Trabalho Prático de Grafos

Gabriel A. de Souza[1], **João P. P. Barbosa**[1],
Luis H. G. Barbosa[1]

[1]Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC-MG)

{gabriel.souza.1523286,joao.barbosa.1527547,

lhgbarbosa}@sga.pucminas.br

Abstract. *This work presents the design and implementation of a computational tool for the analysis of software repositories using Graph Theory. The case study focuses on the Node.js repository on GitHub, extracting data via the GraphQL API to map developers as nodes and their interactions (issues, pull requests, comments) as weighted edges. The tool was developed in Python, allowing for the manual calculation of metrics and comparison with the NetworkX library. The results reveal a hierarchical and unequal network, where a small group of "superusers" concentrates most of the project's influence and connectivity.*

Resumo. *Este trabalho apresenta a concepção e implementação de uma ferramenta computacional para análise de repositórios de software utilizando Teoria dos Grafos. O estudo de caso foca no repositório do Node.js no GitHub, extraindo dados via API GraphQL para mapear desenvolvedores como nós e suas interações (issues, pull requests, comentários) como arestas ponderadas. A ferramenta foi desenvolvida em Python, permitindo o cálculo manual de métricas e comparação com a biblioteca NetworkX. Os resultados revelam uma rede hierarquizada e desigual, onde um pequeno grupo de "superusuários" concentra a maior parte da influência e conectividade do projeto.*

1. Introdução

O presente trabalho consiste na concepção e implementação de uma ferramenta computacional voltada para o processamento de dados não-lineares, especificamente estruturados na forma de grafos. O objetivo principal é aplicar fundamentos matemáticos da Teoria dos Grafos em conjunto com padrões de projeto e boas práticas de Engenharia de Software para criar uma solução performática e manutenível.

Como validação prática, o sistema realiza a ingestão e análise de dados provenientes de um repositório de software de código aberto. A ferramenta mapeia as entidades do projeto (desenvolvedores, pull requests, issues, merges) em nós e suas interações em arestas, construindo um modelo de grafo que serve como base para algoritmos de travessia e cálculo de métricas. Esta abordagem permite transformar o histórico linear de versionamento em uma estrutura relacional rica, facilitando a compreensão das dependências e do fluxo de trabalho da equipe de desenvolvimento.

2. Metodologia

A metodologia adotada para o desenvolvimento deste trabalho fundamenta-se na aplicação de conceitos da Teoria dos Grafos para a análise de repositórios de software,

seguindo práticas de desenvolvimento colaborativo e Engenharia de Software. O processo está dividido nas etapas de definição de ferramentas, coleta de dados, modelagem matemática e procedimentos de entrega.

2.1. Distribuição de Responsabilidades

Os slides foram feitos em conjunto e as tarefas restantes foram distribuídas entre os membros da equipe da seguinte forma:

- Gabriel Assis de Souza: buscar com os outros membros do grupo informações sobre as implementações para redigir o documento e implementação base da ferramenta.
- João Pedro Peres Barbosa: refatoração de alguns pontos da ferramenta base e implementação dos calcúlos estatísticos.
- Luis Henrique Gonçalves Barbosa: implementação dos métodos de extração de dados do repositório do GitHub.

2.2. Ferramentas e Tecnologias

O desenvolvimento da ferramenta computacional será realizado utilizando a linguagem de programação Python, escolhida devido à sua robustez e disponibilidade de bibliotecas para manipulação de estruturas de dados.

A documentação do projeto e o relatório final serão elaborados utilizando LaTeX, garantindo a formatação acadêmica padrão. O versionamento de código e as entregas intermediárias serão gerenciados através do GitHub Classroom, conforme as especificações de cada ponto de controle.

