Problema do Carteiro Chinês

Aplicações com Diferentes Restrições

Otavio Bettega – 2025.2

Contents

1 Introdução

O **Problema do Carteiro Chinês** (PCC) é um problema clássico da teoria de grafos: dado um grafo conexo, queremos encontrar um percurso fechado que percorra todas as arestas ao menos uma vez, com custo total mínimo.

Quando o grafo é Euleriano (todos os vértices têm grau par), existe um circuito Euleriano que percorre cada aresta exatamente uma vez. Quando há vértices de grau ímpar, é necessário duplicar algumas arestas para tornar o grafo Euleriano, e o problema passa a ser:

> Como duplicar arestas do grafo original de forma a torná-lo Euleriano, minimizando o custo adicional?

Neste projeto foram implementadas três variantes:

- **Caso 1**: Grafo não ponderado (todas as arestas com peso 1).
- **Caso 2**: Grafo ponderado, minimizando o custo total.
- **Caso 3**: Grafo ponderado com **restrição de custo** K e objetivo de **maximizar o número de arestas distintas visitadas**.

Todas as implementações foram feitas em Python usando `networkx`, assumindo grafos simples, não direcionados, sem laços múltiplos na entrada. Os exemplos de grafos utilizados, e as implementaçoes podem ser encontradas no repositório de GitHub seguinte:

`https://github.com/OtavioBettega/Projeto-Carteiro-Chines-Matematica-Discreta-2025.2`

2 Caso 1 — Grafo Não Ponderado

2.1 Definição

No Caso 1 o grafo é não ponderado: todas as arestas têm peso 1. O objetivo é:

> Encontrar um percurso que visite todas as arestas, minimizando o número de repetições (arestas duplicadas).

Equivalente a minimizar o número de arestas adicionadas para tornar o grafo Euleriano.

2.2 Algoritmo

O algoritmo segue a formulação clássica do PCC:

1. Identificar os vértices de grau ímpar.
2. Calcular, para cada par de vértices ímpares, a **distância mínima em número de arestas** (via BFS) e o caminho mínimo correspondente.
3. Construir um grafo completo K sobre os vértices ímpares, onde o peso de (u, v) é a distância mínima $d(u, v)$.

4. Encontrar um **emparelhamento perfeito de custo mínimo** em K. Na implementação, isso é feito transformando $d(u, v)$ em $-d(u, v)$ e aplicando `max_weight_matching`.

5. Para cada par emparelhado, duplicar as arestas do caminho mínimo correspondente no multigrafo.

6. Extrair um circuito Euleriano do multigrafo resultante com `networkx.eulerian_circuit`.

2.3 Relação com o Caso 2

Se, no algoritmo ponderado (Caso 2), todas as arestas tiverem peso 1, então:

$$\text{Dijkstra} = \text{BFS}, \qquad \text{minimizar custo} = \text{minimizar número de arestas duplicadas.}$$

Portanto, o Caso 1 é um caso particular do algoritmo ponderado com pesos unitários.

3 Caso 2 — Grafo Ponderado e Minimização de Custo

3.1 Descrição

Agora cada aresta (u, v) possui um peso positivo $w(u, v)$, interpretado como *custo*. O objetivo é:

> Encontrar um passeio fechado que percorra todas as arestas ao menos uma vez, minimizando a soma dos pesos percorridos.

Este é o **Problema do Carteiro Chinês ponderado**. A entrada é um grafo simples não direcionado; internamente, usamos um `MultiGraph` apenas para representar arestas duplicadas.

