



UFOP

Caminho Mais Eficiente Para Viralização

Redes Sociais E Redes Complexas

Luccas Carneiro Thiago Ker Rany Souza Marco Antônio

22 de fevereiro de 2026

icea
Instituto de Ciências
Exatas e Aplicadas

DECSI
DEPARTAMENTO DE
COMPUTAÇÃO E SISTEMAS



Este trabalho prático aborda o tema **Redes Sociais e Redes Complexas**, com foco no estudo do **caminho mais eficiente para viralização**.

A proposta é modelar uma rede social como um **grafo não-direcionado ponderado**, comparando dois critérios de otimização:

- **Baseline (peso = 1):** menor número de saltos (*hops*).
- **Fricção (peso calculado):** menor custo de repasse, derivado da interação.

Nas seções seguintes, apresentamos o algoritmo utilizado, a modelagem, e os resultados comparativos em cenários intra e intercomunidades.



UFOP

Sumário

1 Introdução

► Introdução

► Algoritmo

► Modelagem

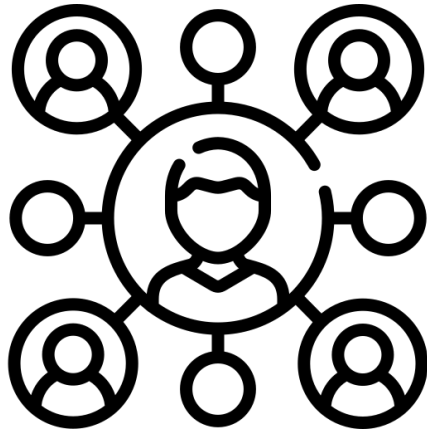
► Resultados



Contexto

1 Introdução

- Redes sociais são sistemas complexos e altamente conectados.
- Viralização não depende apenas do menor número de conexões.
- A força das relações influencia a propagação da informação.
- Grafos permitem modelar e analisar esse comportamento.





UFOP

Definição do Problema

1 Introdução

Tema

Redes Sociais e Redes Complexas.

Motivação

Em redes sociais, mensagens se propagam por cadeias de usuários conectados. Nem sempre o caminho com menor número de saltos é o caminho mais eficiente em termos de repasse.

Problema

Encontrar o caminho mais eficiente para que uma mensagem saia de um usuário A e chegue a um usuário B com menor custo.



Objetivo do Trabalho

1 Introdução

- Modelar uma rede social como grafo não-direcionado ponderado.
- Definir custo de repasse baseado na interação entre usuários.
- Comparar dois critérios para o mesmo par (A,B):
 - **Baseline (peso = 1):** menor número de saltos (*hops*).
 - **Fricção (peso calculado):** menor custo de repasse.



UFOP

Sumário

2 Algoritmo

► Introdução

► Algoritmo

► Modelagem

► Resultados



UFOP

Algoritmo Utilizado

2 Algoritmo

Dijkstra

Algoritmo para encontrar caminhos mínimos em grafos ponderados com pesos não-negativos.

Aplicação

Executado separadamente em cada grafo (baseline e fricção), partindo do vértice origem.



Algoritmo Utilizado

2 Algoritmo

Dijkstra

```
def dijkstra(grafo, origem):
    total_vertices = grafo.numVertices
    distancias = [INF] * total_vertices
    predecessor = [None] * total_vertices
    distancias[origem] = 0
    predecessor[origem] = origem
    abertos = set(range(total_vertices))
    fechados = set()
    while fechados != set(range(total_vertices)):
        vertice_atual = next(iter(abertos))
        menor_distancia = distancias[vertice_atual]
        for v in abertos:
            if distancias[v] < menor_distancia:
                menor_distancia = distancias[v]
                vertice_atual = v

        fechados.add(vertice_atual)
        abertos.remove(vertice_atual)
        for (vizinho, peso) in grafo.vizinhos(vertice_atual):
            if vizinho not in fechados:
                distancia_alternativa = distancias[vertice_atual] + peso
                if distancias[vizinho] > distancia_alternativa:
                    distancias[vizinho] = distancia_alternativa
                    predecessor[vizinho] = vertice_atual

    # retorna as listas dist e prev
    return distancias, predecessor
```



Reconstrução do Caminho

2 Algoritmo

Predecessor (prev)

Após o Dijkstra, cada nó mantém referência para o nó anterior no caminho mínimo.

Reconstrução $A \rightarrow B$

Para reconstruir o caminho:

- iniciamos em B;
- seguimos prev até A;
- invertemos a sequência.



