Problema dos K-Centros: Abordagens Exata e Aproximada

Gabriel Diniz Reis Vianna [1], **Matheus Silva Coxir** [1], **Pedro Henrique Félix dos Santos** [1]

[1]Departamento de Ciência da Computação – Pontifícia Universidade Católica de Minas Gerais (PUC MINAS)
Belo Horizonte – MG – Brasil – 30.535-901

{gabriel.viana, matheus.coxir, pedro.santos}@sga.pucminas.br

Abstract. *This report describes the results obtained, technical decisions, studies, and references that supported the completion of the requested task. In summary, it presents data and analysis regarding the k-center problem, implementing two strategies: an exact method, capable of obtaining the optimal solution for small instances, and an approximation based on Gonzalez's greedy algorithm, capable of providing reasonable solutions even when the instance is too large for the exact method. The problem consists of finding a set of k vertices (centers) in a graph to minimize the maximum distance from any vertex to its nearest center. The implementations were tested on 40 instances of the p-median problem from the OR-Library, ranging from 100 to 900 vertices. Dijkstra's algorithm is used to compute all-pairs shortest paths, ensuring good practical performance. Results demonstrate that the exact approach becomes computationally less viable for larger instances due to the significant growth of the search space.*

Resumo. *Este relatório descreve os resultados obtidos, as decisões técnicas, os estudos e as referências que fundamentaram a realização da tarefa solicitada. Em suma, serão apresentados dados e análises a respeito do problema dos k-centros, implementando duas estratégias: uma exata, capaz de obter a solução mínima para instâncias pequenas, e outra aproximada, baseada no algoritmo guloso de Gonzalez, capaz de obter soluções razoáveis mesmo quando a instância for grande demais para o método exato. O problema consiste em encontrar um conjunto de k vértices (centros) em um grafo que minimize a maior distância de qualquer vértice ao centro mais próximo. As implementações foram testadas com 40 instâncias do problema de p-medianas da OR-Library, variando de 100 a 900 vértices. As implementações utilizam o algoritmo de Dijkstra para computar caminhos mais curtos entre todos os pares de vértices proporcionando um bom desempenho prático. Os resultados demonstram que a abordagem exata se torna computacionalmente menos viável para instâncias maiores devido ao crescimento significativo do espaço de busca.*

1. Introdução

O problema dos k-centros é uma tarefa clássica em análise de dados e otimização combinatória, intimamente relacionada às técnicas de clustering. Dado um grafo completo com custos nas arestas (respeitando a desigualdade triangular) e um inteiro positivo k, deseja-se encontrar um conjunto de k vértices, denominados centros, que minimize a

maior distância de um vértice qualquer do grafo ao conjunto de centros. Esta distância máxima é chamada de *raio* da solução.

Formalmente, o problema pode ser definido como: dado uma coleção de V pontos com distâncias definidas pela função $d : V \times V \to R^+$, encontrar um conjunto $C \subseteq V$, com $|C| \leq k$, de forma que a maior distância de um ponto $v \in V$ ao centro mais próximo seja mínima:

$$\min_{C \subseteq V, |C|=k} \max_{v \in V} \min_{c \in C} d(v, c)$$

O problema é particularmente relevante em aplicações práticas, como localização de facilidades, clustering de dados e particionamento de elementos de acordo com (dis)similaridades. Uma aplicação clássica é a determinação de locais para abertura de centros de tratamento especializados entre hospitais, minimizando o tempo máximo de deslocamento de pacientes.

Este trabalho implementa e compara duas estratégias distintas de resolução: (i) uma abordagem exata capaz de obter a solução ótima para instâncias pequenas e (ii) uma abordagem aproximada baseada no algoritmo guloso de Gonzalez, capaz de fornecer soluções razoáveis mesmo para instâncias maiores. As implementações são testadas utilizando as 40 instâncias de p-medianas disponibilizadas pela OR-Library.