3.2 Etapas do algoritmo e ligação com o código

O procedimento implementado em `chinese_postman_weighted` é:

1. Para cada **componente conexa** da entrada:
   (a) Copiar a componente para um subgrafo *sub*.
   (b) Construir um `MultiGraph` M com as arestas de *sub*, preservando os pesos originais.
   (c) Identificar os vértices de grau ímpar em *sub*.
2. Se não houver vértices ímpares, M já é Euleriano; basta extrair o circuito.
3. Se houver vértices ímpares:
   (a) Para cada vértice ímpar u, executar `single_source_dijkstra` em *sub*, obtendo distâncias $d(u, v)$ e caminhos mínimos até os outros vértices ímpares v.
   (b) Construir um grafo completo K sobre o conjunto de vértices ímpares, usando peso

$$w'(u, v) = -d(u, v),$$

de forma que maximizar o peso em K equivale a minimizar a soma das distâncias reais.

(c) Calcular um **emparelhamento perfeito de peso máximo** em K com `max_weight_matching`.

(d) Para cada par emparelhado (a, b), recuperar o caminho mínimo correspondente em *sub* e duplicar suas arestas em M com o mesmo peso original.

4. Depois das duplicações, todos os vértices de M têm grau par; logo, M é Euleriano. Extrai-se um circuito Euleriano com `networkx.eulerian_circuit`.

5. O custo total da solução é a soma dos pesos das arestas percorridas neste circuito.

3.3 Comentários de implementação

Na implementação final, dois detalhes garantem correção:

- A entrada é assumida como `networkx.Graph` simples. O `MultiGraph` é usado só internamente para permitir múltiplas cópias da mesma aresta.
- Ao somar o custo do circuito Euleriano em M, usamos a versão com chaves:

```
for u, v, key in nx.eulerian_circuit(M, keys=True):
    w = M[u][v][key]['weight']
```

Assim, o peso lido corresponde exatamente à aresta específica percorrida, mesmo quando existem arestas paralelas entre u e v.

4 Algoritmos Gulosos

Um **algoritmo guloso (greedy)** segue a regra:

> Em cada passo, escolher a melhor opção disponível localmente, esperando que isso leve a uma boa solução global.

Nem sempre isso produz a solução ótima, mas em muitos problemas complexos (frequentemente NP-difíceis), heurísticas gulosas fornecem soluções razoavelmente boas, com custo computacional baixo. O Caso 3 é exatamente desse tipo: o problema é difícil, então adotamos uma estratégia gulosa simples e eficiente.

5 Caso 3 — Maximizar Arestas Distintas Sob Limite de Custo

5.1 Formulação do problema

Neste caso, o objetivo muda. Dado um grafo conexo com pesos positivos $w(e)$ nas arestas e um orçamento $K > 0$, queremos:

> Encontrar um passeio (não necessariamente Euleriano) cujo custo total seja $\leq K$ e que maximize o número de arestas distintas visitadas.

Formalmente, se $E_{\text{visitadas}}$ é o conjunto de arestas distintas percorridas e o passeio não repete arestas, queremos:

$$\max |E_{\text{visitadas}}| \quad \text{sujeito a} \sum_{e \in E_{\text{visitadas}}} w(e) \leq K.$$

Este problema está relacionado ao *orienteering problem* e variantes de passeio máximo com restrição de custo, que são NP-difíceis. Por isso, não é viável buscar uma solução ótima exata em instâncias grandes; usamos uma heurística gulosa.

5.2 Por que o PCC clássico não é adequado aqui

O algoritmo do PCC ponderado não serve diretamente porque:

- Ele **obriga** a visitar todas as arestas.
- Para tornar o grafo Euleriano, o PCC **duplica arestas**, aumentando o custo.
- Aqui, queremos justamente o contrário: **evitar duplicações** e aceitar não visitar todas as arestas, desde que respeitemos K.

