Problema do Carteiro Chinês (PCC)

Disciplina: Projeto e Análise de Algoritmos

Apresentadores: Gilson Inácio da Silva • Ederson Manoel de Oliveira

Agenda

1. Contexto e motivação
2. Definição formal
3. Ideia do algoritmo
4. Complexidade
5. Exemplo didático
6. Caso real (ruas)
7. Execução (CLI) e resultados
8. Conclusões e referências

1) Contexto e motivação

- Problemas de **cobertura de arestas**: varredura de ruas, coleta de lixo, inspeção de redes, entrega postal.
- Objetivo do PCC: encontrar um **circuito fechado** que **percorra todas as arestas** de um grafo com **custo total mínimo**.
- Quando o grafo já é **euleriano** (todos graus pares), basta o **circuito de Euler** (Hierholzer).

2) Definição formal

- **Entrada:** grafo **não dirigido**, **conexo**, ponderado (G=(V,E,w)).
- **Saída:** um **circuito** que inicia/termina no mesmo vértice e cobre todas as arestas (com repetições mínimas), minimizando (\sum w(e)).
- **Observação:** se o grafo tem vértices de **grau ímpar**, é necessário **duplicar** algumas arestas (por caminhos mínimos) para tornar o grafo euleriano.

3) Ideia do algoritmo (ótimo)

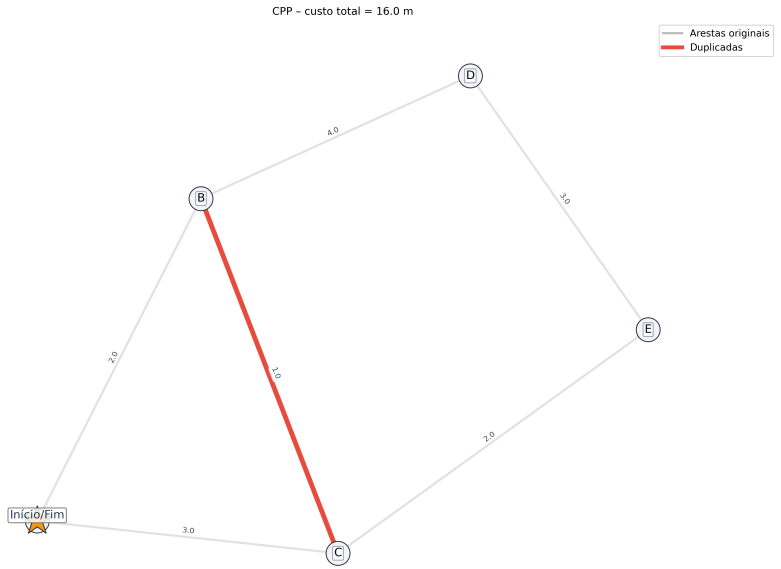
1. **Verificar graus** dos vértices; coletar o conjunto (O) de vértices **ímpares**.
2. Calcular **distâncias mínimas** entre todos os pares de (O) (Dijkstra sobre (G)).
3. Resolver um **pareamento perfeito mínimo** sobre (O) (usamos **DP com bitmask**).
4. **Duplicar** as arestas dos **caminhos mínimos** correspondentes ao pareamento.
5. No grafo resultante (todos graus pares), obter **circuito de Euler** (Hierholzer).
6. O circuito encontrado é **ótimo** para o PCC **não dirigido**.

4) Complexidade (resumo)

- Dijkstra (todas-origens restrito aos ímpares): (O(|O| \cdot (|E| \log |V|))) com heap.
- Pareamento mínimo via **DP bitmask**: (O(2^{|O|} \cdot |O|)) — viável quando (|O|) é pequeno (na prática costuma ser).
- Hierholzer: (O(|E|)).
- Na prática, o gargalo é o **pareamento**. Para instâncias grandes, alternativas: Blossom / Edmonds ou heurísticas.

5) Exemplo didático (A–E)

Exemplo didático A–E (não é bairro real); arestas com peso.



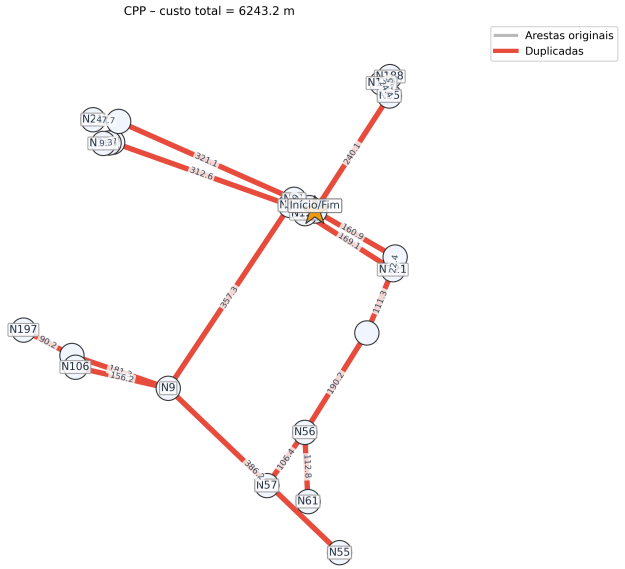
Passos (alto nível):

