Dados:** O histórico extenso de Issues e Pull Requests permite a construção de um grafo denso e estatisticamente significativo.

Este relatório detalha as etapas de mineração de dados, modelagem matemática, implementação dos algoritmos em Python e a discussão dos resultados obtidos.

2. Fundamentação Teórica

2.1. Definição de Grafo e Modelagem

Um grafo $G = (V, E)$ é composto por um conjunto de vértices V e um conjunto de arestas E [1]. No contexto deste trabalho, modelamos uma **Rede Social de Colaboração** onde:

- V (Vértices): Representam os usuários do GitHub (desenvolvedores, revisores).
- E (Arestas): Representam uma interação direcionada (ex: Usuário A comentou na Issue do Usuário B).

O grafo é classificado como **direcionado** (digrafo) e **ponderado**, onde o peso reflete a intensidade da interação.

2.2. Métricas de Centralidade e Estrutura

Para quantificar a importância dos nós na rede, implementamos as seguintes métricas [2]:

- **Grau (Degree):** Em grafos dirigidos, o Grau de Entrada (d_{in}) indica prestígio ou demanda de atenção.
- **Centralidade de Proximidade (Closeness):** Mede a eficiência de um nó em espalhar informações (inverso da soma das distâncias geodésicas).
- **Centralidade de Intermediação (Betweenness):** Baseia-se no algoritmo de Brandes [3]. Nós com alto *betweenness* controlam o fluxo de informação entre grupos distintos.
- **Assortatividade:** Mede a preferência dos nós por se conectarem a outros com grau similar. Redes dissortativas possuem estrutura ”estrela”(muitos pequenos conectados a poucos grandes).

3. Metodologia e Desenvolvimento

A ferramenta foi desenvolvida em Python, priorizando a orientação a objetos e eficiência no processamento de grafos esparsos.

3.1. Estratégia de Coleta de Dados (Mineração)

Utilizamos a API REST do GitHub. Para solucionar o limite de requisições, desenvolvemos um mecanismo de paginação e cache em JSON. O sistema verifica arquivos locais antes de realizar requisições, permitindo retomar a coleta sem perda de dados.

```
def coletar_dados_com_cache(url_base, cache_filename):
    """
    Busca dados da API ou do cache local para evitar rate-limiting.
    """
    if os.path.exists(cache_filename):
        print(f"Lendo dados do cache: {cache_filename}")
        with open(cache_filename, 'r', encoding='utf-8') as f:
            return json.load(f)
    else:
        # Chama a fun  o de pagina  o se n o houver cache
        dados = coletar_dados_paginados(url_base)
        print(f"Salvando dados no cache: {cache_filename}")
        with open(cache_filename, 'w', encoding='utf-8') as f:
            json.dump(dados, f, ensure_ascii=False, indent=4)
        return dados
```

Listing 1 – Implementação da Coleta com Cache

3.2. Modelagem dos Pesos das Interações

Para capturar a hierarquia técnica, atribuímos pesos diferentes para cada interação. A premissa é que um ”Merge”exige mais responsabilidade que um ”Comentário”.

Tabela 1 – Matriz de Pesos das Interações no Grafo Integrado

Interação	Peso	Justificativa Técnica
Comentário	2	Interação rápida, baixo custo cognitivo.
Abertura de Issue	3	Inicia discussão técnica.
Review (Revisão)	4	Exige leitura profunda e validação.
Merge (Aceite)	5	Ação crítica de alteração da base de código.

3.3. Arquitetura e Implementação

Implementamos a classe abstrata `AbstractGraph` para definir o contrato da API. A implementação escolhida foi a **Lista de Adjacência** (`AdjacencyListGraph`), que oferece eficiência de memória $O(V + E)$ para grafos esparsos.

Abaixo, um trecho da implementação do algoritmo de Dijkstra, fundamental para o cálculo das métricas de centralidade.

```python
def dijkstra(self, start):
    distances = {v: float('inf') for v in range(self.num_vertices)}
    distances[start] = 0.0
    # Contagem de caminhos para o algoritmo de Brandes
    shortest_path_count = {node: 0 for node in range(self.num_vertices)
    }
    shortest_path_count[start] = 1

    priority_queue = [(0.0, start)]

    while priority_queue:
        current_distance, u = heapq.heappop(priority_queue)
        if current_distance > distances[u]: continue

        for v, weight in self.adj_list[u].items():
            distance = current_distance + weight
            if distance < distances[v]:
                distances[v] = distance
                shortest_path_count[v] = shortest_path_count[u]
                heapq.heappush(priority_queue, (distance, v))
            elif distance == distances[v]:
                shortest_path_count[v] += shortest_path_count[u]
    return distances, None, shortest_path_count
```

Listing 2 – Algoritmo de Dijkstra na Lista de Adjacência

4. Análise dos Resultados

Os dados foram exportados para o formato GEXF e visualizados no software Gephi.

