

Algoritmos Evolutivos: Otimizando Soluções Inspiradas na Natureza

Explorando técnicas de otimização que imitam a evolução natural para resolver problemas complexos do mundo moderno

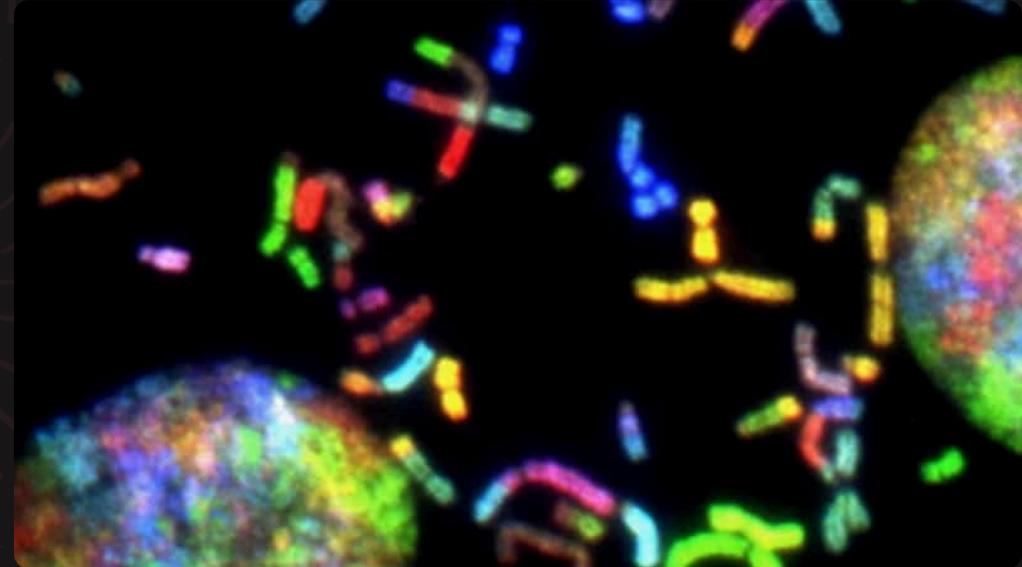


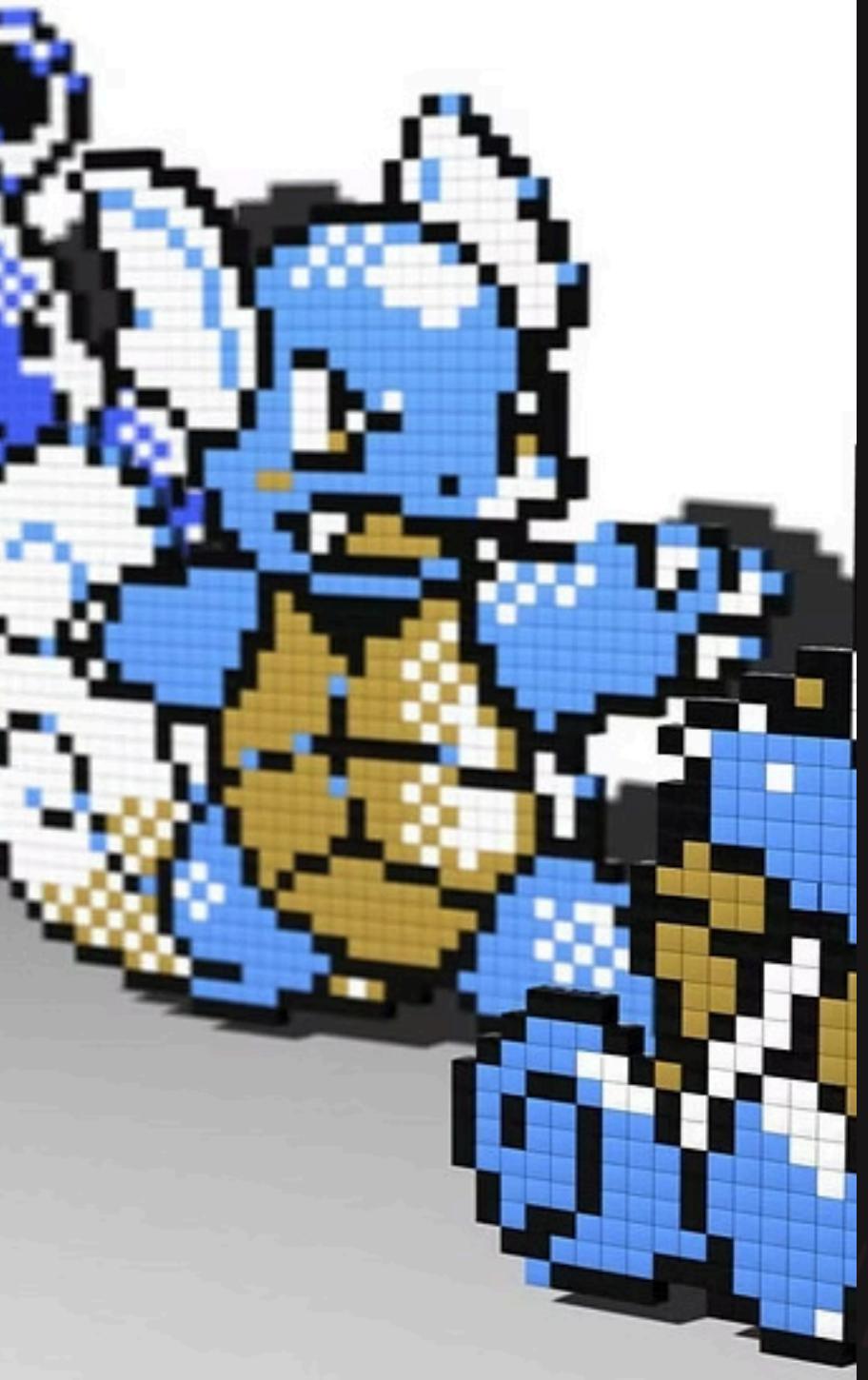
O que são Algoritmos Evolutivos?

