PARTICIONAMENTO DE GRAFOS E A HEURÍSTICA DE KERNIGHAN–LIN (KL)

Discente: José Bruno Barros dos Santos
Programa de Pós-Graduação em Ciência da Computação
Disciplina: Projeto e Análise de Algoritmos

São Cristóvão - SE
2025

DEFINIÇÃO DO PROBLEMA

- Dado um grafo não direcionado G=(V,E) e um inteiro k (geralmente 2 para bipartição), particionar os vértices V em dois subconjuntos A e B de tamanhos pré-definidos (por exemplo |A| = |B| para balanceamento) de forma a minimizar o número (ou o peso) das arestas que cruzam entre A e B (o edge-cut).

Entrada formal:

- Um grafo G=(V,E), possivelmente com pesos nas arestas.
- Um inteiro n = |V|.
- A restrição de balanceamento (ex.: |A| = |B| quando n é par).

A HEURÍSTICA DE KERNIGHAN–LIN (KL)

KL é uma heurística de melhoria local que procura a melhor sequência de trocas de pares de vértices ($a \in A$, $b \in B$) para reduzir o cut.

Ganho Local D(v):

$$D(v) = E(v) - I(v)$$

Ganho de Troca g(a,b):

$$g(a,b) = D(a) + D(b) - 2 \cdot cab$$

O CICLO ITERATIVO DE KERNIGHAN–LIN

1. Busca: Escolher o par (a, b) desbloqueado que maximiza g(a,b).
2. Bloqueio: Marcar a e b como bloqueados (não podem mais ser movidos nesta iteração).
3. Atualização: Recalcular os D(v) para os vizinhos desbloqueados.
4. Repetição: Continuar até que todos os vértices estejam bloqueados, registrando a sequência de ganhos g1, g2, ...

PSEUDOCÓDIGO

```
KL(G, A, B):
  repita:
    marcar todos como desbloqueados
    para i = 1 até |V|/2:
       escolher par (a,b) desbloqueado que maximiza g(a,b)
       bloquear a e b
       registrar g_i
       atualizar D(v)
    calcular prefixo S_k = Σ g_i até k
    se max(S_k) > 0:
       aplicar as k trocas
    senão:
       parar
```

PARTIÇÃO INICIAL BALANCEADA (A | B)

CALCULAR D(V) PARA TODOS OS VÉRTICES

ESCOLHER PAR (A,B) QUE MAXIMIZA G(A,B)

BLOQUEAR A,B; ATUALIZAR D(·)

REPETIR ATÉ TODOS BLOQUEADOS

AVALIAR PREFIXOS DE TROCAS E APLICAR O MELHOR

REPETIR ITERATIVAMENTE ATÉ CONVERGIR

CENÁRIO DE APLICAÇÃO: SHARDING OTIMIZADO EM BANCOS DE DADOS DISTRIBUÍDOS

Grafo ponderado:

- Vértices (V): Entidades de dados (ex.: tabelas, registros, usuários).
- Arestas (E): Representam relações ou a frequência/custo de consultas que exigem a interação entre as entidades (ex.: uma transação que acessa o registro de um usuário e de um produto).
- Peso da Aresta: O custo ou frequência da consulta inter-entidades.

