Gustavo Molina Figueiredo - Tech Challenge Fase 2 - RM 359124

Problema: Criação um Roteiro de Viagem

O problema de otimização escolhido foi maximizar a satisfação de um turista ao visitar determinadas atrações turísticas da cidade de São Paulo, respeitando um limite de tempo. Cada atração turística possui tempo de duração da visita e nível de interesse distintos.

Este problema envolve a alocação de recursos para visitar as atrações mais interessantes possíveis para o turista dentro do tempo máximo disponível para visitas.

Definição do Problema

- **Objetivo:** Maximizar a satisfação do turista.
- **Restrições:** O tempo total das visitas não pode exceder o tempo disponível.
- **Critérios de Sucesso:** Maximizar a satisfação do turista, garantindo que o tempo total das visitas não exceda o limite.
- **Recursos:**
    - **Tempo:** Cada turista possui um tempo máximo disponível para visitar as atrações.
    - **Atrações Turísticas:** Cada atração tem um tempo de visita e um nível de interesse associado.

O problema descrito é semelhante ao clássico problema da mochila, uma vez que o turista precisa selecionar as atrações (itens) de modo a maximizar sua satisfação (valor), sem ultrapassar o tempo disponível (capacidade da mochila).

A seguir, definimos os dados das atrações disponíveis para visita, onde cada tupla é organizada na ordem: atração, tempo, interesse.

Além disso, é definido o tempo máximo disponível para visitas.

```
ATRACOES = [
    ("Avenida Paulista", 2, 8),
    ("Parque do Ibirapuera", 3, 9),
    ("Museu de Arte de São Paulo (MASP)", 2, 10),
    ("Mercadão de São Paulo", 1.5, 7),
    ("Pinacoteca do Estado", 2, 9),
    ("Teatro Municipal", 1.5, 8),
    ("Estádio do Pacaembu", 2, 6),
    ("Beco do Batman", 1, 7),
    ("Jardim Botânico", 2, 8),
```

```
    ("Museu do Futebol", 2, 9)
]
TEMPO_DISPONIVEL = 10
```

Através da função **criar_individuo**, cada indivíduo será gerado como um vetor em que o número de elementos será igual ao número de atrações disponíveis.

A codificação utilizada foi a codificação binária. Sendo assim, cada elemento do vetor representa um gene e cada gene será 0 ou 1, dependendo de uma escolha aleatória. Se o gene for igual a 0, significa que a atração localizada no mesmo índice do vetor ATRACOES não foi escolhida para visita; se for igual a 1, significa que a atração foi escolhida.

A função **criar_populacao** utiliza a função **criar_individuo** para criar um determinado número de indivíduos e inicializar a população de maneira aleatória.

A função **calcula_fitness_individuo** recebe um indivíduo da população e calcula o seu fitness. O fitness, é calculado com base no nível de interesse de cada atração. Portanto, o fitness de cada indivíduo será a soma dos níveis de interesse das atrações selecionadas para visita.

Os indivíduos cuja soma do tempo total de visita de cada atração escolhida ultrapassar o tempo disponível serão penalizados e retornarão fitness igual a 0, de modo a respeitar a restrição do tempo máximo disponível.

A função **selecao_torneio_maior_fitness** seleciona aleatoriamente um determinado número de indivíduos da população, ordena e retorna aquele que possuir maior fitness.

A função **cruzamento** recebe dois indivíduos pais e retorna dois indivíduos filhos. *A técnica de cruzamento utilizada foi a Two-Point Crossover*, em que dois pontos de cruzamento, são selecionados aleatoriamente dentro do comprimento dos pais, sendo possível dividir o indivíduo em três partes. A seguir, a parte do meio de um pai é substituída pela parte central do outro pai, combinando, assim, características de ambos os indivíduos.

A próxima função, **mutação**, inverte ou não cada gene do indivíduo de acordo com a probabilidade descrita pela taxa de mutação.

Para implementação do algoritmo genético, algumas variáveis serão inicializadas com valores fixos, sendo elas:

- **Número de gerações/populações;**
- **Quantidade de indivíduos por geração/população;**
- **Taxa de cruzamento:** define em que proporção haverá cruzamento para gerar dois novos indivíduos a partir de dois indivíduos pais ou serão mantidos os indivíduos pais;
- **Taxa de mutação:** define em que proporção um indivíduo poderá sofrer mutação;
- **Tamanho do torneio:** define qual número de indivíduos será escolhido aleatoriamente dentre a população para sofrer seleção pelo maior fitness;

- **Elitismo:** define quantos dos melhores indivíduos, sendo aqueles que possuem maior fitness, serão mantidos na nova geração/população. Se for igual a zero, o melhor indivíduo não será preservado.
- **Quantidade de gerações para conversão ao melhor fitness:** define quantos rodadas sem mudança do resultado de melhor fitness são necessárias para afirmar que o algoritmo já convergiu para a solução subótima.

Os valores destas variáveis foram escolhidos conforme desempenho nos testes realizados. Portanto, os valores apresentados acima são aqueles que resultaram numa melhor convergência do algoritmo à uma solução subótima aceitável.