Algoritmos evolutivos são técnicas computacionais poderosas inspiradas nos processos da evolução biológica e seleção natural de Darwin. Eles funcionam como uma simulação digital da natureza, onde soluções "competem" para sobreviver.

Através de ciclos iterativos de seleção, mutação e recombinação, esses algoritmos exploram o espaço de soluções possíveis, identificando progressivamente candidatos cada vez melhores.

São particularmente valiosos para problemas complexos de otimização onde métodos tradicionais falham ou são computacionalmente inviáveis.





Base Biológica e Conceitos Fundamentais



Cromossomos e Genes

Representam soluções candidatas codificadas. Cada cromossomo contém genes que definem características específicas da solução.



Função Fitness

Medida quantitativa da qualidade de cada solução. Determina quais indivíduos têm maior probabilidade de sobreviver e reproduzir.



Operadores Evolutivos

Seleção natural escolhe os melhores, mutação introduz variabilidade, e cruzamento combina características de soluções parentais.

Principais Tipos de Algoritmos Evolutivos

1

Algoritmos Genéticos (AG)

Baseados em representações binárias ou de valores reais, utilizam operadores clássicos de cruzamento e mutação para evoluir populações de soluções.

- Codificação flexível de soluções
- Cruzamento de um ou múltiplos pontos
- Mutação bit a bit ou gaussiana

2

Evolução Diferencial (ED)

Especializado em otimização contínua, usa diferenças vetoriais entre membros da população para gerar novos candidatos. Simples e altamente eficaz.

- Operador de mutação diferencial
- Estratégias de cruzamento binomial
- Excelente para espaços contínuos

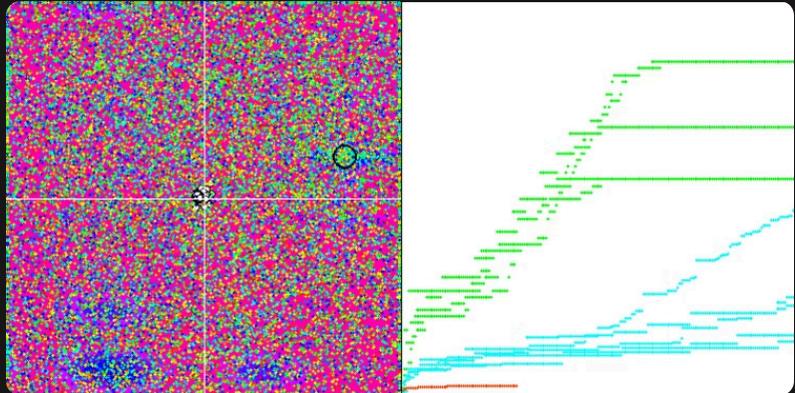
3

Programação Evolutiva (PE)

Evolui tanto parâmetros quanto estruturas de soluções. Muito popular em aprendizado de máquina para otimização de arquiteturas neurais.

