

Trabalho Prático 02: Problema dos k -Centros

Gabriel Tamietti Mauro Pedro Hosken

16 de novembro de 2025

Resumo

Este trabalho apresenta a implementação e análise de duas abordagens para o Problema dos k -Centros: uma solução exata (Força Bruta) e uma solução aproximada (heurística Farthest-First). Foram detalhadas as decisões de implementação em Java, incluindo o pré-processamento de dados via algoritmo de **Floyd-Warshall** e a estrutura de cada algoritmo. Os experimentos demonstram a enorme diferença de complexidade entre as duas abordagens. A solução exata, devido à sua natureza combinatória, é inviável para a maioria das instâncias, enquanto a solução aproximada manteve um desempenho extremamente rápido em todas as execuções, confirmando a eficiência de sua complexidade polinomial.

1 Introdução ao Problema

O Problema dos k -Centros busca, em um grafo com distâncias métricas, selecionar um subconjunto de k vértices (os **centros**) tal que a maior distância de qualquer outro vértice ao centro mais próximo (o **raio** da solução) seja minimizada.

2 Detalhes da Implementação e Decisões Técnicas

O projeto foi implementado em Java, focado na clareza e na fidelidade às complexidades teóricas de cada algoritmo.

2.1 Pré-processamento: Algoritmo de Floyd-Warshall

A primeira etapa crítica é a conversão do grafo de entrada, que pode ser esparsa, em um grafo completo onde as arestas representam o **menor caminho** entre todos os pares de vértices. O problema dos k -Centros exige distâncias métricas para sua formulação, o que é garantido pelo cálculo de menores caminhos. Utilizamos o algoritmo de **Floyd-Warshall** para este fim.

```

1  for (int k_fw = 0; k_fw < V; k_fw++) {
2      for (int i = 0; i < V; i++) {
3          for (int j = 0; j < V; j++) {
4              if (distancias[i][k_fw] != infinito && distancias[k_fw]
5                  ][j] != infinito) {
6                  if (distancias[i][k_fw] + distancias[k_fw][j] <
7                      distancias[i][j]) {
8                      distancias[i][j] = distancias[i][k_fw] +
9                          distancias[k_fw][j];
10                 }
11             }
12         }
13     }
14 }
```

Listing 1: Trecho de Instancia.java com Floyd-Warshall

A complexidade de $O(V^3)$ é aceitável, pois é uma etapa única de pré-processamento. Para $V \leq 900$ (maior instância), o custo é administrável antes da aplicação das heurísticas.

2.2 Solução Exata: Força Bruta

A solução exata busca o raio ótimo OPT_k testando exaustivamente todas as $\binom{V}{k}$ combinações de k centros possíveis. A complexidade total do algoritmo é dominada pelo número de combinações multiplicado pelo custo de avaliação de cada combinação, resultando em $O\left(\binom{V}{k} \cdot V \cdot k\right)$, o que é **exponencial** em V . A geração das combinações é feita recursivamente. O método `calcularRaio`, responsável por avaliar o raio de uma solução candidata, tem complexidade $O(V \cdot k)$ e é executado para cada combinação:

```

1  private long calcularRaio(int[] centros) {
2      long raioMaximoDaSolucao = 0;
3      for (int v = 0; v < V; v++) {
4          long distanciaMinimaCentro = Long.MAX_VALUE;
5          for (int centro : centros) {
6              long dist = distancias[v][centro];
7              if (dist < distanciaMinimaCentro) {
8                  distanciaMinimaCentro = dist;
9              }
10         }
11         if (distanciaMinimaCentro > raioMaximoDaSolucao) {
12             raioMaximoDaSolucao = distanciaMinimaCentro;
13         }
14     }
15     return raioMaximoDaSolucao;
16 }
```

Listing 2: Trecho de SolucaoExata.java

2.3 Solução Aproximada: Heurística Farthest-First

A classe `SolucaoAproximada.java` implementa a heurística **Farthest-First** (Primeiro-Mais-Distante), um algoritmo guloso conhecido por fornecer uma 2-

aproximação garantida para o OPT_k . A estratégia é escolher o primeiro centro arbitrariamente e, em cada um dos $k - 1$ passos seguintes, selecionar o vértice que está mais distante do conjunto de centros já escolhidos. A complexidade é $O(V \cdot k)$. Essa complexidade é mantida eficientemente pelo uso de um vetor auxiliar (`distMinimaParaCentro`), que armazena a menor distância de cada vértice para o centro mais próximo, evitando recalcular todas as distâncias em cada iteração k , como ilustrado:

```

1 for (int v = 0; v < V; v++) {
2     int distParaNovoCentro = distancias[v][novoCentro];
3     if (distParaNovoCentro < distMinimaParaCentro[v]) {
4         distMinimaParaCentro[v] = distParaNovoCentro;
5     }
6 }
```

Listing 3: Trecho de SolucaoAproximada.java

3 Resultados dos Experimentos

A Tabela 1 (Vista na última página) apresenta os resultados de raio e tempo de execução obtidos pelas duas abordagens para todas as instâncias do problema.

3.1 Análise do Desempenho

3.1.1 Contraste de Tempo de Execução

A diferença de desempenho é o resultado mais notável, e espelha diretamente a complexidade teórica dos algoritmos.

