Projeto 2 IC

Aluno: Silvano Martins da Silva Junior - 12011BCC042

Artigo Escolhido

An Efficient Genetic Algorithm for Solving the Generalized Traveling Salesman Problem

Código

Repositório

Principais Elementos

Descrição do Problema

O artigo trata do Generalized Traveling Salesman Problem (GTSP), que é uma variação do problema clássico do Caixeiro Viajante (TSP). No GTSP, o objetivo é encontrar um ciclo de custo mínimo em um grafo onde os nós estão agrupados em cluster. O ciclo deve incluir exatamente um nó de cada cluster. Os problemas Generalized são tipicamente pertencem a classe NP-completo e são mais difíceis que o problema clássico, atualmente estão sendo bastante estudados por conta das possíveis aplicações no mundo real. Resumindo tudo isto, o GTSP é basicamente o problema de encontrar o custo minimo em um circuito Hamiltoniano, porém, tratamos os "nodes" como "node sets" (cluster).

Representação do Problema

O grafo $G = (V, E)$ possui V (conjunto de nós) e E (conjunto de arestas com custos), o conjunto V é particionado em m subconjuntos, chamados de clusters.

Representação do Indivíduo (Cromossoma)

Cada indivíduo é representado por uma tupla (C, N).
C é uma lista que representa a sequência de clusters visitados, descrita como um ciclo Hamiltoniano sobre os clusters.
N é um conjunto de nós selecionados de cada cluster, ou seja $N = (N_1, N_2, \ldots, N_m)$ onde N_i é o nó selecionado do cluster C_i.

Operadores Genéticos

1. Crossover
   - Mantém a ordem e a posição dos símbolos (clusters) de um dos pais, preservando a sequência relativa dos símbolos do outro pai.
2. Mutação
   - Intra-cluster mutation: Um cluster é escolhido aleatoriamente, e o nó selecionado é trocado por outro nó dentro do mesmo cluster.
   - Inter-cluster mutation: Dois clusters têm suas posições trocadas aleatoriamente na sequência C.

Função de Adaptação

A função de adaptação calcula o custo total do ciclo de Hamilton generalizado com base no custo das arestas entre os nós selecionados de cada cluster.

Parâmetros de Funcionamento do Algoritmo

Número de Iterações (Critério de Parada)

O número de gerações varia entre 400 e 1000, dependendo do experimento.

Tamanho da População e Offspring

- Tamanho da População (μ): Definido como **5 vezes o número de clusters**.
- Tamanho da População Intermediária (λ): Definido como **2 vezes o tamanho da população (λ = 2 * μ)**.

Taxa de Mutação

- A **probabilidade de mutação** foi definida em **5%**.

Benchmarks Utilizados

- O algoritmo foi testado em nove problemas de benchmark da **TSPLIB**, com instâncias que variam de **198 a 442 nós**, divididos em um número específico de clusters.

Forma de Apresentação dos Resultados e Medidas Estatísticas

- Os resultados são comparados com dois outros algoritmos meta-heurísticos: **Random-Key Genetic Algorithm (RK-GA)** e **Memetic Algorithm (MA)**.

- Os benchmarks incluem medidas de custo mínimo de diferentes instâncias e são apresentados em uma tabela que compara os resultados dos três algoritmos para cada instância testada

---

Experimentos

Dados pelo autor:

Problem	*m*	*n*	*MA*	*RK-GA*	*GA*
40d198	40	198	10557	10557	10557
40kroa200	40	200	13406	13406	13406
40krob200	40	200	13111	13111	13111
46gr229	46	229	71641	71641	71832
53gil262	53	262	1013	1013	1014
60pr299	60	299	22615	22615	22618
80rd400	80	400	6361	6361	6389
84fl417	84	417	9651	9651	9651
89pcb442	89	442	21657	21657	21665

Para nos ajudar nos experimentos, coloquei o seguinte loop no código, para conseguirmos pegar sempre o melhor valor em 50 testes.

Loop

```
# Matriz de custo aleatória para teste (deve ser substituída por instâncias TSPLIB para experimentos reais)
gtsp = GTSP(clusters, cost_matrix)
solutions = []
solutions_val = []
for i in range(50):
    print("Solução ", i + 1)
    best_solution = genetic_algorithm(gtsp)
    solutions.append(best_solution)
    best_solution_val = gtsp.evaluate(best_solution)
    solutions_val.append(best_solution_val)
min_value, min_index = min((value, index) for index, value in enumerate(solutions_val))
print("Melhor Solução: ", solutions[min_index])
print("Valor total: ", min_value)
```

No final, conseguimos o seguinte:

Problem	m	n	MA	RK-GA	GA	Meu
40d198	40	198	10557	10557	10557	10557
40kroa200	40	200	13406	13406	13406	13406
40krob200	40	200	13111	13111	13111	13111
46gr229	46	229	71641	71641	71832	71843
53gil262	53	262	1013	1013	1014	1022
60pr299	60	299	22615	22615	22618	22620
80rd400	80	400	6361	6361	6389	6396
84fl417	84	417	9651	9651	9651	9651
89pcb442	89	442	21657	21657	21665	21665

Devido à limitação do tempo de apresentação, optei por incluir os experimentos com um número maior de clusters no relatório detalhado.

Conclusão

Apesar de não termos conseguido melhorar os resultados apresentados pelo autor no artigo, os valores obtidos pelo nosso código se aproximam significativamente dos resultados originais. Essa proximidade indica que a implementação está correta e produz soluções de alta qualidade, alinhadas com as melhores práticas e métodos utilizados. Embora existam ainda margens para ajustes e refinamentos no algoritmo, o desempenho atual é bastante satisfatório e demonstra a eficácia da abordagem adotada. Assim, podemos concluir que a solução desenvolvida é bastante competitiva e adequada para resolver os problemas propostos, reforçando a validade do nosso trabalho.