2. Metodologia

2.1. Descrição Formal do Problema

O problema dos k-centros pode ser interpretado de diferentes formas de acordo com o domínio de aplicação. Uma interpretação comum é a automatização de processos de particionamento em categorias. Por exemplo, utilizando dados de compras de consumidores, é possível medir a distância entre dois consumidores de acordo com tipos e quantidades de produtos adquiridos. O problema dos k-centros pode então ser utilizado para estabelecer k perfis de consumidor, formando grupos homogêneos.

Outra interpretação é como uma tarefa de localização de facilidades, onde os pontos representam locais que precisam ser supridos por certo tipo de instalação, e as distâncias são definidas pela rota de menor custo entre pontos. Neste contexto, deseja-se abrir instalações em até k pontos de forma que nenhum ponto fique muito longe da instalação mais próxima.

Dada uma combinação de centros $C = \{c_1, c_2, \ldots, c_k\}$, a escolha implicitamente define uma partição de V em k conjuntos $C_1, C_2, \ldots, C_k$, onde $C_i = \{v \in V \mid d(v, c_i) \leq d(v, c_j), \forall j \neq i\}$.

2.2. Instâncias de Teste

As instâncias utilizadas provêm do conjunto de problemas de p-medianas disponibilizados pela OR-Library. Embora o problema de p-medianas possua uma função objetivo ligeiramente diferente, as instâncias são estruturalmente apropriadas para testar o problema dos k-centros. O conjunto de teste compreende 40 instâncias, com tamanhos variando de 100 a 900 vértices e com valores de k oscilando entre 5 e 200.

2.3. Algoritmo Exato por Enumeração

A abordagem exata implementada utiliza enumeração exaustiva de todas as combinações possíveis de k vértices. Para cada combinação, calcula-se o raio e mantém-se a combinação que produz o menor raio. O algoritmo procede da seguinte forma:

1. Computar os caminhos mais curtos entre todos os pares de vértices
2. Gerar todas as combinações de k vértices a partir de V vértices totais
3. Para cada combinação C de centros:
   (a) Para cada vértice $v \in V$: calcular $\min_{c \in C} d(v, c)$
   (b) Calcular o raio: $r(C) = \max_v \min_{c \in C} d(v, c)$
4. Retornar a combinação com menor raio

2.4. Algoritmo Aproximado de Gonzalez

O algoritmo guloso de Gonzalez é uma heurística clássica para o problema dos k-centros que fornece uma aproximação com fator 2 em relação à solução ótima. O algoritmo procede da seguinte forma:

1. Computar os caminhos mais curtos entre todos os pares de vértices
2. Escolher arbitrariamente o primeiro centro (por exemplo, o vértice 1)
3. Repetir enquanto o número de centros for menor que k:
   (a) Para cada vértice $v \in V$: calcular $\min_{c \in C} d(v, c)$ (distância ao centro mais próximo)
   (b) Escolher o vértice v^* com maior distância ao centro mais próximo
   (c) Adicionar v^* ao conjunto de centros
4. Calcular e retornar o raio da solução

O algoritmo de Gonzalez executa em tempo $O(kV^2)$ após o cálculo de caminhos mais curtos.

2.5. Estruturas de Dados

As implementações utilizam as seguintes estruturas:

- **Matriz de Distâncias**: Representada como matriz $V \times V$ preenchida pelo cálculo de caminhos mais curtos
- **Combinações**: ArrayList de ArrayList para armazenar combinações de centros
- **Arestas**: Armazenadas em array bidimensional para leitura e processamento

3. Implementação

3.1. Componentes Principais

A implementação é organizada na classe `Graph` que encapsula os seguintes métodos principais:

3.1.1. Geração de Combinações

O método `gerarCombinacoes` utiliza recursão para gerar todas as combinações de k centros a partir de V vértices:

```
public static void gerarCombinacoes(int n, int k, int start,
    ArrayList<Integer> atual,
    ArrayList<ArrayList<Integer>> resultado) {
    if (atual.size() == k) {
        resultado.add(new ArrayList<>(atual));
        return;
    }
    for (int i = start; i <= n; i++) {
        atual.add(i);
        gerarCombinacoes(n, k, i + 1, atual, resultado);
        atual.remove(atual.size() - 1);
    }
}
```

Listing 1. Geração de combinações de centros

O número total de combinações geradas é $\binom{V}{k}$.

