Otimização por Colônia de Formigas no Problema do Caixeiro Viajante

Estudo de Caso: Instância Berlin52

Michael Y. Todoroki

Engenharia de Computação
Centro Federal de Educação Tecnológica de Minas Gerais
Programação em Python

Dezembro de 2025

Roteiro da Apresentação

1. O Problema e o Cenário
2. Metodologia: Ant Colony Optimization
3. Conclusão

O Problema do Caixeiro Viajante (TSP)

O Desafio:

- Visitar N cidades exatamente uma vez.
- Retornar à cidade de origem.
- Objetivo: **Minimizar a distância total.**

Complexidade:

- Problema *NP-Hard* (Não-polinomial).
- Complexidade Fatorial $O(n!)$.
- Métodos de força bruta são inviáveis para $N > 20$.

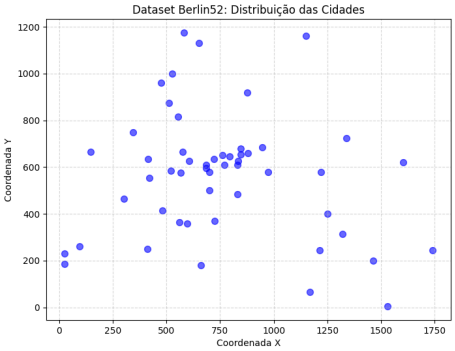
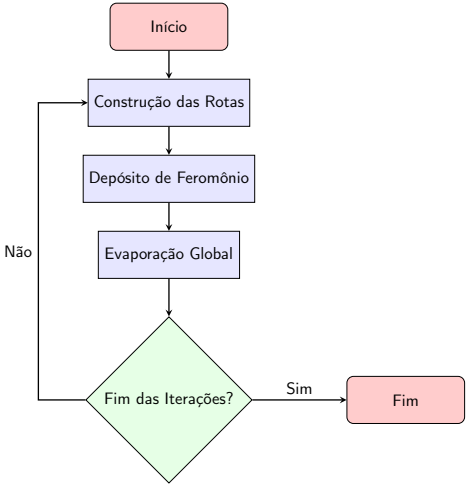


Figura: Dataset Berlin52: 52 locais em Berlim. Note a distribuição irregular (aglomerados e vazios).

O Algoritmo: Ciclo de Vida da Formiga



1. Escolha da Próxima Cidade: A probabilidade de ir de i para j:

$$P_{ij} = \frac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{k \in \mathcal{N}} (\tau_{ik})^{\alpha} \cdot (\eta_{ik})^{\beta}}$$

- τ (Feromônio): Rastro histórico.
- η (Visibilidade): $1/d_{ij}$ (inverso da distância).
- $\mathcal{N}$: Conjunto de cidades não visitadas.

2. Atualização do Rastro:

- **Depósito:** $\tau_{ij} \leftarrow \tau_{ij} + Q/L_{rota}$
- **Evaporação:** $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$

Detalhes da Implementação (Python)

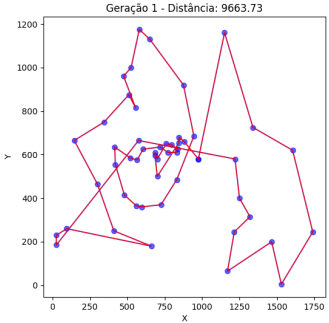
O projeto foi desenvolvido focando em performance e modularização.

- **Uso de Matrizes (NumPy):**
  - Matriz de Distâncias (52×52) pré-calculada para acesso $O(1)$.
  - Matriz de Feromônios atualizada via operações vetoriais (evitando loops lentos).