2.3. Seleção da Fonte de Dados

Para garantir a relevância estatística e a riqueza das interações analisadas, o estudo de caso será realizado sobre um repositório público hospedado no GitHub. O critério de seleção exige que o projeto possua uma comunidade ativa. O repositório do Node.js foi escolhido para esta análise devido ao seu grande volume de interações e à sua comunidade de desenvolvimento altamente atuante, fatores cruciais para a validade e a riqueza dos dados do grafo. Um repositório secundário também foi utilizado para testes mais rápidos: tendo sido escolhido o repositório Starship.

A extração de dados focará nas interações sociais e técnicas entre os colaboradores, coletando informações sobre:

- Comentários em issues e pull requests;
- Fechamento de issues;
- Ciclo de vida de pull requests (abertura, revisão, aprovação e merge).

2.4. Extração dos Dados

Para se extrair os dados foi utilizado a API GraphQL [Inc.] pública do GitHub, a versão GraphQL foi escolhida em detrimento da opção REST por causa de sua flexibilidade em escolher os dados que estão sendo retornados e por causa do rate limit adotado pela API, sendo possível ter mais informações com um número menor de requisições. Foram escritos 7 queries: 3 relacionadas as Issues, 3 relacionadas aos Pull Requests e 1 para

verificação do Rate Limit. Ambas as queries de Issue e de Pull Request seguem um mesmo fluxo, com 1 query principal seguida de 2 auxiliares que buscam as informações faltantes, como comentários, reviews e fechamentos. A busca por Pull Requests e Issues foram paralelizadas em processos diferentes por causa do tempo gasto para a extração completa; houve também uma tentativa de paralelizar as queries secundárias com Threads porém não seria possível sem uma refatoração mais profunda pelo fato de compartilharem as mesmas variáveis e estados de conexão com o servidor.

Para se otimizar ainda mais os gastos da API, as queries secundárias recebiam como paramêtro a quantidade restante da informação que elas buscam, reduzindo efetivamente o consumo por query. Mesmo após as otimizações, o repositório com muitas informações fez com que não fosse possível completar toda a busca sem gastar todo o limite de uma conta, então foi necessário implementar uma rotação de chaves, que busca qual a próxima chave disponível e faz marcações de quando a chave utilizada reseta ao limite.

Durante o desenvolvimento dessas funcionalidades, cada erro fazia com que a busca voltasse ao início, então um sistema de cache foi adicionado, utilizando do algoritimo SHA256 sobre a query GraphQL com os paramêtros inicializados e salvos em um arquivo json. Com isso, as requisições já feitas não teriam que ser reprocessadas, acelerando o processo de desenvolvimento e diminuindo o gasto de limite de cada token utilizado.

A seguir, na figura 1, está disposto a estrutura de classes final do sistema de extração:

