Trabalho 1 - Programação Dinâmica

Eduardo Tomacheski, Gabriel Fanti e Matheus Boff
Profº Michael da Costa Móra

1. Descrição do Problema

Este relatório é dedicado ao estudo, implementação e análise do problema das Paradas Mínimas no Rally do Deserto de Dakkar, um desafio clássico de otimização gulosa (greedy) que busca determinar o menor número de paradas necessárias para completar uma jornada com restrição de distância diária. O problema modela uma situação em que um veículo percorre uma trilha linear de comprimento total L, na qual o time pode viajar, no máximo, d quilômetros por dia antes de precisar acampar. Ao longo do percurso, existem diversos pontos de parada potenciais localizados em posições conhecidas, representadas pelas distâncias x_1, x_2, …, x□ a partir do ponto inicial.

O objetivo é selecionar um subconjunto mínimo desses pontos de parada de modo que o time consiga completar todo o trajeto, respeitando o limite diário de deslocamento d. A cada novo ponto alcançado, o grupo avalia se é possível seguir até o próximo ponto antes do anoitecer; se for possível, continua; caso contrário, acampa no ponto atual. Essa estratégia define uma abordagem gulosa que sempre escolhe o ponto de parada mais distante possível dentro do limite de deslocamento diário.

A formulação do problema é relevante para o estudo de algoritmos gulosos, pois ilustra como decisões locais (parar ou seguir) podem levar a uma solução globalmente ótima. A validade dessa abordagem se deve ao fato de que, em um ambiente linear e com restrições uniformes, sempre avançar o máximo possível sem ultrapassar o limite diário minimiza o número total de paradas. Assim, o problema das Paradas Mínimas no Rally demonstra a eficiência e a correção da técnica gulosa em contextos de planejamento de trajetos e otimização de recursos limitados.

2. Algoritmo

O algoritmo desenvolvido para resolver o problema das Paradas Mínimas no Rally do Deserto de Dakkar baseia-se em uma estratégia gulosa (greedy). O princípio fundamental desse método é realizar, a cada passo, a melhor escolha local possível, com o objetivo de chegar a uma solução globalmente ótima.

Inicialmente, o conjunto de pontos de parada é ampliado para incluir o ponto de partida (posição 0) e o ponto de chegada (posição L). Em seguida, o algoritmo percorre essa

lista de pontos ordenados e verifica a distância entre o ponto atual e o próximo.

Enquanto a distância for menor ou igual à autonomia diária d, o veículo continua avançando. Quando a distância ao próximo ponto excede d, o algoritmo seleciona o ponto anterior como local de parada, pois ele representa o ponto mais distante possível alcançável dentro do limite diário. O processo é repetido até que o destino final seja atingido.

Essa abordagem elimina a necessidade de explorar múltiplas combinações ou caminhos alternativos, uma vez que a natureza linear do problema garante que sempre avançar o máximo possível sem ultrapassar o limite diário levará à menor quantidade de paradas. Dessa forma, o algoritmo apresenta um comportamento determinístico, eficiente e facilmente implementável.

3. Análise do algoritmo

A análise do algoritmo proposto para o problema das Paradas Mínimas no Rally do Deserto de Dakkar considera três aspectos principais: correção, otimalidade e complexidade computacional. Esses critérios permitem avaliar não apenas se o algoritmo produz resultados corretos, mas também se o faz de maneira eficiente e com o menor custo possível em termos de processamento.

3.1 Correção

A correção do algoritmo está fundamentada no fato de que, em cada iteração, ele verifica a distância entre o ponto atual e o próximo ponto de parada, assegurando que o limite máximo diário de deslocamento d nunca seja ultrapassado. Sempre que o próximo ponto estiver além desse limite, o algoritmo decide acampar no ponto anterior, que é o último ainda alcançável dentro da distância permitida. Esse comportamento garante que o trajeto seja completado sem violações de restrição. Além disso, como o algoritmo percorre a lista de pontos em ordem crescente, ele nunca “volta atrás” em uma decisão, o que elimina a possibilidade de inconsistência lógica. Dessa forma, o algoritmo é correto, pois sempre encontra uma sequência viável de paradas capaz de levar o grupo até o destino final.