- **Solução Exata (Força Bruta):** A execução na Instância 01 ($V = 100, k = 5$), que representa $\binom{100}{5} \approx 75$ milhões de combinações, consumiu aproximadamente **35.65 segundos**. Os valores de N/A nas demais instâncias ($V > 100$ ou $k > 5$) indicam que a execução foi **interrompida** ou **omitida** devido à complexidade combinatória $\binom{V}{k}$, cujo tempo de processamento extrapolaria o limite de tempo razoável. Por exemplo, a Instância 02 ($V = 100, k = 10$) exigiria testar ≈ 17.3 bilhões de combinações, o que demonstra a inviabilidade da força bruta para o escopo do problema.
- **Solução Aproximada (Farthest-First):** O tempo de execução desta heurística é significativamente baixo, com o maior tempo registrado sendo de apenas **30.22 ms** (Instância 30: $V = 600, k = 200$). Isso confirma a eficiência da complexidade $O(V \cdot k)$ da heurística, tornando-a a única opção viável para instâncias de grande porte.

3.1.2 Análise da Qualidade da Solução

Na única instância em que ambas as soluções foram comparadas (Instância 01):

- O Raio Exato encontrado foi ****127****.
- O Raio Aproximado encontrado foi ****186****.

A diferença entre os raios demonstra o **trade-off** entre tempo e qualidade: a Força Bruta garante o raio mínimo, mas o Farthest-First, em menos de 1 milissegundo, entrega uma solução que, embora não seja a ótima, é aceitável em cenários onde a velocidade é prioritária.

4 Conclusão

O trabalho demonstrou o contraste entre a complexidade exponencial da Solução Exata (Força Bruta) e a complexidade linear $O(V \cdot k)$ da Solução Aproximada (Farthest-First). Enquanto a primeira é útil apenas para validação em problemas triviais, a heurística se provou um método escalável e eficiente, fornecendo soluções em milissegundos para todas as instâncias do problema.

Tabela 1: Resultados obtidos pelas Soluções Exata e Aproximada. Tempos em milissegundos (ms).

| Inst. | $ V $ | k | Raio Encontrado | | Tempo Exec. (ms) | |
|-------|-------|-----|-----------------|------------|------------------|------------|
| | | | Exato | Aproximado | Exato | Aproximado |
| 01 | 100 | 5 | 127 | 186 | 35650.74 | 0.4534 |
| 02 | 100 | 10 | N/A | 131 | N/A | 0.6056 |
| 03 | 100 | 10 | N/A | 154 | N/A | 0.7575 |
| 04 | 100 | 20 | N/A | 114 | N/A | 1.2184 |
| 05 | 100 | 33 | N/A | 71 | N/A | 1.5142 |
| 06 | 200 | 5 | N/A | 138 | N/A | 0.7474 |
| 07 | 200 | 10 | N/A | 96 | N/A | 1.1431 |
| 08 | 200 | 20 | N/A | 82 | N/A | 2.6592 |
| 09 | 200 | 40 | N/A | 57 | N/A | 3.0019 |
| 10 | 200 | 67 | N/A | 31 | N/A | 4.7215 |
| 11 | 300 | 5 | N/A | 73 | N/A | 0.8863 |
| 12 | 300 | 10 | N/A | 71 | N/A | 2.0020 |
| 13 | 300 | 30 | N/A | 59 | N/A | 3.3138 |
| 14 | 300 | 60 | N/A | 40 | N/A | 7.2940 |
| 15 | 300 | 100 | N/A | 25 | N/A | 6.3696 |
| 16 | 400 | 5 | N/A | 84 | N/A | 0.9112 |
| 17 | 400 | 10 | N/A | 56 | N/A | 2.3325 |
| 18 | 400 | 40 | N/A | 44 | N/A | 8.0255 |
| 19 | 400 | 80 | N/A | 28 | N/A | 10.9041 |
| 20 | 400 | 133 | N/A | 19 | N/A | 14.1082 |
| 21 | 500 | 5 | N/A | 53 | N/A | 1.5273 |
| 22 | 500 | 10 | N/A | 56 | N/A | 1.8422 |
| 23 | 500 | 50 | N/A | 34 | N/A | 10.9143 |
| 24 | 500 | 100 | N/A | 23 | N/A | 19.2929 |
| 25 | 500 | 167 | N/A | 15 | N/A | 20.0759 |
| 26 | 600 | 5 | N/A | 50 | N/A | 1.4441 |
| 27 | 600 | 10 | N/A | 43 | N/A | 3.6130 |
| 28 | 600 | 60 | N/A | 28 | N/A | 14.1766 |
| 29 | 600 | 120 | N/A | 19 | N/A | 27.3792 |
| 30 | 600 | 200 | N/A | 14 | N/A | 30.2236 |
| 31 | 700 | 5 | N/A | 42 | N/A | 1.8673 |
| 32 | 700 | 10 | N/A | 45 | N/A | 4.9255 |
| 33 | 700 | 70 | N/A | 25 | N/A | 24.2579 |
| 34 | 700 | 140 | N/A | 17 | N/A | 23.1478 |
| 35 | 800 | 5 | N/A | 38 | N/A | 2.3893 |
| 36 | 800 | 10 | N/A | 41 | N/A | 3.2490 |
| 37 | 800 | 80 | N/A | 25 | N/A | 19.8016 |
| 38 | 900 | 5 | N/A | 39 | N/A | 2.9978 |
| 39 | 900 | 10 | N/A | 35 | N/A | 4.0669 |
| 40 | 900 | 90 | N/A | 21 | N/A | 25.0004 |