5.3 Heurística gulosa implementada (de acordo com o código)

A função `chinese_postman_max_edges_under_cost(G, K)` implementa a seguinte estratégia:

1. Verificar que G é conexo e que todas as arestas possuem um peso (`"weight"`). Caso contrário, é lançada uma exceção.
2. Inicializar variáveis para guardar a **melhor solução**: número máximo de arestas distintas visitadas, custo total da melhor solução e o caminho correspondente.
3. Para **cada vértice** s do grafo como possível ponto inicial:
   (a) Definir o vértice atual como $current = s$.
   (b) Iniciar:
   $$used\_edges = \emptyset, \quad total\_cost = 0, \quad path = [s].$$
   (c) Repetir:
      i. Construir uma lista de **todas** as arestas (u, v) ainda não utilizadas (nem como (u, v) nem como (v, u)), junto com seus pesos.
      ii. Ordenar essa lista por peso crescente.
      iii. Percorrer a lista ordenada e escolher a **primeira aresta** (u, v) tal que:
         - a aresta é **incidente** ao vértice atual ($u = current$ ou $v = current$); e
         - $total\_cost + w(u, v) \leq K$.
      iv. Se existir tal aresta:
         - marcar a aresta como usada (adicionando (u, v) a `used_edges`);
         - atualizar $total\_cost \leftarrow total\_cost + w(u, v)$;
         - mover o vértice atual para a outra ponta da aresta;

- acrescentar o novo vértice ao final de `path`.

v. Se **nenhuma** aresta satisfizer as condições, encerrar o laço: não é possível adicionar arestas sem ultrapassar K.

(d) Ao terminar a construção a partir de s, comparar $|used\_edges|$ com o melhor valor encontrado até então. Se for maior, atualizar a melhor solução com este caminho, custo e número de arestas distintas.

4. Após testar todos os vértices iniciais, devolver o melhor caminho encontrado e estatísticas com:

   - número de arestas distintas visitadas;
   - custo total;
   - valor de K.

Essa heurística é puramente local: em cada passo, ela tenta usar a *aresta mais barata disponível* a partir do vértice atual, enquanto o orçamento permitir. A variação por vértice inicial ajuda a escapar de escolhas ruins causadas por um único começo.

5.4 Comentários e possíveis melhorias

Algumas melhorias naturais, que não foram implementadas para manter o código simples:

- Em vez de varrer *todas* as arestas a cada passo, é possível considerar apenas as arestas incidentes a `current`, reduzindo a complexidade.
- Poderíamos permitir deslocamentos por arestas já usadas (pagando o custo) para alcançar arestas não usadas mais distantes, usando Dijkstra a partir do vértice atual.
- Em grafos grandes, heurísticas mais sofisticadas (GRASP, *simulated annealing*, etc.) poderiam explorar melhor o espaço de soluções, ao custo de maior tempo de execução.

Mesmo assim, a estratégia atual é fácil de entender, rápida e suficiente para ilustrar a diferença entre o PCC clássico e um problema de maximização sob restrição de custo.

6 Implementação e Exemplos

Nesta seção, utilizaremos os arquivos contendo os grafos de exemplo, encontrados no repositório, para aplicar nossos algoritmos concretamente. Note que temos, para cada caso, os mesmos grafos utilizados, somente acrescentando informações quando necessário. Assim, serão demonstrados os mesmos três grafos como exemplos, tendo eles complexidades crescentes. Desta forma, podemos nao só vizualizar cada grafo, mas também analisar as consequências de cada algorítmo, ou seja, o custo adicional causado pelo caminho criado.

6.1 Utilizando Caso 1

Na versão não ponderada (Caso 1), todas as arestas têm peso 1.