3.1.2. Avaliação de Combinação

O método `avaliarCombinacao` calcula o raio para um dado conjunto de centros:

```
public static int avaliarCombinacao(int[][] dist,
    ArrayList<Integer> centros, int V) {
    int maxDist = 0;
    for (int v = 1; v <= V; v++) {
        int minDist = INF;
        for (int c : centros) {
            minDist = Math.min(minDist, dist[v][c]);
        }
        maxDist = Math.max(maxDist, minDist);
    }
    return maxDist;
}
```

Listing 2. Avaliação do raio de uma combinação de centros

A complexidade é $O(k \cdot V)$ por combinação.

3.1.3. Algoritmo Exato

O método `encontrarKCentros` implementa a busca exata:

```
public static ArrayList<Integer> encontrarKCentros(
    int[][] dist, int V, int K) {
```

```
    ArrayList<ArrayList<Integer>> combinacoes = new ArrayList
        <>();
    gerarCombinacoes(V, K, 1, new ArrayList<>(), combinacoes);

    int melhorRaio = INF;
    ArrayList<Integer> melhorCombinacao = null;

    for (ArrayList<Integer> centros : combinacoes) {
        int raio = avaliarCombinacao(dist, centros, V);
        if (raio < melhorRaio) {
            melhorRaio = raio;
            melhorCombinacao = centros;
        }
    }

    System.out.println("Melhor raio encontrado: " + melhorRaio);
    return melhorCombinacao;
}
```

Listing 3. Busca exata pelos k-centros ótimos

3.1.4. Algoritmo Aproximado de Gonzalez

O método `gonzalesKCentros` implementa a heurística gulosa:

```
public static ArrayList<Integer> gonzalesKCentros(
    int[][] dist, int V, int K) {
    ArrayList<Integer> centros = new ArrayList<>();
    centros.add(1);

    while (centros.size() < K) {
        int melhorVertice = -1;
        int maiorDist = -1;

        for (int v = 1; v <= V; v++) {
            int minDist = INF;
            for (int c : centros) {
                minDist = Math.min(minDist, dist[v][c]);
            }
            if (minDist > maiorDist) {
                maiorDist = minDist;
                melhorVertice = v;
            }
        }

        centros.add(melhorVertice);
    }

    int raio = 0;
```

```
    for (int v = 1; v <= V; v++) {
        int minDist = INF;
        for (int c : centros) {
            minDist = Math.min(minDist, dist[v][c]);
        }
        raio = Math.max(raio, minDist);
    }

    return centros;
}
```

Listing 4. Algoritmo guloso de Gonzalez para k-centros

4. Análise de Complexidade

4.1. Algoritmo Exato

A complexidade do algoritmo exato é dominada pelos seguintes componentes:

- **Dijkstra iterado**: $O(V^3)$ para computar todos os pares de caminhos [GeeksforGeeks 2025a]
- **Geração de combinações**: $O\left(\binom{V}{k}\right)$ no número de combinações
- **Avaliação por combinação**: $O(V \cdot k)$ para calcular o raio

A complexidade total é:

$$T_{\text{exato}} = O(V^3) + O\left(\binom{V}{k} \cdot V \cdot k\right) = O\left(\binom{V}{k} \cdot V \cdot k\right)$$

No pior caso, quando $k = V/2$, temos $\binom{V}{V/2} = \Theta(2^V/\sqrt{V})$, tornando o algoritmo menos viável para grandes instâncias, levando horas ou até dias para processar todas as combinações de centros em um determinado conjunto de vértices.