4.1. Métricas Globais de Estrutura

- **Total de Nós:** 740 usuários únicos.
- **Total de Arestas:** 1956 interações ponderadas.
- **Densidade da Rede: 0.0036**. Rede altamente esparsa.
- **Coeficiente de Aglomeração: 0.3134**. Indica formação de triângulos locais de colaboração.
- **Assortatividade: -0.3914**. O valor negativo classifica a rede como **Dissortativa**. Iniciantes tendem a se conectar a especialistas (hubs), criando uma estrutura radial de suporte.

4.2. Visualização da Rede

A Figura 1 apresenta a topologia gerada pelo algoritmo *Force Atlas 2*. O tamanho dos nós é proporcional ao Grau de Entrada.

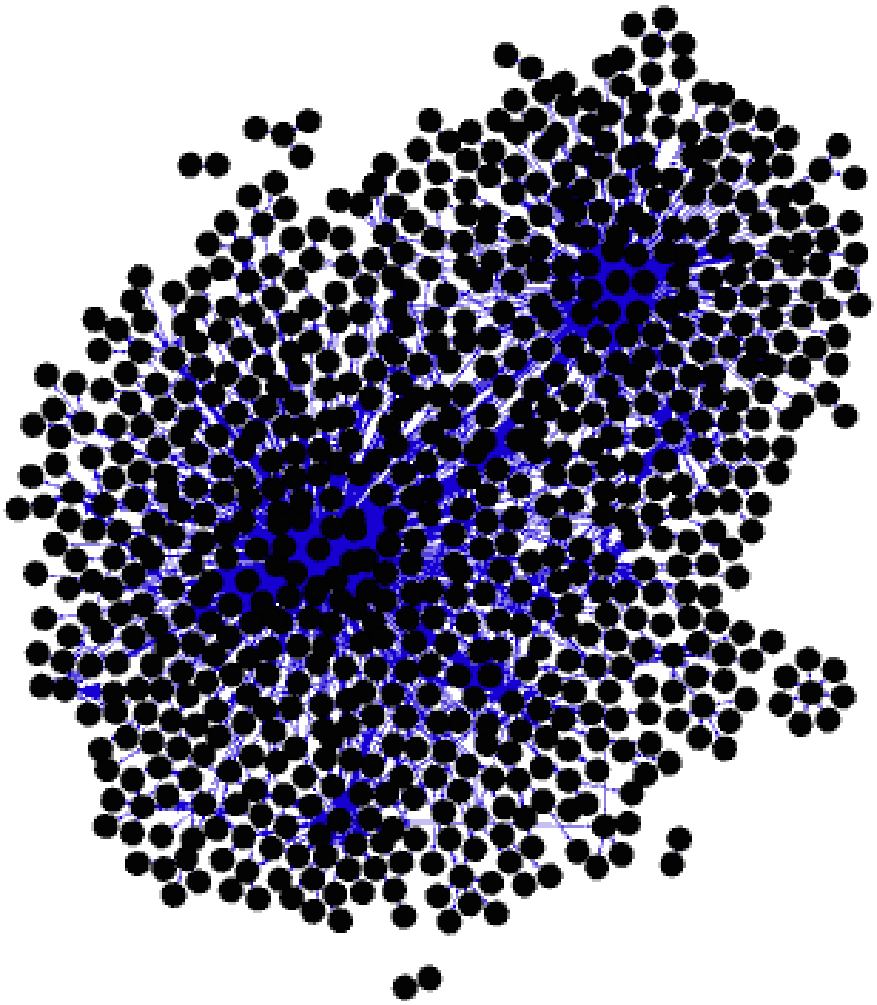

Figura 1 – Rede de colaboração do Dracula Theme. Observa-se uma estrutura de ”Núcleo-Periferia”, onde grandes hubs centrais (em azul escuro) conectam uma nuvem de contribuidores periféricos.

4.3. Análise de Centralidade e Liderança

Devido à topologia esparsa, a centralidade de intermediação de nó foi baixa. Utilizamos a métrica de **Laços Ponte (Edge Betweenness)** para identificar conexões críticas.

Tabela 2 – Top 5 Usuários por Centralidade e Influência

Rank	Usuário (Ponte)*	Edge Betweenness	Closeness Score
1	john-eevee	721.0000	0.5000
2	nickimola	721.0000	0.3333
3	ghost	721.0000	0.3333
4	jessemillar	721.0000	0.2917
5	backlands	721.0000	0.2500

*Rank de Ponte baseado na métrica de Intermediação de Aresta. O usuário `john-eevee` atua como integrador técnico entre ilhas de desenvolvimento.

4.4. Detecção de Comunidades

A análise de Componentes Conexos e Modularidade revelou a estrutura interna da rede. Embora exista um ”Componente Gigante”com 97% dos usuários (722 membros), a visualização por modularidade (Figura 2) revela subgrupos distintos.