Implementação do Algoritmo Genético

A seguir, o algoritmo genético, que utilizará todas as funções descritas acima é implementado.

1 - População Inicial: primeiramente, são inicializados de forma aleatória todos os indivíduos que formarão a primeira população.

2 - Formação de Novas Gerações: em seguida, ocorre a criação de cada geração.

2.1 – Elitismo: ocorre quando cada uma das gerações é inicializada com o indivíduo de melhor *fitness* da geração anterior.

2.2 – Cruzamento: Os demais indivíduos desta geração são criados a partir do cruzamento de diferentes pares de indivíduos pais da população anterior, sendo que alguns destes indivíduos podem ser mantidos na nova geração, dada a taxa de cruzamento.

2.3 - Seleção Natural: Os indivíduos pais escolhidos para realizar o cruzamento, são aqueles de melhor *fitness*, selecionados, utilizando a técnica de torneio, dentre grupos de três indivíduos da geração anterior, formados aleatoriamente.

2.4 – Mutação: Então, os dois indivíduos filhos gerados sofrem mutação, pela técnica de *bit-flip*. Ou seja, na proporção dada pela taxa de mutação, cada indivíduo sofrerá a inversão dos valores de um ou mais de seus bits, selecionados aleatoriamente. Após a mutação, os indivíduos filhos passam a compor a nova geração e novos filhos são gerados até que cada geração alcance seu tamanho máximo.

3 - Convergência para Solução Subótima

O processo é repetido até a formação da última geração, cujo indivíduo de melhor *fitness* será a solução subótima, que representa uma aceitável alocação de recursos para visitar as atrações mais interessantes possíveis para o turista dentro do tempo máximo disponível para visita.

Testes de Eficácia

A fim de comprovar a eficácia do algoritmo genético implementado acima, os resultados obtidos foram comparados com os resultados de dois métodos convencionais para resolver problemas de alocação de recursos, tais quais o clássico problema da mochila e este, apresentado aqui, da elaboração de um roteiro de viagem. Os dois métodos convencionais escolhidos foram: **Algoritmo Guloso (*Greedy*)** e **Programação Dinâmica.**

- **Algoritmo Guloso (Greedy)**

Seleciona as atrações com maior valor (nível de interesse) até preencher o limite de tempo. Apesar de não garantir a solução ótima, suas principais vantagens são a velocidade de execução e o baixo uso de recursos computacionais.

- **Programação Dinâmica**

Testa todas as combinações possíveis de atrações, respeitando a restrição de tempo. Este método garante a solução ótima, porém exige mais recursos computacionais e possui implementação mais complexa.

Análise dos Resultados

Os resultados obtidos para o problema de maximização da satisfação do turista em visitas a atrações turísticas revelaram um desempenho variável entre os algoritmos Genético, Greedy e Programação Dinâmica. Após sucessivas execuções, observa-se que o algoritmo genético pode, ocasionalmente, encontrar soluções melhores do que as obtidas por métodos convencionais.

Comparação de Desempenho

Qualidade da Solução

Neste aspecto, a Programação Dinâmica é a mais eficaz, uma vez que fornece a solução ótima para o problema. A natureza determinística deste algoritmo permite uma busca exaustiva, garantindo a melhor solução possível.

O Algoritmo Greedy mostrou-se uma escolha rápida e eficiente, mas com a limitação de não considerar todas as combinações possíveis de atrações, o que pode resultar em soluções subótimas.

O Algoritmo Genético, por sua vez, foi capaz de alcançar resultados iguais aos da programação dinâmica e soluções frequentemente melhores do que as oferecidas pelo Algoritmo Greedy.

Tempo de Execução

Em termos de tempo de execução, o Algoritmo Greedy foi o mais rápido, mostrando sua eficiência em encontrar soluções rapidamente, mas não necessariamente as melhores.

A Programação Dinâmica apresenta, neste caso, tempos de execução competitivos para encontrar a solução ótima, o que é esperado para problemas de tamanho e complexidade moderados, como o aqui apresentado.

O Algoritmo Genético frequentemente apresentou tempo de execução maior que os demais métodos devido à sua natureza de exploração e iterações múltiplas.

Vantagens de Utilizar o Algoritmo Genético

Variabilidade

O Algoritmo Genético é projetado para explorar o espaço de soluções de maneira aleatória, o que significa que pode, em algumas execuções, encontrar combinações de atrações que superem as soluções fornecidas por métodos convencionais como o Greedy. Essa variabilidade pode ser vantajosa em certos contextos.

Performance em Grande Escala

Para problemas de otimização de alta dimensão ou com um espaço de busca complexo, os algoritmos genéticos podem ser mais eficazes do que abordagens exatas, já que estas últimas podem ser inviáveis devido a limitações computacionais. Assim, o Algoritmo Genético se destaca como uma alternativa quando a garantia da solução ideal se torna impraticável.

Conclusão

O Algoritmo Genético tem potencial para oferecer soluções melhores que métodos mais determinísticos em problemas mais complexos. A escolha entre esses algoritmos deve considerar fatores como a natureza do problema, os recursos computacionais disponíveis, o tempo disponível para a execução e a necessidade de garantia de uma solução ótima versus uma solução suficientemente boa e de rápida execução.