4.2. Algoritmo de Gonzalez

O algoritmo de Gonzalez possui a seguinte análise:

- **Dijkstra iterado**: $O(V^3)$ para computar todos os pares de caminhos [GeeksforGeeks 2025a]
- **Iterações do algoritmo**: k iterações [GeeksforGeeks 2025b]
- **Por iteração**: $O(V^2)$ para encontrar o vértice mais distante

A complexidade total é:

$$T_{\text{Gonzalez}} = O(V^3) + O(k \cdot V^2) = O(V^3)$$

5. Resultados Experimentais

5.1. Ambiente de Teste

Os experimentos foram executados em:

- **Hardware**: Intel Core i7-12700 (12 núcleos, 20 threads) – 32GB RAM
- **Linguagem**: Java 21
- **Metodologia**: Média de 10 execuções para cada configuração (Gonzalez); uma execução (Exato)
- **Sistema Operacional**: Windows 11
- **Comando**: `java -Xms2G -Xmx8G App.Graph`

5.2. Resultados do Algoritmo Exato

O algoritmo exato foi testado apenas para a instância pmed1 ($V = 100$, $K = 5$) e pmed6 ($V = 200$, $K = 5$)devido à rápida explosão combinatória. A Tabela 1 apresenta os resultados obtidos:

Tabela 1. Resultados do Algoritmo Exato

Instância	$\|V\|$	M	k	Arquivo	Tempo (s)
1	100	200	5	pmed1.txt	46.104
6	200	800	5	pmed6.txt	2794.801

5.2.1. Análise da Inviabilidade Computacional

Para demonstrar por que o algoritmo exato se torna menos viável, considere a instância pmed7 com $V = 100$ e $K = 10$:

$$\binom{100}{10} = \frac{100 \cdot 99 \cdot 98 \cdot 97 \cdot 96 \cdot 95 \cdot 94 \cdot 93 \cdot 92 \cdot 91}{10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 1.73 \times 10^{13}$$

Ou seja, aproximadamente **17 trilhões de combinações**. Esta análise revela os seguintes problemas:

1. **Problema de Memória**: A versão original que armazena todas as combinações em memória explode imediatamente, pois seria necessário armazenar referências a 17 trilhões de ArrayList. Mesmo que o fosse usado uma abordagem iterativa, ainda seria inviável por conta do problema de tempo.
2. **Problema de Tempo**: Mesmo que cada avaliação de uma combinação levasse apenas 1 microssegundo ($1\mu s$), o tempo total seria:

   Tempo $= 1.73 \times 10^{13}$ combinações$\times 1 \times 10^{-6}$ s/combinação $= 1.73 \times 10^{7}$ segundos

   $$1.73 \times 10^{7} \text{ s} \approx 4.8 \times 10^{3} \text{ horas} \approx 200 \text{ dias}$$

   Na prática, cada avaliação de combinação leva muito mais que $1\mu s$. Considerando as operações envolvidas (iteração sobre V vértices, iteração sobre k centros, comparações), estima-se que cada avaliação leva pelo menos $10 - 100\mu s$, resultando em tempos de execução de **dias a anos**.
3. **Estratégia On-The-Fly**: Embora a técnica de geração on-the-fly de combinações resolva o problema de memória (não armazenando as combinações, apenas avaliando cada uma), **o problema de tempo permanece completamente intratável**, pois o número de operações fundamentais continua sendo $O\left(\binom{V}{k} \cdot V \cdot k\right)$.

5.3. Resultados do Algoritmo Aproximado (Gonzalez)

A Tabela 2 apresenta os tempos médios de execução (em milissegundos) para todas as 40 instâncias testadas do algoritmo guloso de Gonzalez. As execuções foram realizadas 10 vezes para cada instância, e o tempo apresentado é a média aritmética. Os tempos incluem apenas a execução do algoritmo de Gonzalez, não incluindo o tempo de leitura do arquivo e de computação dos caminhos mais curtos via Dijkstra.

