# Abordagens para o problema dos K-centros em grafos de diferentes tamanhos

1st Lucas Abreu Lopes
*ICEI PUCMINAS*
lalopes@sga.pucminas.br

2nd Lucas Lima Publio
*ICEI PUCMINAS*
llpublio@sga.pucminas.br

3nd Ricardo Meireles Trindade Pereira
*ICEI PUCMINAS*
ricardo.meireles@sga.pucminas.br

***Abstract*—Este estudo investiga soluções para o problema dos K-centros em grafos de diferentes tamanhos, comparando uma abordagem exata com uma heurística. Utilizamos o algoritmo de Floyd-Warshall para calcular as distâncias mínimas entre todos os pares de vértices e aplicamos uma estratégia de força bruta para encontrar a melhor combinação de centros em grafos menores. Para grafos maiores, implementamos uma heurística que seleciona centros de forma aproximada, garantindo que a solução nunca seja mais de duas vezes pior que a ideal. Nossos resultados mostram que a abordagem exata é eficaz para grafos pequenos, enquanto a heurística oferece uma boa aproximação com menor custo computacional para grafos maiores.**

***Index Terms*—K-centros, Grafos, Identificação de K-Centros, Floyd-Warshall**

## I. Introdução

O problema dos K-centros em grafos emerge como um clássico da teoria dos grafos, com aplicações em diversas áreas, desde a logística e o planejamento urbano até as redes sociais. Este trabalho se dedica a investigar técnicas para resolver este problema, focando em grafos de diferentes tamanhos e complexidades.

Primariamente, apresentamos uma abordagem que, utilizando a matriz de distâncias mínimas gerada pelo algoritmo de Floyd-Warshall, calcula as distâncias mais curtas entre todos os pares de pontos do grafo. Isso permite a determinação precisa de cada um dos K-centros de maneira a minimizar a distância máxima de todos os vértices em relação a cada centro. Contudo, essa solução se baseia em uma abordagem de ”força bruta”, comparando todas as combinações possíveis de centros, o que a torna extremamente custosa em termos computacionais.

Para grafos de grande porte, a busca por soluções exatas torna-se inviável devido à complexidade computacional exponencial. Diante disso, esse trabalho propõe uma solução heurística que utiliza uma estratégia de seleção de centros aproximada através da seleção de centros distantes dos outros. Embora essa abordagem não garanta a perfeição, ela oferece uma aproximação satisfatória - que nunca é mais de duas vezes pior que a solução ideal - com um custo computacional significativamente menor.

Ao longo deste trabalho as duas abordagens são comparadas através analisando as técnicas disponíveis para atacar o problema dos K-centros em grafos, avaliando a eficiência e a eficácia de cada uma, e propondo melhorias para a aplicação em grafos de grande porte.

## II. Metodologia

### A. Contexto

O problema dos K-centros em grafos é um desafio clássico. Investigamos a diferença de dois diferentes métodos na resolução do problema dos K-centros em grafos, comparando os seus tempos de execução utilizando a mesma base comparativa e os testando com as instâncias disponíveis na OR-Library. Mais especificamente, foram usadas as 40 instâncias para o problema das p-medianas não capacitado (que é bem semelhante ao problema dos k-centros exceto pela função objetivo). A tarefa central é selecionar K vértices em um grafo de modo que a distância máxima de qualquer vértice ao centro mais próximo seja minimizada. Na seção C. Implementações de algoritmos, serão descritos individualmente ambos os métodos utilizados.

### B. Constantes

*1) Compiladores:* Durante o estudo, não vamos considerar a influência da escolha do compilador, embora essa escolha possa afetar os resultados devido à otimização ou outras características. Para garantir a consistência, utilizaremos a JDK (versão 1.83) em conjunto com o Visual Studio Code na versão 1.90.2, que foi atualizada em 14/05/2024, em todos os casos de teste.

*2) Linguagem de Programação:* Devido às diferenças nos paradigmas de programação, no modelo de alocação de memória e outras especificidades, diferentes linguagens de programação podem resultar em tempos de execução diferentes para cada algoritmo. Para evitar tais diferenças, todos os códigos apresentados foram escritos na linguagem Java.