- Mutação adaptativa de estruturas
- Sem operador de cruzamento
- Foco em auto-adaptação

Evolução Diferencial: Destaque e Aplicações



□ **Criado em 1995** por Rainer Storn e Kenneth Price, a Evolução Diferencial rapidamente se tornou uma das técnicas mais respeitadas em otimização contínua.

Por que a ED se destaca?

Reconhecida mundialmente pela sua robustez, simplicidade de implementação e desempenho excepcional em problemas de otimização não linear e multidimensional.

Aplicações em Destaque

- **Projeto de filtros digitais:** otimização de coeficientes para resposta ideal
- **Redes neurais:** treinamento e arquitetura de modelos deep learning
- **Sistemas de reservatórios:** gestão otimizada de recursos hídricos
- **Engenharia elétrica:** design de circuitos e sistemas de potência
- **Problemas financeiros:** otimização de portfólios de investimento

Algoritmos Genéticos: História e Funcionamento



Os Algoritmos Genéticos são especialmente poderosos para **problemas combinatórios** como o clássico problema do caixeiro viajante, onde encontrar a rota ótima entre múltiplas cidades se torna exponencialmente complexo com métodos tradicionais.

Aplicações Práticas dos Algoritmos Evolutivos

Arquitetura Inteligente

Geração automática de layouts otimizados para espaços residenciais e comerciais, considerando luz natural, fluxo e eficiência.



Robótica

Controle e navegação autônoma

Inteligência Artificial

Treinamento de redes neurais profundas e ajuste automático de hiperparâmetros para máximo desempenho.



Visão Computacional

Reconhecimento de padrões



Engenharia

Design estrutural otimizado



Bioinformática

Análise de sequências genéticas

Exemplo Prático: Implementação em Python

Pipeline de Otimização com DEAP

A biblioteca DEAP (Distributed Evolutionary Algorithms in Python) oferece uma estrutura completa para implementar algoritmos evolutivos de forma eficiente e modular.

Componentes Principais

- **Gerenciamento de populações:** criação e evolução de indivíduos
- **Operadores evolutivos:** seleção, cruzamento e mutação personalizáveis
- **Avaliação de fitness:** funções objetivo flexíveis
- **Estatísticas e logs:** monitoramento da evolução

Integração com Deep Learning

Combinando DEAP com **PyTorch** e **TorchVision**, podemos criar sistemas que otimizam automaticamente arquiteturas de redes neurais e hiperparâmetros.

Parâmetros otimizáveis: taxa de aprendizado, momentum, batch size, número de camadas, neurônios por camada, função de ativação.

 Código completo disponível em:

https://github.com/otacs-dev/Trabalho_AlgoritimosEvolucionarios

```
# Define fitness e indivíduo
creator.create("FitnessMax",
    base.Fitness, weights=(1.0,))
creator.create("Individual",
    list, fitness=creator.FitnessMax)
```

```
# Operadores
toolbox = base.Toolbox()
toolbox.register("mutate",
    tools.mutGaussian,
    mu=0, sigma=1, indpb=0.2)
toolbox.register("select",
    tools.selTournament,
    tournsize=3)
```

```
# Evolução
pop = toolbox.population(n=100)
algorithms.eaSimple(
    pop, toolbox,
    cxpb=0.7, mutpb=0.2,
    ngen=50
)
```

Benefícios e Desafios dos Algoritmos Evolutivos

✓ Vantagens Decisivas

- **Adaptabilidade excepcional:** funcionam em diversos tipos de problemas sem modificações estruturais profundas
- **Escape de mínimos locais:** população diversa permite exploração ampla do espaço de busca
- **Simplicidade conceitual:** fácil entendimento e implementação dos princípios básicos
- **Paralelização natural:** avaliações de fitness podem ser distribuídas eficientemente
- **Sem derivadas:** não requerem gradientes ou continuidade da função objetivo

