JOGO DAS JARRAS

Lívia R. Pessamilio (202165088AC)
Gabriel O. Quaresma (202265178AC)

DESCRIÇÃO DO PROBLEMA

- Conjunto de jarros com capacidades e volumes iniciais
- Ações: encher, esvaziar, transferir (adjacentes)
- Objetivo: todas as jarras com mesmo volume QQQ

DESCRIÇÃO DO PROBLEMA

Implementar e comparar:

- Backtracking
- BFS, DFS
- Busca Ordenada (Uniform Cost)
- Busca Gulosa
- A*
- IDA*

Métricas coletadas:

- Profundidade, custo, nós visitados/expandidos, fator de ramificação, tempo

DIVISÃO DE TAREFAS

Gabriel:

- Backtracking
- A*
- IDA*
- Refatoração e testes automatizados

Lívia:

- BFS e DFS
- Busca Ordenada
- Busca Gulosa
- Redação, formatação do relatório e slides

FERRAMENTAS

- Linguagem: C++ (VS Code / GitHub)
- Entrada: definição na main com capacidade e volume inicial de cada jarro
- IDE: VSCode
- Controle de Versão: GitHub

ESTRUTURA BÁSICA

Jarros

- Classe Jar
- current_value, max_capacity, ações: fill(), empty(), transfer()

GameState

- Classe GameState
- Vetores jars, values; metadados: g_cost, f_cost, parent, index
- Métodos: to_key(), is_goal(), heuristic(), geração de filhos, impressão de estado

HEURÍSTICAS

ANTIGA

- Calculava apenas o desvio total
- Convertia esse desvio em passos pela razão com max_cap
- Subestimava casos em que um jarro isolado estava muito fora de alvo

NOVA

- Mantém o critério do desvio total (ajuste global)
- Adiciona o desvio máximo de um único jarro
- Converte ambos em passos (divisão por max_cap e arredondamento)
- Retorna o maior entre ajuste global e ajuste individual, garantindo admissibilidade e cobrindo o pior caso local e global ao mesmo tempo.

BACKTRACKING

Tabela 1: Métricas de desempenho para o algoritmo Backtracking

Métrica	3 garrafões	4 garrafões	5 garrafões
Profundidade da solução	46	414	1343
Custo da solução	144	1074	3694
Número de nós visitados	650	1637	1852
Número de nós expandidos	651	1638	1853
Fator médio de ramificação	1.280	1.468	1.125
Tempo de execução (ms)	4.574	670.445	1636.24

- Busca em profundidade sistemática
- Gera filhos por ação e jarro, evita ciclos com to_key()
- Completa, mas explode rapidamente em estados
- Métricas principais: custo alto, tempo proibitivo para n≥5

LARGURA (BSF)

- Expansão por níveis: usa fila FIFO
- Garante custo ótimo em número de passos
- Escala mal: número de nós e tempo crescem exponencialmente

Tabela 2: Métricas de desempenho para o algoritmo BFS

Métrica	**3 garrafões**	**4 garrafões**	**5 garrafões**
Profundidade da solução	6	16	11
Custo da solução	12	8	22
Número de nós visitados	43	323	2007
Número de nós expandidos	251	2756	23447
Fator médio de ramificação	4.302	7.152	10.336
Tempo de execução (ms)	0.444	5.254	53.225

PROFUNDIDADE (DFS)

Tabela 3: Métricas de desempenho para o algoritmo DFS

Métrica	3 garrafões	4 garrafões	5 garrafões
Profundidade da solução	10	10	10
Custo da solução	22	22	22
Número de nós visitados	3876	694749	69500000
Número de nós expandidos	12198	3710706	371000000
Fator médio de ramificação	3.147	5.342	5.338
Tempo de execução (ms)	60.956	18455.1	1.850.000

- Limite de profundidade usado (profundidade 10)
- Explora profundamente antes de retroceder
- Bom custo, mas número de nós e custo de recursão muito alto

ORDENADO

- Prioriza menor g_cost acumulado
- Usa deque e ordenação por custo
- Reduz estados expandidos vs. BFS, mas sem heurística não atinge qualidade ótima

Tabela 4: Métricas de desempenho para o algoritmo Busca Ordenada

Métrica	**3 garrafões**	**4 garrafões**	**5 garrafões**
Profundidade da solução	25	148	811
Custo da solução	51	292	1668
Número de nós visitados	55	179	1660
Número de nós expandidos	366	1642	22158
Fator médio de ramificação	6.636	9.168	13.348
Tempo de execução (ms)	0.672	3.581	46.162