*3) Código:* Para encontrar a tabela de distâncias mínimas iniciais de cada vértice será utilizado o algoritmo de Floyd-Warshall com a implementação destacada a seguir.

```
public static int[][] floydWarshall(int[][]
    grafo) {
   int n = grafo.length;
   int[][] dist = new int[n][n];

   for (int i = 0; i < n; i++) {
      System.arraycopy(grafo[i], 0, dist[i],
          0, n); // Copia o grafo inicial
          para a matriz de distancias
   }
```

```
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] !=
                    Integer.MAX_VALUE &&
                    dist[k][j] !=
                    Integer.MAX_VALUE) {
                    dist[i][j] =
                        Math.min(dist[i][j],
                        dist[i][k] +
                        dist[k][j]); //
                        Atualiza a distancia
                        minima entre i e j
                }
            }
        }
    }

    return dist;
}
```

### C. Implementações de algoritmos

*1) Método Força-Bruta:* O método de força bruta gera todas as possíveis combinações de centros e calcula o raio de cada combinação, que é a maior distância mínima de qualquer vértice para o centro mais próximo. A combinação que minimiza este raio é considerada a melhor solução. Embora intensiva em termos de computação devido ao grande número de combinações possíveis, essa abordagem garante a solução ótima.

A complexidade do método é determinada pela geração de combinações e o cálculo do raio para cada combinação. A geração de todas as combinações possíveis de K centros a partir de n vértices tem complexidade O(C(n, k)), onde C(n, k) é o coeficiente binomial n!/(k!(n-k)!). Para cada combinação, o cálculo do raio percorre todos os n vértices e verifica a distância mínima até qualquer um dos k centros, resultando em uma complexidade de O(n * k). Portanto, a complexidade total do método é O(C(n, k) * n * k), sendo extremamente custosa para grandes valores de n e k, tornando a abordagem impraticável para grandes grafos.

```
private static void combinar(int[] centros,
    int pos, int start, int n, int k, int[][]
    distancias, int[] melhorCentros, int[]
    melhorRaio) {
    if (pos == k) {
        int raio = calcularRaio(centros,
            distancias);
        if (raio < melhorRaio[0]) {
            melhorRaio[0] = raio;
            System.arraycopy(centros, 0,
                melhorCentros, 0, k);
        }
        return;
    }

    for (int i = start; i < n; i++) {
        centros[pos] = i;
        combinar(centros, pos + 1, i + 1, n, k,
            distancias, melhorCentros,
            melhorRaio);
    }
}

private static int calcularRaio(int[]
    centros, int[][] distancias) {
    int n = distancias.length;
    int maxDist = 0;

    for (int i = 0; i < n; i++) {
        int minDist = Integer.MAX_VALUE;
        for (int centro : centros) {
            minDist = Math.min(minDist,
                distancias[i][centro]);
        }
        maxDist = Math.max(maxDist, minDist);
    }

    return maxDist;
}
```

*2) Método Aproximado:* O trecho de código apresentado implementa uma abordagem heurística para selecionar K vértices como centros de um grafo. A função "selecionarKVertices" começa inicializando duas estruturas: uma para armazenar as distâncias entre os vértices e outra para armazenar os centros escolhidos. Em seguida, um vértice aleatório é selecionado e seu índice é armazenado. Em cada iteração do laço principal, o vértice cujo índice se encontra armazenado é transformado em um centro e as distâncias mínimas de todos os vértices aos centros são atualizadas. O próximo centro é então escolhido através da seleção do vértice cuja distância mínima aos centros atuais é a maior.

```
static void selecionarKVertices(int[][]
    pesos, int numCentros, int numVertices) {
    Map<Integer, Integer> distancias =
        inicializarDistancias(numVertices); //
        Inicializa o mapa de distancias
    Set<Integer> centros = new HashSet<>(); //
        Conjunto para armazenar os indices dos
        vertices escolhidos como centros
    int indiceMax =
        encontrarIndiceInicial(distancias); //
        Encontra o indice inicial aleatorio

    for (int i = 0; i < numCentros; i++) {
        centros.add(indiceMax); // Adiciona o
            vertice atual ao conjunto de centros
        atualizarDistancias(distancias, pesos,
            indiceMax); // Atualiza as
            distancias ao centro mais proximo
        indiceMax =
            encontrarIndiceMaximo(distancias);
            // Encontra o proximo centro com
            maior distancia minima
    }
}
```

### D. Códigos completos no github

Os códigos completos podem ser encontrados no link a seguir, cada um nomeado adequadamente com relação ao método usado.

https://github.com/L-Lukke/TP2-Grafos

## III. Avaliação de Resultados

Devido à alta quantidade de testes, torna-se contraproducente descrever todos os testes feitos. Por isso, os testes descritos serão os realizados com os casos de teste 1, 11, 21, 31 e 39 - que são grafos que não possuem tantos centros e podem ser confortavelmente escritos neste. Alguns testes não tiveram sua execução completa devido ao seu alto custo computacional, e sua solução de maior distância mínima foi encontrada na Tabela 1 no documento Trabalho Prático N.02.