⚠ Desafios a Considerar

- **Custo computacional:** milhares de avaliações podem ser necessárias para convergência
- **Parametrização delicada:** taxas de mutação, cruzamento e tamanho da população afetam muito o desempenho
- **Convergência lenta:** em alguns problemas, métodos tradicionais podem ser mais rápidos
- **Garantias teóricas limitadas:** dificuldade em provar convergência para o ótimo global
- **Ajuste problema-específico:** representação e operadores devem ser adaptados ao domínio

"O sucesso dos algoritmos evolutivos anos." reside no equilíbrio entre exploração de novas soluções e exploração das melhores já encontradas - um princípio que a natureza aperfeiçoou ao longo de milhões de



Algoritmo Genético e A* para Otimização de Rotas

Uma solução híbrida que combina busca informada com evolução artificial para resolver problemas complexos de roteirização com recompensas variáveis

O Desafio: TSP com Recompensas Variáveis

Este projeto implementa uma variação sofisticada do Problema do Caixeiro Viajante (TSP), onde o agente não apenas busca o caminho mais curto, mas também maximiza recompensas coletadas.

Objetivo principal: Encontrar a sequência ótima de visita a 10 pontos de surto de Pokémon, retornando ao início, maximizando a razão Recompensa/Custo.

O sistema considera terrenos com custos variáveis (Planície(1), Água(3), Montanha(7), Floresta(2), Lava(10) e recompensas diferenciadas por ponto.



Modelagem do Terreno e Custos com A*

01

Geração do Mapa

Grid 2D de 50×50 células com tipos de terreno distribuídos aleatoriamente

02

Atribuição de Custos

Planície (1), Água (3), Floresta (2), Montanha (7), Lava (10)

03

Função de Avaliação

$f(n) = g(n) + h(n)$, onde $g(n)$ é custo acumulado e $h(n)$ é heurística Manhattan

04

Pré-Cálculo da Matriz

A* executado N^2 vezes (100 execuções) entre todos os pares de pontos



Algoritmo A*: Busca Informada Eficiente

Função de Avaliação

$$|x_1 - x_2| + |y_1 - y_2|$$

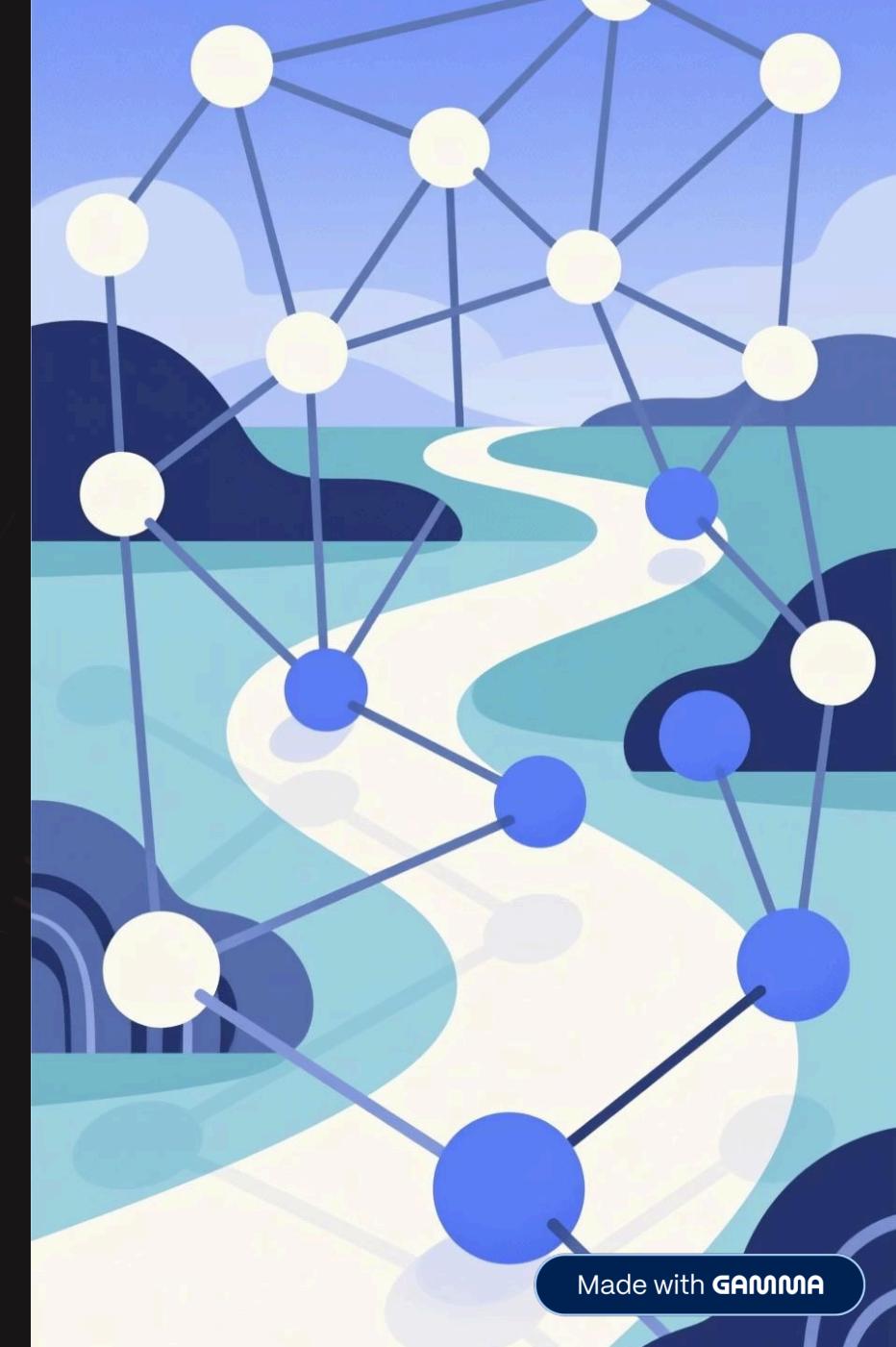
g(n) - Custo Real: Soma acumulada dos custos de terreno de cada célula percorrida desde o início.