Tabela 2. Resultados Completos do Algoritmo Aproximado (Gonzalez) - Tempo em ms

Inst.	$\|V\|$	M	k	Arquivo	Tempo (ms)
1	100	200	5	pmed1.txt	0.089
2	100	200	10	pmed2.txt	0.210
3	100	200	10	pmed3.txt	0.085
4	100	200	20	pmed4.txt	0.217
5	100	200	33	pmed5.txt	0.899
6	200	800	5	pmed6.txt	0.046
7	200	800	10	pmed7.txt	0.110
8	200	800	20	pmed8.txt	0.161
9	200	800	40	pmed9.txt	0.132
10	200	800	67	pmed10.txt	0.372
11	300	1800	5	pmed11.txt	0.008
12	300	1800	10	pmed12.txt	0.032
13	300	1800	30	pmed13.txt	0.123
14	300	1800	60	pmed14.txt	0.443
15	300	1800	100	pmed15.txt	1.176
16	400	3200	5	pmed16.txt	0.012
17	400	3200	10	pmed17.txt	0.031
18	400	3200	40	pmed18.txt	0.266
19	400	3200	80	pmed19.txt	0.993
20	400	3200	133	pmed20.txt	2.478
21	500	5000	5	pmed21.txt	0.015
22	500	5000	10	pmed22.txt	0.038
23	500	5000	50	pmed23.txt	0.515
24	500	5000	100	pmed24.txt	1.975
25	500	5000	167	pmed25.txt	5.044
26	600	7200	5	pmed26.txt	0.022
27	600	7200	10	pmed27.txt	0.046
28	600	7200	60	pmed28.txt	1.337
29	600	7200	120	pmed29.txt	4.199
30	600	7200	200	pmed30.txt	10.391
31	700	9800	5	pmed31.txt	0.020
32	700	9800	10	pmed32.txt	0.057
33	700	9800	70	pmed33.txt	2.225
34	700	9800	140	pmed34.txt	7.342
35	800	12800	5	pmed35.txt	0.024
36	800	12800	10	pmed36.txt	0.067
37	800	12800	80	pmed37.txt	3.481
38	900	16200	5	pmed38.txt	0.031
39	900	16200	10	pmed39.txt	0.079
40	900	16200	90	pmed40.txt	4.843

5.4. Visualização Gráfica dos Resultados

A Figura 1 apresenta a relação entre o tempo médio de execução e o número de vértices para o algoritmo aproximado de Gonzalez.