BUSCA A*

Tabela 6: Métricas de desempenho para o algoritmo A*

Métrica	3 garrafões	4 garrafões	5 garrafões
Profundidade da solução	6	8	11
Custo da solução	12	16	22
Número de nós visitados	30	152	858
Número de nós expandidos	42	254	1361
Fator médio de ramificação	1.783	2.108	2.064
Tempo de execução (ms)	0.339	2.979	18.256

- Combina custo real (g) + estimativa (h) → f = g + h
- Garante completude e optimalidade (com heurística admissível)
- Ótimo compromisso entre tempo e qualidade de solução

BUSCA IDA*

- Iterative Deepening A*
- Limite cresce conforme podas por f > threshold
- Mantém optimalidade, mas múltiplas iterações impactam tempo

Tabela 7: Métricas de desempenho para o algoritmo IDA*

Métrica	**3 garrafões**	**4 garrafões**	**5**
Profundidade da solução	6	8	11
Custo da solução	12	16	22
Número de nós visitados	40	219	1255
Número de nós expandidos	41	220	1256
Fator médio de ramificação	1.538	1.973	2.156
Tempo de execução (ms)	3.153	34.167	370.152

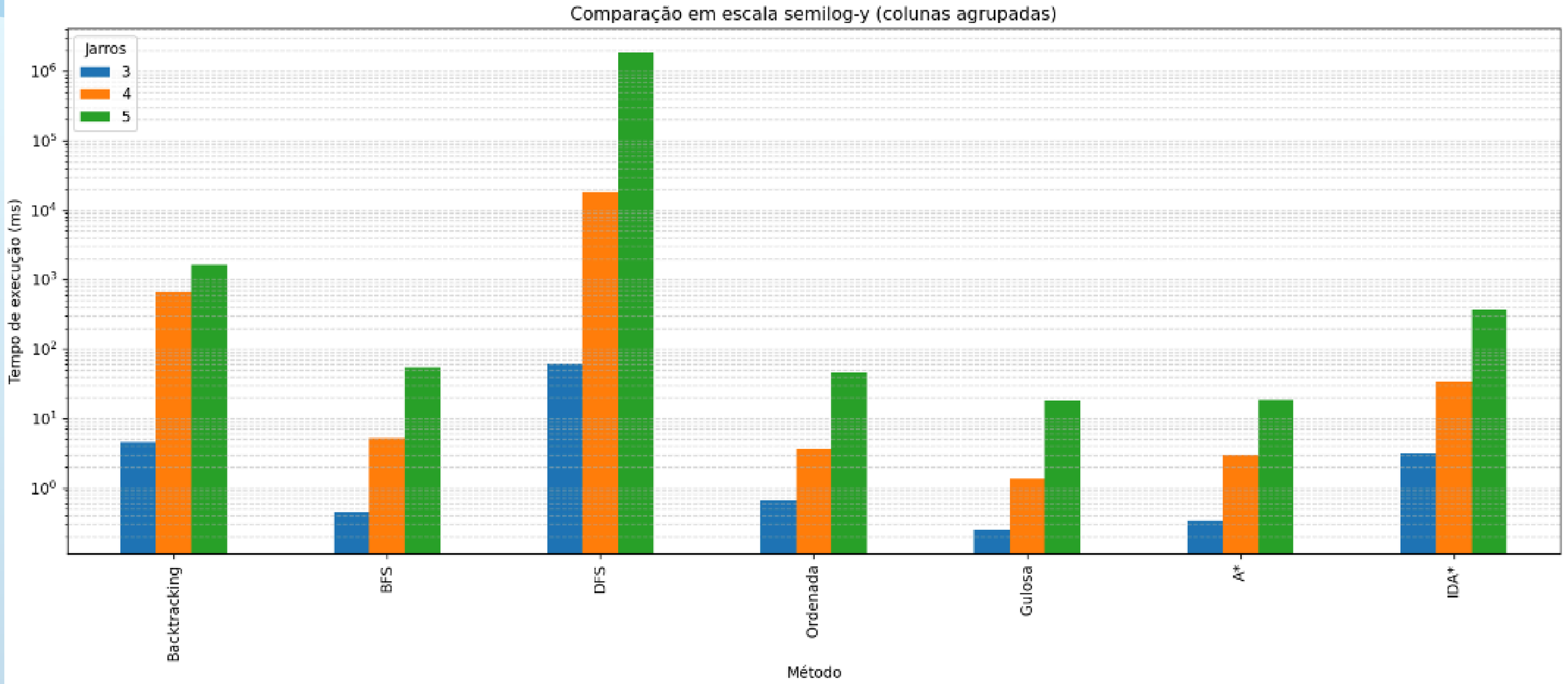

Figura 1: Gráfico do Tempo de Execução (ms)