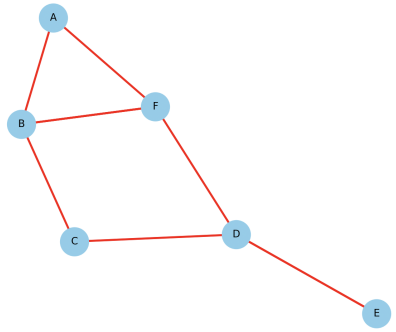

Figure 1: Caso 1 — Exemplo Simples

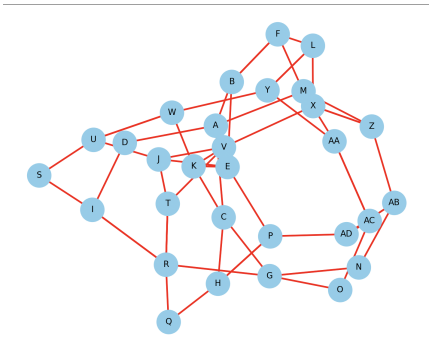

Figure 2: Caso 1 — Exemplo Médio

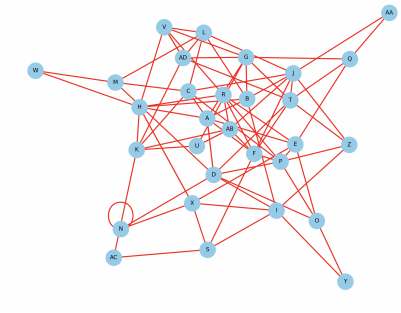

Figure 3: Caso 1 — Exemplo Complexo

Resultado e Caminho (Saída do Terminal)

Exemplo simples:

```
Caminho 1: A -> F -> B -> F -> D -> E -> D -> C -> B -> A

Estatísticas:
  original_edges: 7
  total_traversed_edges: 9
  repeated_edges: 2
```

Exemplo médio:

```
Caminho 1: A -> M -> AA -> AC -> AD -> AB -> AD -> P -> H -> Q -> R -> T -> V -> X -> Z -> AB -> N
-> G -> O -> AC -> AA -> Y -> L -> X -> Z -> M -> F -> B -> F -> L -> Y -> W -> U -> W -> K -> E
-> P -> H -> C -> K -> V -> J -> U -> S -> I -> R -> I -> D -> A -> C -> G -> R -> T -> J -> E
-> B -> A

Estatísticas:
  original_edges: 47
  total_traversed_edges: 56
  repeated_edges: 9
```

Exemplo complexo:

```
Caminho 1:A -> F -> AB -> U -> AB -> R -> V -> AD -> V -> L -> AD -> K -> U -> G -> AD -> T -> Z
-> P -> R -> H -> X -> S -> AC -> K -> AB -> J -> T -> Q -> AA -> J -> Z -> I -> Y -> O -> P -> F
-> J -> L -> M -> W -> H -> X -> I -> S -> F -> P -> D -> O -> E -> R -> F -> T -> G -> Q -> E
-> X -> N -> N -> D -> I -> B -> L -> C -> M -> D -> T -> G -> V -> H -> K -> C -> P -> E -> A
-> D -> A -> H -> B -> J -> C -> A -> G -> B -> A

Estatísticas:
  original_edges: 77
  total_traversed_edges: 83
  repeated_edges: 6
```

6.2 Utilizando Caso 2

Na versão ponderada (Caso 2), cada aresta possui um peso diferente e buscamos minimizar o custo total do percurso.