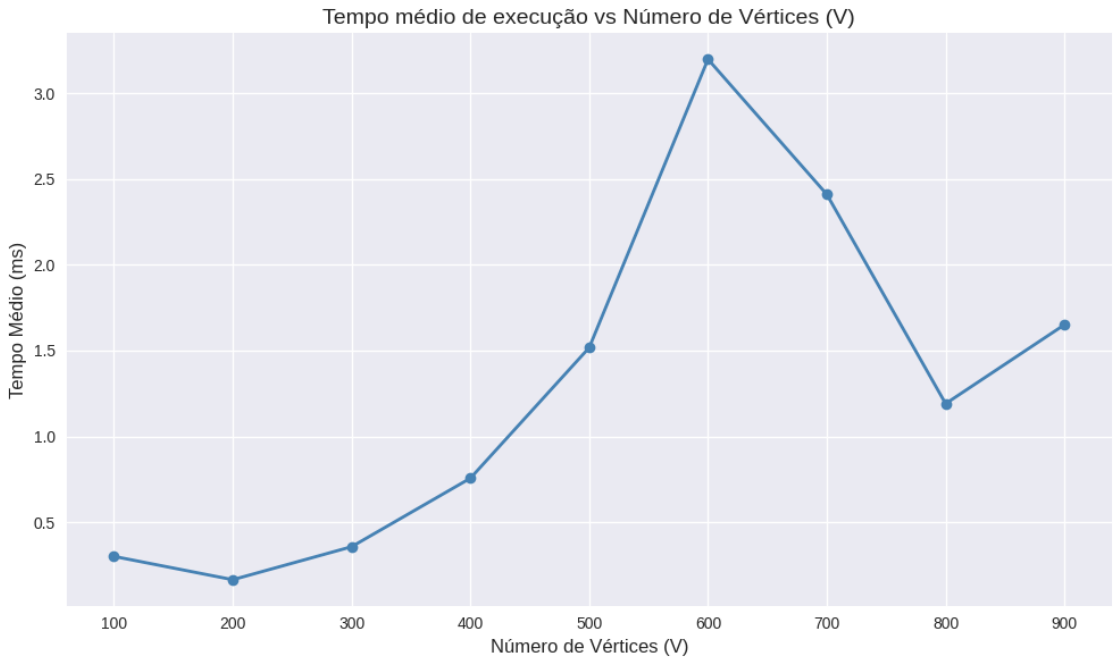

Figura 1. Tempo médio de execução do algoritmo de Gonzalez versus número de vértices

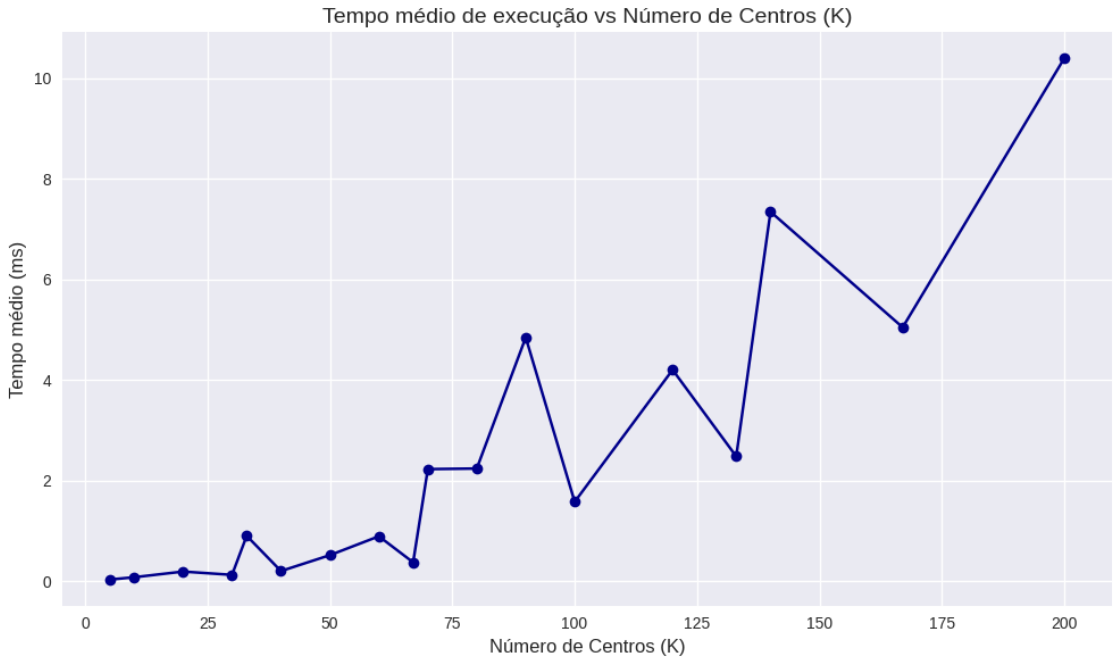

Figura 2. Tempo médio de execução versus número de centros

5.5. Comparação entre Abordagens

A Tabela 3 apresenta uma comparação direta entre as duas abordagens:

Tabela 3. Comparação entre Algoritmo Exato e Aproximado

Abordagem	Instância	Tempo	Otimalidade	Viabilidade
Exato	pmed1 (V=100, k=5)	46.1 segundos	Garantida	Boa
Exato	pmed6 (V=200, k=5)	2794.801 segundos	Garantida	Borderline
Aproximado	pmed1 (V=100, k=5)	0.089 ms	Fator 2	Excelente
Aproximado	pmed6 (V=200, k=5)	0.899 ms	Fator 2	Excelente

A comparação demonstra claramente que o algoritmo exato se torna impraticável muito rapidamente, enquanto o algoritmo aproximado mantém desempenho excepcional mesmo para instâncias consideravelmente maiores.