- Ímpares → pareamento mínimo → duplicações destacadas
- Circuito de Euler no multigrafo resultante

6) Caso real (ruas/OSM)

- Subgrafo de vias obtido do **OpenStreetMap** (licença **ODbL**).
- Pipeline: **GeoJSON → CSV (u,v,w)** com `tools/geojson_to_csv.py` → execução do PCC → tour.

Bairro Jardins (Aracaju). Figura consolidada (real_solution.png). Versão com mapa é opcional via alvo `real-basemap`.



7) Execução (linha de comando)

Instalar dependências

```
pip install -r requirements.txt
```

Exemplo didático (conservador)

```
# Linux / macOS
PYTHONPATH=src python -m pcc.solve_cli --input data/example_edges.csv --plot \
  --save-plot out/example.png --save-tour out/example_tour.txt

# Windows (PowerShell)
$env:PYTHONPATH="src"
python -m pcc.solve_cli --input data\example_edges.csv --plot \
  --save-plot out\example.png --save-tour out\example_tour.txt
```

7) Execução — caso real (conservador)

```
# Linux / macOS
PYTHONPATH=src python -m pcc.solve_cli --input data/real_edges.csv --nodes data/real_nodes.csv \
  --largest-component --plot \
  --save-plot out/real_solution.png --save-tour out/real_tour.txt

# Windows (PowerShell)
$env:PYTHONPATH="src"
python -m pcc.solve_cli --input data\real_edges.csv --nodes data\real_nodes.csv \
  --largest-component --plot \
  --save-plot out\real_solution.png --save-tour out\real_tour.txt
```

Observação: os comandos acima usam apenas flags essenciais garantidas pela CLI.

8) (Opcional) Caso real — conversão GeoJSON → CSV e execução

```
# Converter GeoJSON de ruas para CSV (u,v,w) e CSV de nós (id,lat,lon)
python tools/geojson_to_csv.py data/osm_subgraph.geojson data/real_edges.csv --snap-m 12 --nodes-out data/real_nodes.csv

# Resolver o PCC no dataset real
# Linux / macOS
export PYTHONPATH=src
python -m pcc.solve_cli --input data/real_edges.csv --nodes data/real_nodes.csv --largest-component --plot \
  --save-plot out/real_solution.png --save-tour out/real_tour.txt \
  --save-geojson out/real_tour.geojson --save-gpx out/real_tour.gpx

# Windows (PowerShell)
$env:PYTHONPATH="src"
python tools\geojson_to_csv.py data\osm_subgraph.geojson data\real_edges.csv --snap-m 12 --nodes-out data\real_nodes.csv

python -m pcc.solve_cli --input data\real_edges.csv --nodes data\real_nodes.csv --largest-component --plot \
  --save-plot out\real_solution.png --save-tour out\real_tour.txt \
  --save-geojson out\real_tour.geojson --save-gpx out\real_tour.gpx
```

Saídas típicas (figuras e trilhas): `out/real_solution.png`, `out/real_tour.gpx`, `out/real_tour.geojson`.

9) Geração dos slides (PDF)

- As imagens usadas nos slides ficam em `slides/img/`. Os alvos `plot` e `real` já geram e copiam essas imagens.

```
# Windows (PowerShell)
./make.ps1 plot    # gera out/example.png e copia para slides/img/
./make.ps1 real    # gera out/real_solution.png, GeoJSON/GPX, e copia para slides/img/
./make.ps1 slides  # exporta PDF com --allow-local-files
```

```
# Linux / macOS
make plot
make real
npx @marp-team/marp-cli slides/seminario.md -o slides/seminario.pdf --allow-local-files
```

10) Conclusões

- PCC **não dirigido** possui solução **ótima** via pareamento mínimo + Euler.
- É aplicável em logística urbana, manutenção e inspeção.
- Código **leve** em Python, sem dependências pesadas, com CLI e visualização.

Referências

- Kwan, M. (1962). *Graphic Programming Using Odd or Even Points*.
- Edmonds, J. (1965). *Paths, Trees, and Flowers*.
- Tarjan, R. (1973). *Complexity of Network Problems*.
- **OpenStreetMap** & **ODbL**: https://www.openstreetmap.org / https://opendatacommons.org/licenses/odbl/

Fim

Dúvidas?