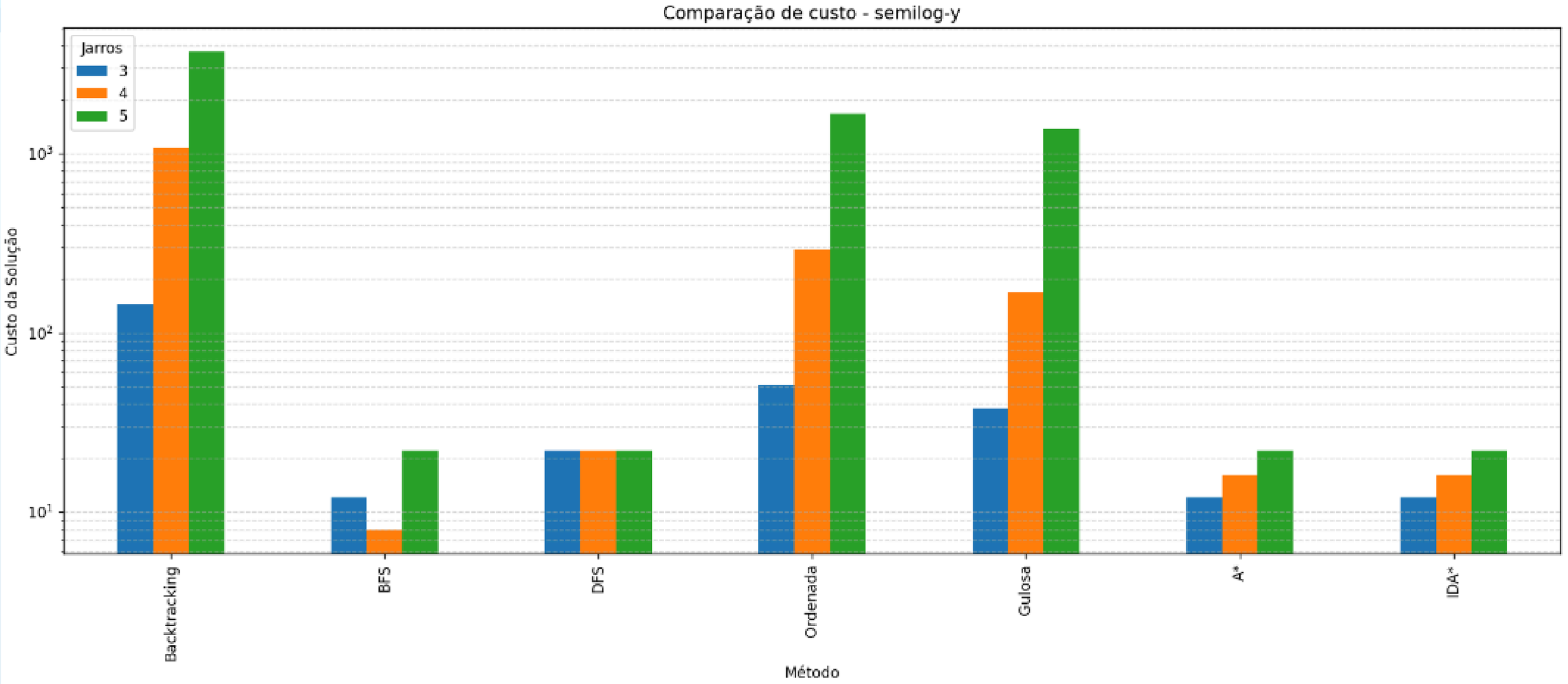

Figura 2: Gráfico do Custo da Solução

ANÁLISE E CONCLUSÃO

- Backtracking/DFS cresc. exponencial
- Busca Gulosa mais rápida
- A* e Ordenada em posições intermediárias
- BFS e A*/IDA* encontra sempre custo ótimo
- Gulosa e Ordenada sacrificam qualidade

- A* se destaca
- Tempo e Custo
- Eficiência compatível com problemas médios
- Métodos puros (Backtracking, DFS) inviáveis em escala
- Heurística proposta é admissível e bem equilibrada

OBRIGADA PELA ATENÇÃO!