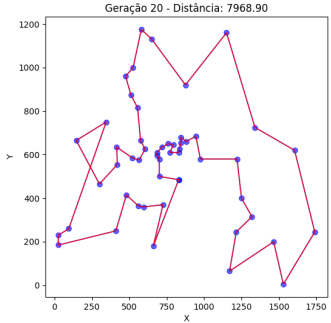
- **Parâmetros Configuradas:**

Parâmetro	**Valor**
Formigas (m)	52 (Uma por cidade)
Iterações	100
Peso da Visibilidade (β)	5.0 (Prioridade Heurística)
Evaporação (ρ)	0.5 (Esquecimento rápido)

Evolução: Fase Inicial (O Caos e a Ordem)



Geração 1
(Dist: 9663)

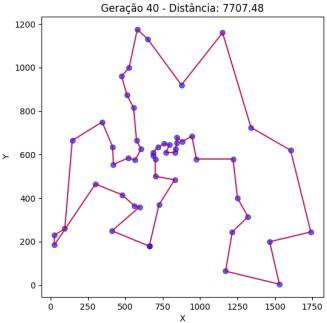


Geração 20
(Dist: 7968)

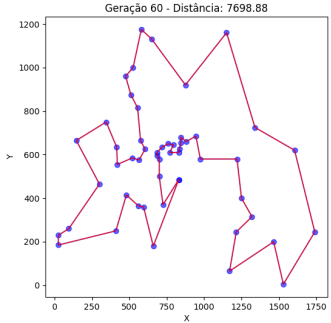
Análise:

- **Exploração:** Na G1, sem feromônio, as formigas chutam caminhos aleatórios (muitos cruzamentos).
- **Emergência:** Na G20, o feedback positivo age rápido: a "forma de C"já aparece.

Evolução: Fase Intermediária (Consolidação)



Geração 40
(Dist: 7707)

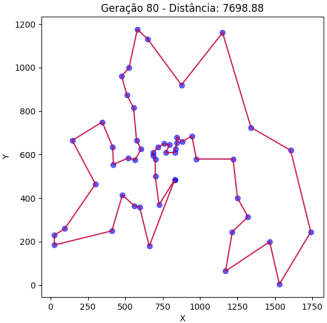


Geração 60
(Dist: 7698)

Análise:

- **Estabilidade:** A rota principal se firma. O algoritmo para de buscar caminhos radicalmente novos.
- **Limpeza:** O foco muda para eliminar pequenos laços e cruzamentos locais no centro do mapa.

Evolução: Fase Final (Otimização Fina)



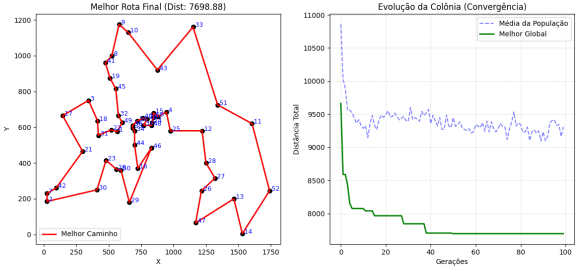
Geração 80
(Dist: 7698)

Geração 100
(Final: 7698)

Análise:

- **Convergência:** As mudanças visuais cessam, indicando um mínimo local forte.
- **Precisão:** A rota final é limpa, com erro de apenas $\approx 2\%$ do ótimo mundial (7542).

Análise de Convergência



Interpretação:

1. **Queda Rápida:** A heurística visual ($\beta = 5$) resolve a estrutura macro rapidamente.
2. **Diversidade:** A linha azul (Média) oscila acima da verde (Melhor), indicando que a colônia não estagnou.

Resultado Final

Encontrado: **7698.88**
Ótimo Conhecido: **7542**
Erro Relativo: $\approx$ **2%**

Conclusão

- **Objetivo Alcançado:** O algoritmo ACO solucionou uma instância complexa do TSP com alta precisão e baixo custo computacional.
- **Eficiência de Implementação:**
  - A utilização da linguagem Python combinada com bibliotecas científicas atendeu aos requisitos de desempenho.
  - O código foi estruturado de forma modular, facilitando a manutenção e leitura.
- **Aprendizado Principal:** Inteligência emergente (formigas simples + regras locais) é capaz de resolver problemas de otimização globais complexos.

Obrigado! Perguntas?