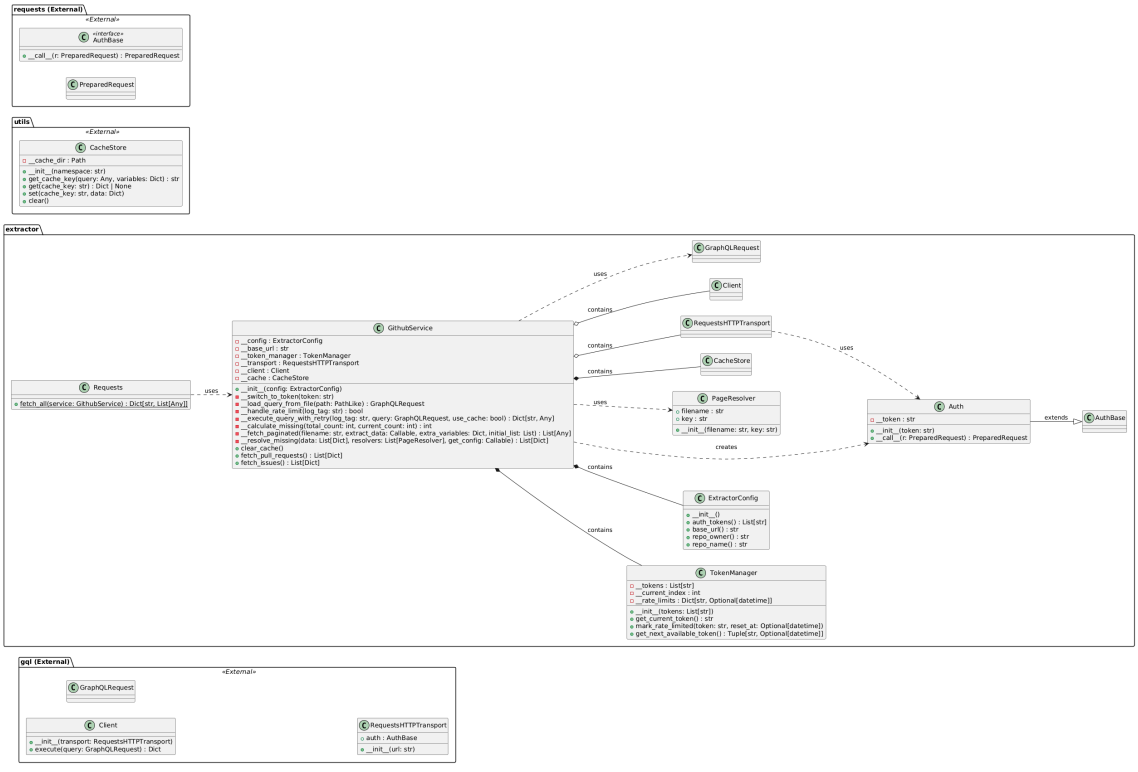

Figure 1. Diagrama de classes do sistema de extração

2.5. Modelagem dos Grafos

A estrutura central do sistema baseia-se na modelagem de redes complexas, onde cada usuário do repositório é representado como um nó e as interações constituem as arestas. O grafo resultante será simples e direcionado. Para representar relações bidirecionais entre dois usuários, serão utilizadas arestas antiparalelas. A modelagem será executada em duas fases distintas:

Fase 1: Grafos Específicos. Serão construídos três grafos independentes para isolar tipos de interações:

- Grafo de Comunicação: Baseado em comentários em issues ou pull requests.
- Grafo de Resolução: Baseado no fechamento de issues por terceiros.
- Grafo de Colaboração Técnica: Baseado em revisões, aprovações e merges de pull requests.

Fase 2: Grafo Integrado e Ponderado Será desenvolvido um grafo consolidado que unifica todas as interações. Neste modelo, as arestas possuem pesos que refletem a intensidade e a relevância técnica da colaboração. A ponderação inicial proposta segue a seguinte escala:

A ponderação inicial proposta segue a seguinte escala:

- **Peso 2**: Comentários em pull requests e issues.
- **Peso 3**: Abertura de issue comentada por outro usuário.
- **Peso 4**: Revisão ou aprovação de pull request.
- **Peso 5**: Merge de pull request.

Criou-se métodos que atuam como um agregador de dados de interação de repositórios, transformando eventos brutos (como comentários, revisões e merges em Pull Requests e Issues) em um grafo social direcionado e ponderado. A nível de alto, o processo consiste em atribuir um peso fixo e predefinido a cada tipo de interação para refletir sua importância ou ”custo de colaboração”. Ele consolida essas interações de diferentes fontes, somando os pesos de eventos múltiplos que ocorrem entre os mesmos dois usuários e na mesma direção, resultando em um único objeto InteractionsData onde o peso final de cada aresta no grafo representa a força ou a frequência combinada dos diferentes tipos de colaboração entre um par de desenvolvedores.

Esta abordagem visa priorizar interações de colaboração técnica direta (como merges) em detrimento de interações mais leves.