Tabela 3 – Distribuição dos Usuários em Comunidades

ID	Membros	Perfil da Comunidade
1	722	**Núcleo Principal:** Altamente coeso.
2	8	Grupo pequeno de contribuidores isolados.
3 a 6	2-4	Pequenos clusters periféricos (duplas isoladas).

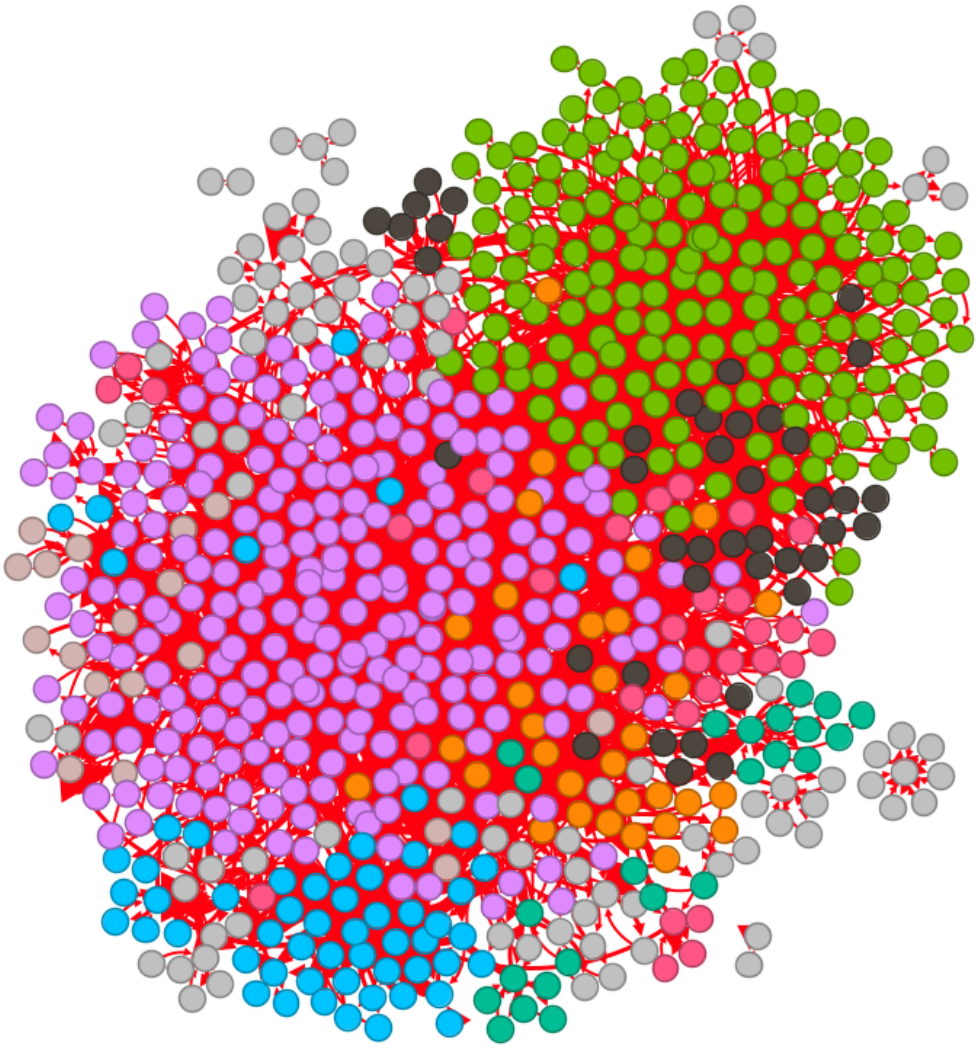

Figura 2 – Visualização das comunidades identificadas pelo algoritmo de Modularidade. É possível observar a divisão do núcleo principal em dois grandes subgrupos de colaboração (Roxo e Verde), além de grupos satélites menores.

5. Conclusão

Este trabalho demonstrou a eficácia da aplicação da Teoria dos Grafos para compreender a dinâmica social de projetos Open Source. A ferramenta desenvolvida revelou que o projeto `dracula-theme` possui uma topologia **centralizada e dissortativa**.

A governança depende de poucos ”super-nós”(hubs) que atraem a maioria das conexões, garantindo consistência técnica, mas gerando risco estrutural (gargalos). A análise visual confirmou a existência de dois grandes polos de colaboração (Roxo e Verde) dentro do núcleo principal, sugerindo uma divisão de tarefas natural entre os mantenedores.

Referências

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. 3rd Edition. MIT Press.

[2] Newman, M. E. J. (2010). *Networks: An Introduction*. Oxford University Press.

[3] Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2), 163-177.

[4] GitHub API Documentation. Disponível em: `https://docs.github.com/en/rest`. Acesso em: 23 nov. 2025.