IMPLEMENTAÇÃO DA SOLUÇÃO

```python
import networkx as nx

# Função auxiliar para calcular o Custo Total de Comunicação (Cut Ponderado)
def calcular_cut_ponderado(graph, particao_A, particao_B):
    """
    Calcula o custo total de comunicação (soma dos pesos das arestas cortadas).
    Este custo simula a latência agregada de consultas cross-shard.
    """
    cut_cost = 0

    for u, v, data in graph.edges(data=True):
        weight = data.get('weight', 1) # Pega o peso da aresta (default é 1 se não houver peso)

        # Verifica se os nós estão em partições diferentes
        if (u in particao_A and v in particao_B) or (u in particao_B and v in particao_A):
            cut_cost += weight

    return cut_cost
```

```python
# 1. Criação do Grafo Ponderado (Entidades e Frequência de Consultas)
G = nx.Graph()

# Definimos duas comunidades de dados (A e B)
# Arestas internas são de baixo custo (peso 1)
G.add_edges_from([
    ('U1', 'U2', {'weight': 1}), ('U1', 'U3', {'weight': 1}), ('U2', 'U3', {'weight': 1}),
    ('P4', 'P5', {'weight': 1}), ('P4', 'P6', {'weight': 1}), ('P5', 'P6', {'weight': 1})
])

# Arestas de alto custo (simulando consultas frequentes entre clusters)
# Estas arestas DEVEM ser internas, mas estão cruzando na partição inicial ruim.
G.add_edges_from([
    ('U2', 'P4', {'weight': 10}), # Alto custo de comunicação!
    ('U3', 'P5', {'weight': 10}),
])

# 2. Partição Inicial Ruim (Sharding não otimizado)
# Shard 1 (A): {U1, U2, P4}
# Shard 2 (B): {U3, P5, P6}
A_inicial = {'U1', 'U2', 'P4'}
B_inicial = {'U3', 'P5', 'P6'}
particao_inicial = (A_inicial, B_inicial)

print("--- Cenário: Otimização de Sharding com Grafo Ponderado ---")
print(f"Número Total de Entidades (Nós): {G.number_of_nodes()}")
print(f"Custo Máximo de Comunicação por Aresta: 10 (U2-P4 e U3-P5)")
print("-" * 60)

# Custo de comunicação inicial (Cut Ponderado)
cut_inicial = calcular_cut_ponderado(G, A_inicial, B_inicial)
print(f"Custo Inicial de Comunicação Cross-Shard: {cut_inicial}")
print(f"Shard 1 (A): {A_inicial}")
print(f"Shard 2 (B): {B_inicial}")
print("-" * 60)
```

```python
# 3. Aplicação da Heurística Kernighan-Lin para Refinamento
# O KL vai trocar os nós para minimizar o cut PONDERADO.
particao_melhorada = nx.algorithms.community.kernighan_lin_bisection(
    G,
    partition=particao_inicial,
    max_iter=5 # Rodadas de refinamento
)

A_final, B_final = particao_melhorada

# Custo de comunicação após o refinamento
cut_final = calcular_cut_ponderado(G, A_final, B_final)

print("--- Resultados do Refinamento Kernighan-Lin ---")
print(f"Novo Custo de Comunicação Cross-Shard (Final): {cut_final}")
print(f"Shard Otimizado 1 (A): {A_final}")
print(f"Shard Otimizado 2 (B): {B_final}")
print("-" * 60)

# Análise do Ganho
if cut_final < cut_inicial:
    reducao_percentual = (1 - cut_final / cut_inicial) * 100
    print(f" Sucesso! Redução no Custo de Comunicação (Cut) de {cut_inicial} para {cut_final}.")
    print(f"Isso representa uma redução de {reducao_percentual:.2f}% na latência de consultas cross-shard.")
else:
    print(f" Convergência: A partição não foi melhorada (Cut={cut_inicial}).")
```

```
--- Cenario: Otimizaçao de Sharding com Grafo Ponderado ---
Número Total de Entidades (Nós): 6
Custo Máximo de Comunicação por Aresta: 10 (U2-P4 e U3-P5)
------------------------------------------------------------
Custo Inicial de Comunicação Cross-Shard: 4
Shard 1 (A): {'U1', 'U2', 'P4'}
Shard 2 (B): {'U3', 'P5', 'P6'}
------------------------------------------------------------
--- Resultados do Refinamento Kernighan-Lin ---
Novo Custo de Comunicação Cross-Shard (Final): 4
Shard Otimizado 1 (A): {'P5', 'U3', 'P6'}
Shard Otimizado 2 (B): {'P4', 'U1', 'U2'}
------------------------------------------------------------
 Convergência: A partição não foi melhorada (Cut=4).
```

CONCLUSÃO

- A aplicação do KL é vital para a eficiência de Sistemas de Uso Intensivo de Dados (como bancos de dados distribuídos, sharding e frameworks de processamento de grafos como GraphX e Pregel).
- Embora a KL pura tenha uma complexidade O(|V|^2) (na forma ingênua), ela continua sendo um pilar refinador essencial.