h(n) - Heurística: Distância de Manhattan até o destino, estimando o custo restante apenas com movimentos ortogonais.

Por que é Admissível?

A Distância de Manhattan nunca superestima o custo real, pois considera apenas a distância pura, enquanto o custo real inclui penalidades de terreno (sempre ≥ 1).

Isso garante que o A* encontrará o caminho de menor custo, sendo muito mais eficiente que métodos de busca cega como BFS, (Breadth-First Search - Busca em Largura).



Algoritmo Genético: Evolução de Rotas



Sistema de Recompensas: Pokémon Shiny e Alpha

Probabilidades Base

- **Shiny:** $1/128.49 \approx 0.78\%$
- **Alpha:** 2% de qualquer captura
- **Tentativas/Outbreak:** 19 capturas

Probabilidades Conjuntas

- **Alpha Shiny:** 0.0156% (Valor: 1000)
- **Shiny Comum:** 0.763% (Valor: 100)
- **Alpha Comum:** 2% (Valor: 20)
- **Comum:** 97.22% (Valor: 0)

Valor Esperado por Outbreak

$$E_{tentativa} = \sum(P_i \times V_i)$$

Etentativa ≈ 1.3183

Recompensa Base (Rbase):

$$R_{base} = E_{tentativa} \times 19 \approx 25.04$$

Este valor é multiplicado por fatores de importância ($2.1\times$ a $19.5\times$) para criar diversidade nas recompensas.

Visualização e Resultados

Mapa Animado (Esquerda)

Grid colorido por tipo/custo de terreno com legenda completa

Pontos de surto marcados com Pokébolas e valor de recompensa (R: valor)

Animação do percurso com custo dinâmico atualizado em tempo real

Gráfico de Convergência (Direita)