3.2 Otimalidade

A otimalidade da solução decorre diretamente da estratégia gulosa adotada. Em cada etapa, o algoritmo seleciona o ponto de parada mais distante possível dentro do limite de deslocamento diário. Essa escolha localmente ótima é, nesse contexto, também globalmente ótima, pois o problema possui uma estrutura linear e restrições homogêneas. Avançar o máximo possível em cada etapa implica que nenhuma parada antecipada é necessária, reduzindo o número total de acampamentos. Se existisse uma solução com menos paradas, ela necessariamente envolveria ultrapassar o limite diário em algum ponto — o que é inviável. Portanto, o algoritmo é ótimo no sentido de que sempre encontra a solução com o menor número possível de paradas válidas.

3.3 Complexidade

A análise de complexidade mostra que o algoritmo é altamente eficiente. A cada passo, ele compara a distância entre o ponto atual e o próximo, realizando apenas uma única varredura sobre a lista de pontos de parada. Assim, sua complexidade temporal é O(n), onde n representa o número total de pontos de parada disponíveis. Nenhuma estrutura de dados adicional é necessária além da lista de pontos visitados e das paradas selecionadas, o que resulta em uma complexidade espacial O(k), sendo k o número de paradas efetivamente escolhidas. Essa eficiência faz com que o algoritmo seja escalável mesmo para trajetos longos, com centenas ou

milhares de pontos potenciais.

4. Implementação

```
const { performance } = require("perf_hooks");
function rallyStops(L, d, stops) {
  const points = [0, ...stops, L];
  const chosenStops = [];
  let current = 0;

  for (let i = 1; i < points.length; i++) {
    if (points[i] - points[current] > d) {
      if (i - 1 === current) {
           throw new Error("Impossível completar o rally com a distância
máxima diária.");
      }
      chosenStops.push(points[i - 1]);
      current = i - 1;
    }
  }
  return chosenStops;
}
function gerarPontosAleatorios(L, n) {
  const stops = [];
  for (let i = 0; i < n; i++) {
    stops.push(Math.random() * L);
  }
  return stops.sort((a, b) => a - b);
}
function testarCenarios() {
  const cenarios = [
    { L: 100, d: 20, n: 10 },
    { L: 1000, d: 50, n: 100 },
    { L: 5000, d: 100, n: 500 },
    { L: 10000, d: 200, n: 1000 },
    { L: 50000, d: 500, n: 5000 },
    { L: 100000, d: 800, n: 10000 },
  ];

    console.log("⏱️ Análise de Desempenho do Algoritmo das Paradas
Mínimas\n");

  cenarios.forEach((cenario, idx) => {
    const { L, d, n } = cenario;
```

```
    const stops = gerarPontosAleatorios(L, n);

    const start = performance.now();
    const result = rallyStops(L, d, stops);
    const end = performance.now();

    const tempo = (end - start).toFixed(4);
    console.log(`Cenário ${idx + 1}:`);
    console.log(`  ➤ L = ${L}, d = ${d}, pontos = ${n}`);
    console.log(`  ➤ Paradas escolhidas: ${result.length}`);
    console.log(`  ➤ Tempo de execução: ${tempo} ms`);
    console.log("-----------------------------------------");
  });
}

testarCenarios();
```

5. Tempo de execução

A fim de avaliar o desempenho do algoritmo desenvolvido, foram realizados múltiplos testes experimentais considerando diferentes cenários de percurso. Cada cenário variou o comprimento total da trilha (L), a distância máxima percorrida por dia (d) e o número de pontos de parada disponíveis (n).