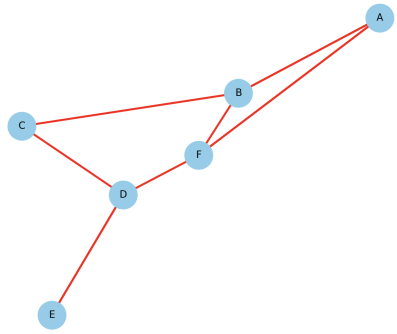

Figure 4: Caso 2 — Exemplo Simples

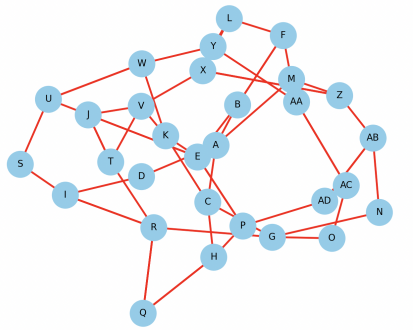

Figure 5: Caso 2 — Exemplo Médio

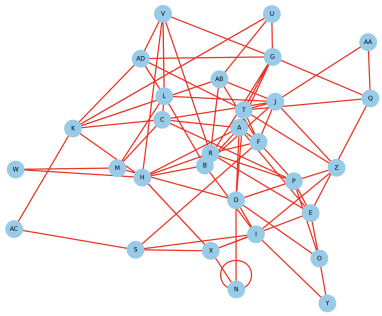

Figure 6: Caso 2 — Exemplo Complexo

Resultado e Caminho

Exemplo simples:

```
Caminho 1: A -> F -> D -> E -> D -> C -> B -> F -> A -> B -> A

Estatísticas:
  original_edges: 7
  original_cost: 223.0
  total_cost: 252.0
  extra_cost: 29.0
```

Exemplo médio:

```
Caminho 1: A -> M -> Z -> X -> Z -> AB -> AD -> AC -> AA -> Y -> L -> Y -> W -> U -> S -> I -> R
-> Q -> H -> Q -> R -> T -> V -> J -> T -> J -> U -> W -> K -> V -> X -> L -> F -> B -> F -> M
-> AA -> AC -> O -> G -> N -> AB -> AD -> P -> E -> P -> H -> C -> G -> R -> I -> D -> A -> C
-> K -> E -> J -> E -> B -> A

Estatísticas:
  original_edges: 47
  original_cost: 548.0
  total_cost: 642.0
  extra_cost: 94.0
```

Exemplo complexo:

```
Caminho 1: A -> F -> AB -> U -> K -> U -> G -> AD -> V -> L -> V -> R -> P -> Z -> T -> AD
-> L -> AD -> K -> AC -> S -> X -> H -> R -> AB -> K -> AB -> J -> T -> Q -> AA -> J -> Z
-> I -> Y -> O -> P -> D -> P -> F -> J -> L -> M -> W -> H -> X -> I -> S -> F -> T -> G
-> Q -> E -> O -> D -> I -> B -> L -> C -> P -> E -> R -> F -> A -> E -> X -> N -> N -> D
-> T -> G -> V -> H -> K -> C -> M -> D -> A -> H -> B -> J -> C -> A -> G -> B -> A
Estatísticas:
  original_edges: 77
  original_cost: 33045.0
  total_cost: 34881.0
  extra_cost: 1836.0
```

6.3 Utilizando Caso 3

Neste caso, queremos maximizar o número de arestas distintas percorridas, respeitando um limite de custo total K.

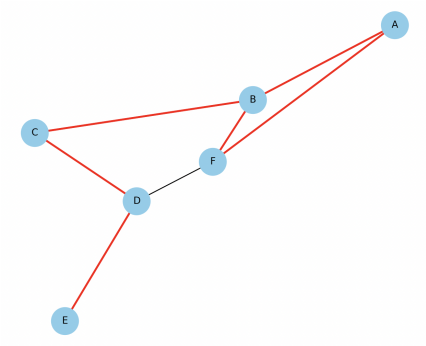

Figure 7: Caso 3 — Exemplo Simples, K=200

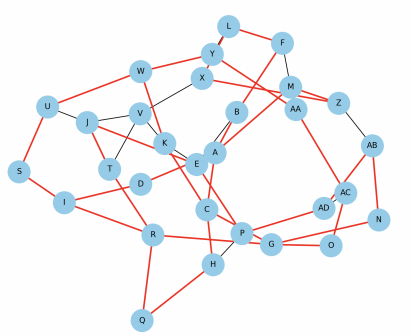

Figure 8: Caso 3 — Exemplo Médio, K=400

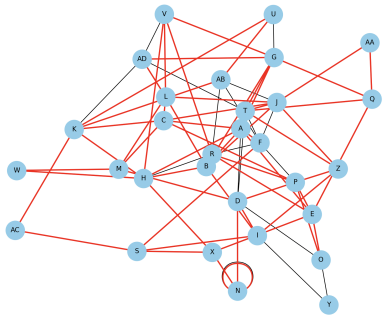

Figure 9: Caso 3 — Exemplo Complexo, K=22000

Resultado e Caminho (Saída do Terminal)