2.6. Critérios de Definição de Arestas

A construção da topologia da rede segue regras estritas de inclusão para garantir que o grafo represente apenas colaborações ativas. A definição formal dos elementos do grafo $G = (V, E)$ obedece à seguinte lógica:

2.6.1. Dinâmica de Criação de Arestas e Direcionalidade

As arestas representam o fluxo de interação direcionada entre dois colaboradores. A direção da aresta é definida pelo vetor da ação: do autor da interação para o autor do

objeto alvo. Para ilustrar esta lógica, considere um cenário hipotético com dois colaboradores, $User_1$ e $User_2$: o $User_1$ inicia um tópico, realizando a abertura de uma Issue ou submetendo um Pull Request. Neste momento, o $User_1$ é estabelecido como um vértice no grafo (caso ainda não o fosse). O $User_2$ engaja-se com este tópico. Isso pode ocorrer através de um comentário, uma revisão de código, uma aprovação ou um merge. No momento em que o $User_2$ realiza essa ação sobre o artefato criado pelo $User_1$, cria-se uma aresta direcionada como representado na Figura 2.

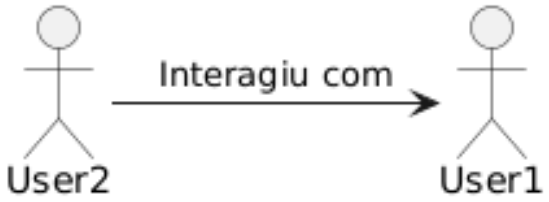

Figure 2. Exemplo de interação entre usuários

Esta aresta simboliza que o $User_2$ despendeu esforço cognitivo ou técnico sobre o trabalho do $User_1$. A mesma lógica se aplica a todas as naturezas de interação (comentários, revisões e aprovações), acumulando-se pesos conforme a tabela de ponderação definida anteriormente.

3. Desenvolvimento da Ferramenta

Após as definições feitas sobre a modelagem dos grafos, foi dado o início do desenvolvimento da biblioteca base e do processamento de estatísticas do grafo integrado. Inicialmente, como disposto na figura 3, definimos duas classes básicas: *Vertex* e *Edge*, porém a classe *Edge* não fez sentido ser utilizada no contexto da implementação concreta dos grafos de matriz de adjacência e de lista de adjacência, então ela foi removida, ficando como na figura 4

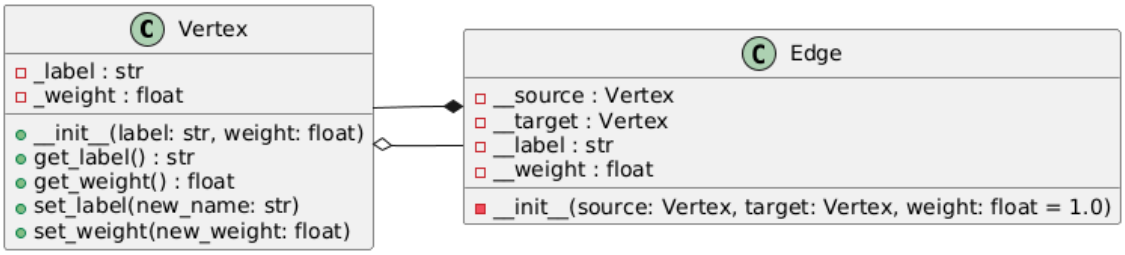

Figure 3. Diagrama inicial com as classes base da ferramenta

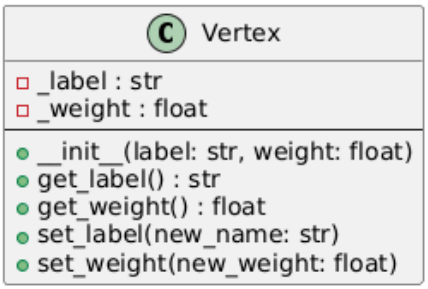

Figure 4. Diagrama com a classe Vertex

A seguir, na figura 5, está disposto o diagrama de classes da ferramenta.