Durante os testes, o algoritmo foi executado diversas vezes, e o tempo total de execução foi medido utilizando a função performance.now(), que fornece uma medição precisa em milissegundos. Essa análise permitiu verificar o comportamento temporal do algoritmo à medida que o número de pontos de parada aumenta, confirmando a expectativa teórica de complexidade linear O(n).

A Figura abaixo apresenta a saída obtida no terminal após a execução do script de testes. Nela, é possível observar, para cada cenário, os seguintes dados:

- O comprimento total da trilha (L);
- A distância máxima diária permitida (d);
- A quantidade de pontos de parada considerados (n);
- O número total de paradas selecionadas pelo algoritmo;
- O tempo de execução correspondente, em milissegundos.

Os resultados evidenciam que, mesmo em cenários com milhares de pontos de parada, o algoritmo mantém tempos de execução extremamente baixos (na ordem de milissegundos), demonstrando sua eficiência e escalabilidade. Além disso, a relação linear entre o número de pontos e o tempo de execução confirma empiricamente a análise de complexidade apresentada anteriormente.

Em síntese, os testes experimentais reforçam que o algoritmo proposto não apenas produz soluções corretas e ótimas, mas também o faz com excelente desempenho computacional, sendo plenamente adequado para aplicações em contextos de planejamento e otimização de rotas.

README.md M | index.js U | Relatorio T2 - POA.pdf U

index.js > testarCenarios

```
27  }
28
29  function testarCenarios() {
30    const cenarios = [
31      { L: 100, d: 20, n: 10 },
32      { L: 1000, d: 50, n: 100 },
33      { L: 5000, d: 100, n: 500 },
34      { L: 10000, d: 200, n: 1000 },
35      { L: 50000, d: 500, n: 5000 },
36      { L: 100000, d: 800, n: 10000 },
```

PROBLEMAS | SAÍDA | CONSOLE DE DEPURAÇÃO | TERMINAL | PORTAS

```
@ETomacheski ➜/workspaces/t2-poa (main) $ node index.js
Análise de Desempenho do Algoritmo das Paradas Mínimas

Cenário 1:
  L = 100, d = 20, pontos = 10
  Paradas escolhidas: 6
  Tempo de execução: 0.0409 ms
-----------------------------------------
Cenário 2:
  L = 1000, d = 50, pontos = 100
  Paradas escolhidas: 24
  Tempo de execução: 0.0356 ms
-----------------------------------------
Cenário 3:
  L = 5000, d = 100, pontos = 500
  Paradas escolhidas: 54
  Tempo de execução: 0.0979 ms
-----------------------------------------
Cenário 4:
  L = 10000, d = 200, pontos = 1000
  Paradas escolhidas: 52
  Tempo de execução: 0.7290 ms
-----------------------------------------
Cenário 5:
  L = 50000, d = 500, pontos = 5000
  Paradas escolhidas: 101
  Tempo de execução: 0.7296 ms
-----------------------------------------
Cenário 6:
  L = 100000, d = 800, pontos = 10000
  Paradas escolhidas: 126
  Tempo de execução: 0.9098 ms
-----------------------------------------
```

6. Conclusão

O estudo do problema das Paradas Mínimas no Rally do Deserto de Dakkar permitiu compreender e demonstrar, de forma prática, a eficiência da abordagem gulosa na resolução de problemas de otimização com restrições lineares. O algoritmo desenvolvido mostrou-se capaz de determinar o menor número possível de paradas válidas, sempre respeitando o limite máximo diário de deslocamento, e sua análise confirmou tanto a correção quanto a otimalidade da solução. Além disso, os experimentos realizados comprovaram o excelente desempenho computacional do método, que apresentou comportamento linear em relação ao número de pontos de parada e tempos de execução extremamente reduzidos. Dessa forma, conclui-se que o algoritmo proposto é uma solução simples, eficiente e escalável, ilustrando com clareza o poder e a aplicabilidade da técnica gulosa em cenários reais de planejamento de trajetos e otimização de recursos.