Linha temporal mostrando melhor Fitness por geração do AG

Demonstra convergência evolutiva ao longo de 300 gerações

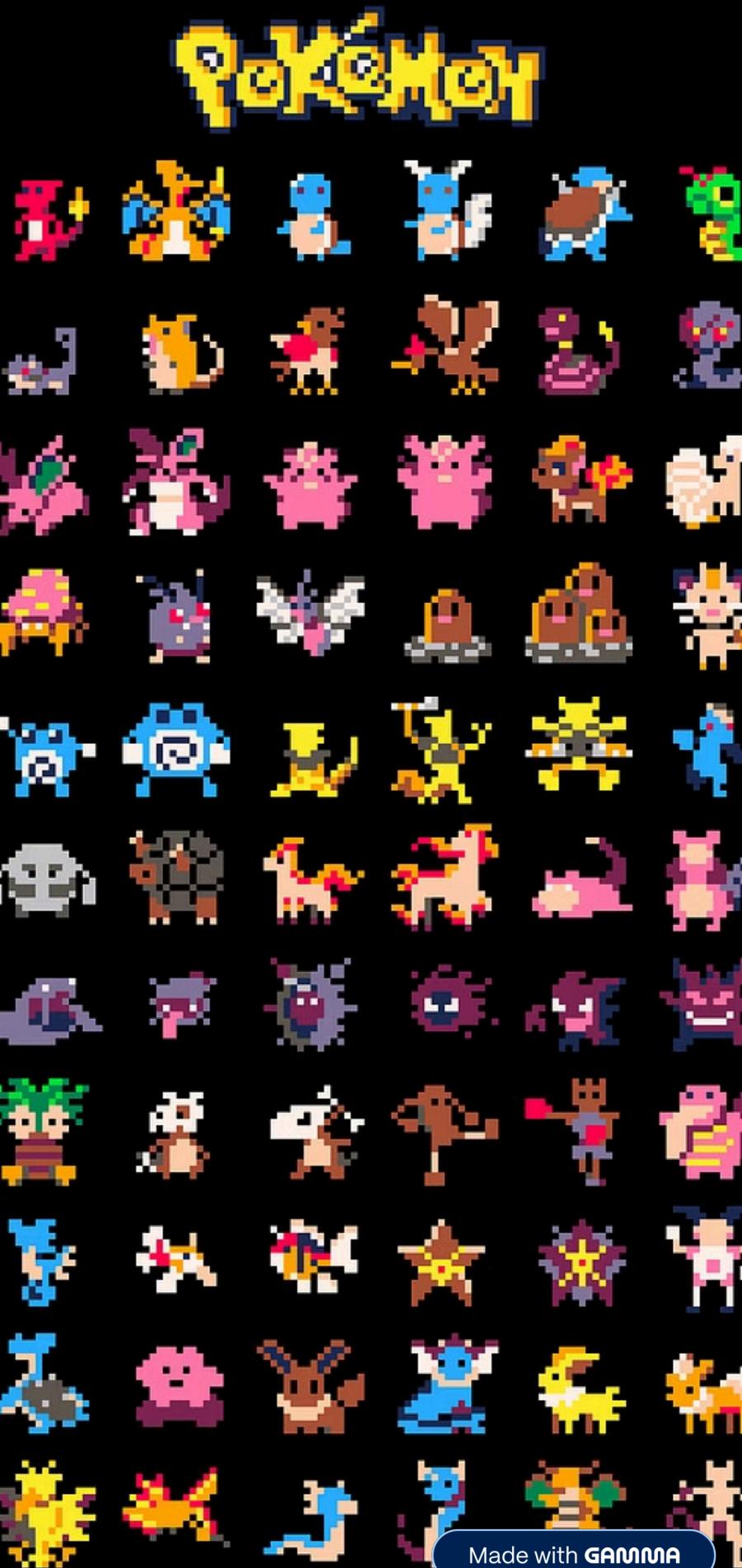
Permite avaliar velocidade de otimização e qualidade da solução

Tabela Final de Métricas

Sequência otimizada da rota encontrada

Recompensa Total coletada e Custo Ótimo de viagem

Valor de Fitness Máxima alcançada pela evolução



Conclusão: Eficácia da Abordagem Híbrida

O projeto demonstra com sucesso a eficácia de combinar algoritmos clássicos de busca informada (A*) com métodos de computação evolucionária (AG) para resolver problemas complexos de otimização.

Principais conquistas:

- Modelagem realista de restrições ambientais através de custos de terreno
- Sistema de recompensas baseado em probabilidades reais do jogo
- Trade-off inteligente entre custo de viagem e valor de recompensa
- Convergência consistente para soluções de alta qualidade

A arquitetura híbrida provou ser robusta e escalável para problemas de roteirização com múltiplos objetivos conflitantes.