A arquitetura do sistema segue princípios sólidos de Orientação a Objetos, utilizando classes abstratas (AbstractGraph e AbstractGraphStatistics) para definir interfaces padronizadas. Isso permite que diferentes formas de representar um grafo (como a AdjacencyGraphList implementada, que usa listas de adjacência) possam interagir com os algoritmos de análise sem que o código de estatística precise saber os detalhes de baixo nível de como os dados estão armazenados.

No coração da análise de dados está a classe ManualGraphStatistics. Esta classe é responsável por calcular "manualmente" (ou seja, implementando a lógica matemática explicitamente em vez de chamar bibliotecas prontas de grafos) uma vasta gama de métricas importantes. Ela calcula medidas de centralidade (Grau, Intermediação, Proximidade, Autovetor), algoritmos de ranqueamento como o PageRank, e métricas estruturais como coeficientes de clusterização, densidade e assortatividade. Para garantir performance em grafos maiores, o código utiliza processamento paralelo (ThreadPoolExecutor) em cálculos intensivos, como as centralidades de intermediação e proximidade.

Além das métricas individuais dos nós, o sistema possui algoritmos complexos para a compreensão da estrutura macroscópica da rede. Ele implementa um mecanismo de detecção de comunidades (otimizando a modularidade) para identificar grupos de nós densamente conectados entre si. O código também inclui um sistema inteligente de cache, que armazena os resultados de cálculos pesados baseados na assinatura única do grafo, evitando que o processamento seja repetido desnecessariamente se os dados não tiverem mudado.

Por fim, a infraestrutura do código lida com a entrada e saída de dados (I/O) de forma prática, projetada para integração com ferramentas como o Gephi. Através da GraphFactory e dos métodos de exportação, o sistema consegue ler arquivos CSV brutos de nós e arestas, construir a estrutura do grafo em memória, realizar todos os cálculos matemáticos e, em seguida, exportar os resultados consolidados. As métricas detalhadas por nó são salvas em planilhas CSV, e os resumos estatísticos globais são salvos em arquivos JSON.

Com o objetivo explícito de fornecer uma base para validação e comparação de desempenho, a classe NetworkXGraphStatistics atua como uma contraparte normativa à implementação manual apresentada anteriormente. Ao herdar da mesma interface abstrata (AbstractGraphStatistics), ela permite que o sistema alterne transparentemente entre os

algoritmos ”feitos à mão” e os algoritmos consagrados da biblioteca NetworkX, servindo essencialmente como um benchmark para verificar a precisão matemática e a eficiência computacional da implementação própria.