Exemplo simples:

```
Caminho 1: E -> D -> C -> B -> A -> F -> B

Estatísticas:
  arestas_distintas: 6
  custo_total: 124.0
  K_limite: 200.0
```

Exemplo médio:

```
Caminho 1:I -> R -> Q -> H -> C -> G -> N -> AB -> AD -> P -> E -> J -> T -> R -> G -> O -> AC
-> AA -> Y -> W -> K -> C -> A -> B -> F -> L -> X -> Z -> M -> A -> D -> I -> S -> U -> W

Estatísticas:
  arestas_distintas: 34
  custo_total: 314.0
  K_limite: 400.0
```

Exemplo complexo:

```
Caminho 1: K -> U -> AB -> K -> AC -> S -> F -> A -> E -> P -> C -> A -> B -> G -> V -> L -> AD
-> G -> T -> J -> Z -> T -> Q -> AA -> J -> L -> M -> C -> K -> H -> V -> R -> E -> Q -> G -> A
-> H -> W -> M -> D -> I -> S -> X -> N -> N -> D -> P -> O -> E -> X -> H -> B -> J -> C -> L
-> B -> I -> Z -> P -> R

Estatísticas:
  arestas_distintas: 59
  custo_total: 21705.0
  K_limite: 22000.0
```

7 Conclusão

Os três casos permitem explorar diferentes aspectos do Problema do Carteiro Chinês e de otimização em grafos:

- O **Caso 1** mostra o PCC em grafos não ponderados, onde o objetivo é minimizar o número de arestas duplicadas.

- O **Caso 2** generaliza para grafos ponderados, usando distâncias mínimas entre vértices ímpares e emparelhamento perfeito mínimo para minimizar o custo total do passeio.
- O **Caso 3** muda completamente a natureza do problema, introduzindo um orçamento de custo e objetivo de maximizar arestas distintas, o que leva a um problema NP-difícil e à necessidade de heurísticas gulosas.

Com isso, temos uma base prática para tratar tanto o PCC clássico quanto variantes mais realistas com restrições de custo, abrindo espaço para aplicações em planejamento de rotas, inspeção de redes e sistemas de transporte, como até possíveis percursos a serem tomados no mapeamento de ruas do Google Maps Street View.

8 Bibliografia

References

[1] Kenneth Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education.

[2] Jayme Luiz Szwarcfiter. *Grafos e Algoritmos Computacionais*. Editora Campus.

[3] Suffolk Maths. *The Chinese Postman Problem*. Disponível em: `https://www.suffolkmaths.co.uk/pages/Maths%20Projects/Projects/Topology%20and%20Graph%20Theory/Chinese%20Postman%20Problem.pdf`. Acesso em: Novembro 2025.

[4] Technical University of Munich (TUM). *Directed Chinese Postman Problem*. Disponível em: `https://algorithms.discrete.ma.tum.de/graph-algorithms/directed-chinese-postman/index_en.html`. Acesso em: Novembro 2025.

[5] GeeksForGeeks. *Chinese Postman (Route Inspection) — Introduction*. Disponível em: `https://www.geeksforgeeks.org/dsa/chinese-postman-route-inspection-set-1-introduction/`. Acesso em: Novembro 2025.

[6] Wolfram Demonstrations Project. *Chinese Postman Problem*. Disponível em: `https://demonstrations.wolfram.com/ChinesePostmanProblem`. Acesso em: Novembro 2025.

[7] Reducible (YouTube). *The Chinese Postman Problem*. Vídeo disponível em: `https://www.youtube.com/watch?v=wScPsS5oTm4`. Acesso em: Novembro 2025.