### A. Teste 1 - 200 vértices, 5 centros

*1) Método Força Bruta:* -
Combinação de centros: [4, 8, 56, 62, 77];
Maior distância entre vértice e centro: 127 unidades;
Tempo de execução: 23673 milissegundos.
*2) Método Aproximado:* -
Combinação de centros: [0, 15, 46, 62, 76];
Maior distância entre vértice e centro: 186 unidades;
Tempo de execução: 1 milissegundo.

### B. Teste 11 - 1800 vértices, 5 centros

*1) Método Força Bruta:* -
Combinação de centros: -;
Maior distância entre vértice e centro: 59 unidades;
Tempo de execução: Não foi concluído ($>$ 60 minutos).
*2) Método Aproximado:* -
Combinação de centros: [0, 25, 267, 187, 191];
Maior distância entre vértice e centro: 73 unidades;
Tempo de execução: 2 milissegundos.

### C. Teste 21 - 5000 vértices, 5 centros

*1) Método Força Bruta:* -
Combinação de centros: -;
Maior distância entre vértice e centro: 40 unidades;
Tempo de execução: Não foi concluído ($>$ 60 minutos).
*2) Método Aproximado:* -
Combinação de centros: [0, 59, 159, 355, 381];
Maior distância entre vértice e centro: 53 unidades;
Tempo de execução: 2 milissegundos.

### D. Teste 31 - 9800 vértices, 5 centros

*1) Método Força Bruta:* -
Combinação de centros: -;
Maior distância entre vértice e centro: 30 unidades;
Tempo de execução: Não foi concluído ($>$ 60 minutos).
*2) Método Aproximado:* -
Combinação de centros: [0, 274, 363, 365, 551];
Maior distância entre vértice e centro: 42 unidades;
Tempo de execução: 5 milissegundos.

### E. Teste 39 - 16200 vértices, 10 centros

*1) Método Força Bruta:* -
Combinação de centros: -;
Maior distância entre vértice e centro: 23 unidades;
Tempo de execução: Não foi concluído ($>$ 60 minutos).
*2) Método Aproximado:* -
Combinação de centros: [0, 19, 23, 77, 129, 158, 251, 726, 759, 899];
Maior distância entre vértice e centro: 35 unidades;
Tempo de execução: 7 milissegundos.

## IV. Conclusões

Este estudo comparou duas abordagens para resolver o problema dos K-centros em grafos: o método de força bruta e um método aproximado. Através de testes em grafos de diferentes tamanhos, identificamos as vantagens e limitações de cada abordagem.

O método de força bruta demonstrou ser extremamente preciso, encontrando a combinação exata de centros que minimiza a maior distância entre qualquer vértice e o centro mais próximo. No entanto, este método revelou-se computacionalmente inviável para grafos que têm tamanhos razoavelmente grandes, nesse caso com mais de 500 vértices, devido à sua complexidade exponencial, resultando em tempos de execução proibitivos para grande parte das instâncias de teste.

Por outro lado, o método aproximado se mostrou extremamente eficiente em termos de tempo de execução, conseguindo encontrar soluções em milissegundos mesmo para grafos muito grandes. Embora esta abordagem não garanta a solução ótima, ela provou ser bastante próxima do ideal, com a maior distância entre vértice e centro nunca excedendo mais que o dobro da solução exata, como evidenciado pelos testes realizados.

O Método Aproximado demonstra uma eficiência de execução significativamente superior ao Método de Força Bruta, sendo aproximadamente 99,996% mais rápido. Enquanto o Método de Força Bruta levou 23673 milissegundos para processar o teste com 200 vértices e 5 centros, o Método Aproximado concluiu a mesma tarefa em apenas 1 milissegundo. Essa enorme diferença em tempo de execução destaca a vantagem do Método Aproximado em termos de velocidade, tornando-o uma escolha muito mais prática para grandes conjuntos de dados onde a rapidez é essencial.

Assim, concluímos que para grafos de pequeno a médio porte, onde a precisão é crítica e o tempo de computação não é uma preocupação, o método de força bruta é adequado - como no exemplo de encontrar centros em Estados em um país. Para grafos de grande porte - como em grafos que mapeiam ruas em uma cidade -, e/ou nos quais o tempo de execução é um fator a ser avaliado, o método aproximado é altamente recomendado devido à sua eficiência e à qualidade satisfatória das soluções obtidas.

Futuras pesquisas podem explorar otimizações adicionais no método aproximado, buscando reduzir ainda mais a diferença entre a solução heurística e a solução exata, como poderia ser a

investigação do impacto da seleção de um vértice não-aleatório inicialmente.

## REFERENCES


[1] $https://en.wikipedia.org/wiki/Metric_k - center$

[2] Documento "Trabalho Prático N.02", apresentado pelo professor Zenilton Patrocínio

[3] $https://people.brunel.ac.uk/\ mastjjb/jeb/orlib/files/$