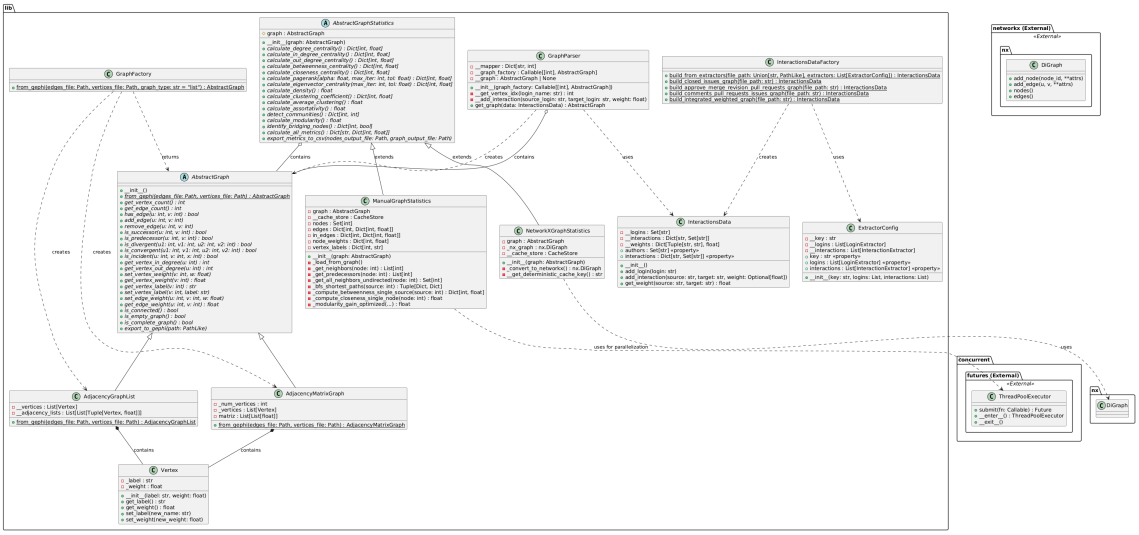

Figure 5. Diagrama de classes da ferramenta

O funcionamento interno desta classe atua como um adaptador (wrapper). A primeira ação crítica é o método convert_to_networkx, que traduz a estrutura de dados genérica do sistema (AbstractGraph) para um grafo direcionado nativo do NetworkX (nx.DiGraph). Uma vez realizada essa conversão, todos os cálculos de métricas como centralidades, PageRank, densidade e coeficientes de agrupamento são delegados diretamente às funções otimizadas da biblioteca externa, em vez de serem calculados por iterações manuais em Python.

Apesar de terceirizar a lógica matemática, a classe mantém a integração com a infraestrutura de otimização do projeto original. Ela preserva o sistema de cache (CacheStore) para evitar o reprocessamento desnecessário de métricas pesadas (como detecção de comunidades e modularidade) e utiliza o mesmo mecanismo de execução paralela para calcular os indicadores. Isso garante que a comparação entre a versão manual e a versão NetworkX seja focada na eficiência algorítmica pura, mantendo constantes as variáveis de ambiente e gerenciamento de recursos.

4. Resultados

4.1. Comparativo

Os dois métodos utilizados para gerar as estatísticas: manual e pela biblioteca networkx, tiveram resultados discrepantes, mostrando que a complexidade de cada algoritmo é alta e que as contribuições feitas pela comunidade open-source sobre os estudos de grafos é grande.

4.2. Análise dos Dados

A análise da base de dados, que compreende 18.464 usuários, revela uma estrutura de rede profundamente desigual e hierarquizada. Os dados estatísticos indicam uma distribuição

clássica de ”cauda longa”, o que significa que a vasta maioria dos usuários possui métricas de influência e conexão muito baixas, enquanto a relevância da rede está concentrada em uma minoria extremamente seleta.

No topo dessa hierarquia, identificamos um pequeno grupo de ”superusuários” que detém a maior parte do poder da rede. Nomes como nodejs-github-bot, jasnell, Trott e bnoordhuis aparecem consistentemente como os líderes, acumulando não apenas o maior número de conexões diretas (Centralidade de Grau), mas também os maiores índices de PageRank. Eles atuam como os pilares centrais que sustentam e direcionam o fluxo de informações dentro da comunidade.

Do ponto de vista das correlações estatísticas, os dados mostram que a influência (PageRank) está intrinsecamente ligada à capacidade de conexão e intermediação. Com uma correlação forte (superior a 0,80) entre o PageRank e as métricas de Grau e Intermediação, fica claro que os usuários mais importantes são aqueles que agem como ”hubs” (muitas conexões) ou ”pontes” (conectando grupos diferentes), e não necessariamente aqueles que estão apenas próximos da média dos outros participantes.

Por fim, é notável que a métrica de proximidade (Closeness Centrality) mostrou-se irrelevante para determinar a importância de um usuário, apresentando uma correlação quase nula com o PageRank. Isso reforça a conclusão de que esta rede não é democrática ou descentralizada; sua dinâmica depende inteiramente desses poucos nós centrais, cuja remoção poderia fragmentar significativamente a comunicação entre os demais integrantes.

References

Inc., G. Github graphql api documentation.