Fluxo de Execução

2 Algoritmo

1. Selecionar par (A,B) conforme o cenário.
2. Rodar Dijkstra(A) no grafo baseline.
3. Reconstruir caminho $A \rightarrow B$ e calcular métricas.
4. Rodar Dijkstra(A) no grafo de fricção.
5. Reconstruir caminho $A \rightarrow B$ e calcular métricas.
6. Comparar divergências entre caminhos e custos.



Complexidade e Escalabilidade

2 Algoritmo

Discussão

O custo de execução depende da implementação e do tamanho da rede (V e E).

Evidência Experimental

Executamos um benchmark para observar o tempo médio de execução variando a quantidade de usuários.



UFOP

Sumário

3 Modelagem

► Introdução

► Algoritmo

► **Modelagem**

► Resultados



Modelagem da Rede

3 Modelagem

Representação

- Usuário \rightarrow vértice.
- Relação social \rightarrow aresta.
- Grafo **não-direcionado**: relação bidirecional $u \leftrightarrow v$.

Comunidades

A rede é organizada em comunidades: conexões internas são mais prováveis que conexões entre comunidades.



Da Rede Social para o Grafo

3 Modelagem

Geração

A rede social é gerada sinteticamente a partir de parâmetros (número de usuários, comunidades e probabilidades de conexão).

Conversão

Cada relação criada é convertida diretamente em arestas na lista de adjacências:

- adiciona (u, v) e (v, u) para refletir não-direcionamento;
- associa um peso conforme o modelo escolhido.



Dois Grafos (Mesma Topologia)

3 Modelagem

Estratégia

Construímos dois grafos sobre a mesma topologia (mesmos vértices e arestas), mudando apenas a função de peso.

- **Grafo Baseline:** peso = 1 em todas as arestas.
- **Grafo de Fricção:** peso calculado a partir da interação.

Objetivo

Isolar o impacto do critério de custo na escolha do caminho.



Modelagem do Peso

3 Modelagem

Baseline

Todas as arestas possuem peso 1 (otimiza menor número de saltos).

Fricção

Peso calculado por:

$$peso = 1 + \frac{\alpha}{1 + interacao}$$

Interpretação

- Interação alta → custo menor (repassa mais fluido).
- Interação baixa → custo maior (repassa mais friccionado).
- α controla a sensibilidade do custo.



Codigo Fonte

3 Modelagem

Função que calcula o peso de uma aresta no grafo de fricção

```
# Função interna que calcula o peso (custo) de uma aresta no grafo de fricção.  
def _peso_friccao(interacao, friccao_alpha):  
    """  
    Documenta a regra de negócio usada para converter "interação" em "custo de repasse".  
    Modelo de fricção (custo de repasse) por aresta.  
  
    Intuição corporativa:  
    - interacao alta => relacionamento forte => menor fricção => menor custo de repasse  
    - interacao baixa => relacionamento fraco => maior fricção => maior custo  
  
    Fórmula (custo aditivo):  
    | peso = 1 + friccao_alpha / (1 + interacao)  
  
    Onde:  
    - interacao: inteiro >= 0 (ex.: 0..10)  
    - friccao_alpha: controla o "impacto" da fricção.  
    """  
    return 1.0 + (float(friccao_alpha) / (1.0 + float(interacao)))
```



Codigo Fonte

3 Modelagem

Função que gera a rede social e retorna os dois grafos

```
# Função pública do módulo que gera a rede social e retorna os dois grafos + mapa de comunidades.
def gerar_rede_social(
    num_vertices=4000,          # Quantidade total de nós (Usuários).
    num_comunidades=4,          # Quantidade de comunidades (blocos).
    p_intra=0.06,                # Probabilidade de aresta dentro de uma comunidade.
    p_inter=0.0005,              # Fator de amostragem de pontes entre comunidades.
    max_pontes_por_par=10,       # Limite de pontes entre cada par de comunidades.
    seed=42,                     # Seed para reprodutibilidade.
    interacao_intra_min=20,       # Interação mínima intra-comunidade.
    interacao_intra_max=100,      # Interação máxima intra-comunidade.
    interacao_inter_min=0,        # Interação mínima inter-comunidade.
    interacao_inter_max=5,        # Interação máxima inter-comunidade.
    friccao_alpha=8.0            # Intensidade do custo de fricção.
):
```



Métricas Avaliadas

3 Modelagem

Para cada par (A,B), reportamos:

- **CustoTotal:** soma dos pesos das arestas no caminho.
- **Hops:** número de saltos ($|\text{caminho}| - 1$).
- **Caminho:** sequência de nós reconstruída via predecessor.

Leitura Rápida

Baseline minimiza hops; fricção minimiza custo total de repasse.



