UNIVERSIDADE FEDERAL DE SÃO JOÃO DELREI

TRABALHO FINAL - GRAFOS

Algoritmo genético para solução do Problema do Caixeiro Viajante

Aluno: Rafael Carvalho Avidago Geraldo (212050057)
Professor: Vinicius Da Fonseca Vieira

Introdução

Este código representa a implementação de um algoritmo genético para otimização em grafos, especificamente projetado para resolver o problema do Caixeiro Viajante. Algoritmos genéticos são técnicas de busca e otimização inspiradas nos princípios da genética e seleção natural, e são particularmente eficazes em encontrar soluções ótimas ou aproximadas para problemas complexos

Implementação

criar_grafo: cria e retorna um grafo completo usando a biblioteca NetworkX. Um grafo completo é aquele em que cada nó está conectado a todos os outros nós. A função atribui um peso aleatório, entre 1 e 10, a cada aresta do grafo. O parâmetro num_nos especifica o número total de nós no grafo.

calcular_custo: essa função calcula o custo total de um caminho específico em um grafo. Ela itera sobre os nós no caminho fornecido, somando os pesos das arestas entre cada nó consecutivo. O custo é calculado de forma cíclica, ou seja, adiciona-se também o peso da aresta do último nó de volta ao primeiro. Essa função é crucial, pois atua como a função de avaliação (fitness) no contexto do algoritmo genético, determinando o quão bom é um determinado caminho.

cruzamento_ox: é um operador de cruzamento específico para problemas de permutação. Esta função recebe dois caminhos (pais) e gera dois novos caminhos (filhos) combinando partes dos pais. Ela garante que cada cidade apareça apenas uma vez no caminho do filho, mantendo a validade da solução.

inicializar_populacao: Ela cria num_individuos caminhos, onde cada caminho é uma permutação aleatória dos nós (cidades). Essas permutações são os indivíduos iniciais da população, cada um representando uma possível solução para o problema.

selecao_torneio: A seleção por torneio funciona selecionando vários pequenos grupos(torneios) da população, e em cada grupo o indivíduo com melhor desempenho e escolhido para a próxima geração.

mutacao: A mutação ocorre pela troca de posição de dois nós aleatórios no caminho. A função traz uma variação na população, o que e de suma importância para evitar a convergência prematura.

calcular_diversidade: Essa função mede a diversidade da população, calculando a diferença entre os caminhos. Uma maior diversidade normalmente apresenta uma melhor exploração de busca, enquanto uma diversidade menor pode sinalizar uma convergência prematura.

Algoritmo_genetico: Esta e a função principal que implementa o algoritmo genético, ela utiliza todas a funções acima para evoluir a população de soluções ao longo de várias gerações. A função busca encontrar o caminho com o menor custo total.

Main: e responsável por solicitar ao usuário os parâmetros necessários, como o número de cidades, tamanho da população e as taxas de cruzamento e mutação em seguida ela chama a função algoritmo genético para exibir o melhor resultado.

Funcionamento geral do código:

Quando o código é executado, ele inicia com a função main(), que primeiro solicita ao usuário que informe o número de cidades. Este número é usado para criar um grafo completo onde cada cidade é um nó e cada par de cidades é conectado por uma aresta com um peso aleatório. Essa estrutura de grafo é essencial para simular o Problema do Caixeiro Viajante, onde o objetivo é encontrar o caminho mais eficiente visitando todas as cidades.

Após a criação do grafo, o usuário é solicitado a fornecer mais informações, incluindo o tamanho da população para o algoritmo genético, o número de gerações, e as taxas de cruzamento e mutação. O tamanho da população define quantos caminhos diferentes ou soluções serão considerados simultaneamente. O número de gerações determina quantas vezes o processo de seleção, cruzamento e mutação será repetido, permitindo que as soluções evoluam ao longo do tempo.

A taxa de cruzamento controla a frequência com que dois caminhos na população são combinados para criar caminhos, enquanto a taxa de mutação define a probabilidade de um caminho sofrer alterações aleatórias. Esses parâmetros são fundamentais para balancear a exploração de novas soluções e a explotação das soluções existentes, visando encontrar o caminho mais eficiente.

À medida que o algoritmo executa, informações sobre cada geração são impressas no terminal, incluindo o número da geração atual, as rotas (caminhos) geradas nessa geração, e o custo do melhor caminho encontrado até então. O algoritmo também verifica a diversidade das soluções para evitar a convergência prematura para soluções subótimas.

Ao final das gerações, ou se o algoritmo detectar estagnação, o melhor caminho encontrado e seu custo total são exibidos, oferecendo ao usuário a solução mais eficiente para o problema dado. Este resultado representa o caminho ótimo ou próximo do ótimo para percorrer todas as cidades no grafo, minimizando o custo total do percurso.

Resultados e discussões

Durante sua execução, o algoritmo genético percorre várias gerações para evoluir uma população de caminhos, visando encontrar a rota mais eficiente. Em cada geração, o algoritmo exibe o custo do melhor caminho encontrado até o momento, permitindo acompanhar a melhoria progressiva das soluções. Para evitar a convergência prematura em soluções subótimas, a diversidade da população é monitorada, e são feitos ajustes, como introduzir novos indivíduos ou aumentar a taxa de mutação, para explorar melhor o espaço de busca. Ao final, o algoritmo apresenta o caminho mais eficiente encontrado e seu custo, oferecendo uma solução aproximada ao problema proposto.

A eficácia do algoritmo genético depende fortemente da configuração dos parâmetros, como o tamanho da população e as taxas de cruzamento e mutação, que influenciam diretamente sua capacidade de encontrar a solução ótima. Ajustes nesses parâmetros podem ter um impacto significativo no desempenho do algoritmo. De maneira fundamental, o algoritmo deve equilibrar a exploração de novas soluções (mantendo a diversidade) com a exploração das melhores soluções encontradas (focando na convergência), para evitar ficar preso em mínimos locais e garantir uma busca mais eficiente pelo espaço de soluções.

Embora os algoritmos genéticos sejam métodos robustos e flexíveis para resolver problemas de otimização, eles têm limitações, especialmente em relação à complexidade do problema e ao tempo de computação necessário. Em casos de grafos muito grandes, o algoritmo pode se tornar impraticável, seja pela demanda de recursos computacionais ou pelo tempo necessário para convergir para uma solução aceitável. Portanto, a aplicabilidade do algoritmo genético deve ser avaliada com base nas características específicas do problema em questão.

Conclusão

Em resumo, os resultados obtidos com este código ilustram a capacidade dos algoritmos genéticos em abordar problemas complexos de otimização. No entanto, a eficácia da solução depende de vários fatores, incluindo a configuração dos parâmetros do algoritmo e a natureza do problema em questão. A experimentação e o ajuste dos parâmetros são essenciais para otimizar o desempenho do algoritmo para uma aplicação específica.