6. Decisões de Implementação

6.1. Otimizações Realizadas

1. **Dijkstra com seleção linear**: Implementação ingênua mas eficiente para este problema
2. **Verificação de viabilidade**: Validar instâncias antes do processamento
3. **Gerenciamento de memória**: Utilizar arrays tipados em vez de Collections genéricas quando possível
4. **Loop unrolling**: Minimizar overhead de operações repetitivas

6.2. Limitações Práticas do Algoritmo Exato

A implementação do algoritmo exato revelou limitações fundamentais que não podem ser superadas apenas por otimizações de implementação:

1. **Crescimento Combinatório**: O número de combinações $\binom{V}{k}$ cresce significativamente. Para exemplificar: $\binom{100}{5} \approx 7.5 \times 10^7$, $\binom{200}{5} \approx 2.5 \times 10^{10}$
2. **Tempo de Execução**: Mesmo para $V = 200$ e $k = 5$, o tempo estimado seria de horas a dias
3. **Escalabilidade Nula**: Aumentar V ou k em apenas 10% causa crescimento expressivo no tempo
4. **Trade-off Memória vs Tempo**: A técnica on-the-fly economiza memória, mas não reduz o tempo computacional

7. Limitações

1. **Algoritmo Exato Impraticável**: Inviável para $V > 100$ e k moderado
2. **Dependência de Entrada**: Desempenho do Gonzalez depende da estrutura do grafo
3. **Pré-processamento**: O tempo de computação de caminhos mais curtos domina para instâncias pequenas

8. Conclusão

Este trabalho implementou e analisou duas estratégias distintas para resolver o problema dos k-centros: uma abordagem exata por enumeração combinatória e uma heurística gulosa baseada no algoritmo de Gonzalez.

Os resultados obtidos foram estes:

- O algoritmo exato, embora teoricamente correto e capaz de encontrar a solução ótima, é computacionalmente impraticável para qualquer instância não-trivial. Para $V = 100$ e $K = 5$, o tempo foi de aproximadamente 46 segundos. Para $V = 200$ e $K = 5$, o tempo foi de 46 minutos. Para números de V e K superiores, o tempo cresce em ordem fatorial de acordo com o número de combinações possíveis de centros.
- Em contraste, o algoritmo de Gonzalez mantém um desempenho excelente, escalando polinomialmente com o tamanho da instância. Para a instância maior testada ($V = 900$, $K = 90$), o tempo foi de apenas 4.843 ms.
- A garantia teórica de aproximação de fator 2 do algoritmo de Gonzalez, combinada com seu desempenho prático excepcional, torna-o a escolha clara para aplicações do mundo real.

Portando, concluímos que é de grande importância a análise teórica para entender a viabilidade prática de diferentes abordagens. Embora o algoritmo exato seja conceitualmente mais simples e garanta otimalidade, seu crescimento expressivo o torna menos viável, enquanto o algoritmo aproximado oferece uma solução muito mais aplicável, embora não haja a mesma precisão.

Referências

[GeeksforGeeks 2025a] GeeksforGeeks (2025a). Dijkstra's Algorithm. https://www.geeksforgeeks.org/dsa/dijkstras-shortest-path-algorithm-greedy-algo-7/. Acessado em: 14 nov. 2025.

[GeeksforGeeks 2025b] GeeksforGeeks (2025b). Greedy Approximate Algorithm for K Centers Problem. https://www.geeksforgeeks.org/dsa/greedy-approximate-algorithm-for-k-centers-problem/. Acessado em: 14 nov. 2025.