Cenários Avaliados

3 Modelagem

- **Cenário 1:** origem e destino na mesma comunidade.
- **Cenário 2:** origem e destino em comunidades diferentes.

Expectativa

Caminhos podem coincidir no cenário intra-comunidade e divergir no cenário inter-comunidades, pois o custo por fricção penaliza relações fracas.



UFOP

Sumário

4 Resultados

► Introdução

► Algoritmo

► Modelagem

► Resultados



Resultados: O que será mostrado

4 Resultados

Para cada cenário

```
luccascarneiro@Luccas-MBP src % python3 Main.py
=====
CASO 1 - Mesma Comunidade
Origem=0 | Destino=166
=====
Baseline (Menor Número De Saltos | peso=1)
CustoTotal=3 | Hops=3 | Caminho=[0, 12, 82, 166]
=====
Fricção (Menor Custo De Repasse | peso calculado)
CustoTotal=4.671782 | Hops=3 | Caminho=[0, 12, 151, 166]
=====
Divergência Detectada: o caminho por fricção difere do caminho por saltos (esperado em redes reais).
=====
CASO 2 - Comunidades Diferentes
Origem=0 | Destino=167
=====
Baseline (Menor Número De Saltos | peso=1)
CustoTotal=7 | Hops=7 | Caminho=[0, 12, 16, 93, 324, 280, 184, 167]
=====
Fricção (Menor Custo De Repasse | peso calculado)
CustoTotal=16.904558 | Hops=7 | Caminho=[0, 154, 152, 136, 332, 298, 228, 167]
=====
Divergência Detectada: o caminho por fricção difere do caminho por saltos (esperado em redes reais).
luccascarneiro@Luccas-MBP src %
```

- CustoTotal, Hops, Caminho (lista de nós)
- Baseline (peso=1) vs Fricção (peso calculado)



Cenário 1 — Mesma Comunidade

4 Resultados

Print da Execução

```
=====
CASO 1 - Mesma Comunidade
Origem=0 | Destino=166
=====
Baseline (Menor Número De Saltos | peso=1)
CustoTotal=3 | Hops=3 | Caminho=[0, 12, 82, 166]
=====
Fricção (Menor Custo De Repasse | peso calculado)
CustoTotal=4.671782 | Hops=3 | Caminho=[0, 12, 151, 166]
=====
Divergência Detectada: o caminho por fricção difere do caminho por saltos (esperado em redes reais).
=====
```

Interpretação

- Baseline e Fricção encontram caminhos com o mesmo número de hops (3).
- O custo total na Fricção é maior devido à penalização de relações fracas.
- Confirma que, dentro da mesma comunidade, há abundância de conexões fortes.



UFOP

Cenário 2 — Comunidades Diferentes

4 Resultados

Print da Execução

```
===== Rodada | Tempo(s) | Custo | Hops | Status
CASO 2 — Comunidades Diferentes
Origem=0 | Destino=167
=====
Baseline (Menor Número De Saltos | peso=1)
CustoTotal=7 | Hops=7 | Caminho=[0, 12, 16, 93, 324, 280, 184, 167]
=====
Fricção (Menor Custo De Repasse | peso calculado)
CustoTotal=16.904558 | Hops=7 | Caminho=[0, 154, 152, 136, 332, 298, 228, 167]
=====
Divergência Detectada: o caminho por fricção difere do caminho por saltos (esperado em redes reais).
luccascarneiro@Luccas-MBP src %
```

Interpretação

- Baseline e Fricção apresentam o mesmo número de hops (7).
- Apesar disso, os caminhos escolhidos são diferentes.
- Fricção evita arestas com alta resistência, mesmo mantendo a mesma profundidade.



Benchmark: Resumo (10 Rodadas)

4 Resultados

Resultados Consolidados

Caso	Modelo	Tempo Médio (s)	Custo Médio	Hops Médio
1 (Mesma Comunidade)	Baseline	0.0199	3.00	3.00
1 (Mesma Comunidade)	Fricção	0.0136	4.67	3.00
2 (Com. Diferentes)	Baseline	0.0175	6.00	6.00
2 (Com. Diferentes)	Fricção	0.0139	14.28	7.00

Leitura

Baseline otimiza distância topológica (menor número de saltos). Fricção prioriza caminhos com menor resistência de repasse, mesmo com mais hops.



Conclusões dos Resultados

4 Resultados

- O modelo de peso altera diretamente o caminho ótimo.
- Baseline é indicado para análise estrutural do grafo.
- Fricção é mais adequado para simular propagação realista.
- O aumento da rede (500 \rightarrow 1000 usuários) não altera a tendência dos resultados.



UFOP

Obrigado!

