**Problema da Mochila**

**1. Descrição do Problema**

O problema da mochila (Knapsack Problem) é um problema de otimização combinatória. O objetivo é selecionar um subconjunto de itens, cada um com um valor e um peso associados, de forma que a soma dos pesos dos itens selecionados não ultrapasse a capacidade máxima da mochila, enquanto o valor total dos itens seja maximizado.

**2. Detalhes da Implementação do Algoritmo**

**Construção do Algoritmo**

**generate\_random\_population:** A população é gerada aleatóriamente por várias soluções representadas como bits (0s e 1s). Cada bit indica se um item foi incluído (1) ou não (0) na solução.

**calculate\_fitness:** calcula o valor total dos itens incluídos, retorna total dos valores e peso ou 0 e peso para valores que excedem a capacidade máxima da mochila.

**Crossover:** O crossover é responsável por combinar duas soluções parentais, trocando partes de suas representações para gerar novos indivíduos.

**Mutação:** A mutação altera aleatoriamente a solução de um indivíduo com base na taxa de mutação.

**Seleção:** A seleção dos pais é baseada na aptidão das soluções. Indivíduos com fitness mais alto têm maior chance de serem escolhidos para reprodução.

**Elitismo:** O elitismo garante que a melhor solução de uma geração seja preservada para a próxima, evitando a perda de boas soluções.

**3. Análise dos Resultados**

Os resultados são obtidos após a execução de múltiplas gerações. A cada geração, os melhores indivíduos são selecionados e combinados para formar novas soluções. O gráfico gerado em tempo real durante a execução do algoritmo mostra a progressão do valor da melhor solução encontrada até o momento. A convergência é observada quando o valor fitness estabiliza, indicando que o algoritmo encontrou uma solução quase ótima ou ótima.

Fazendo testes com o algoritmo percebemos grande variação dos resultados quando não utilizamos **elitismo,** sendo o mesmo responsável por manter as melhores soluções.O que pode acontecer também é devido ao uso de uma população alta e baixa mutação o resultado fique **estagnado** devido a ter encontrado uma solução tão boa que qualquer outra solução é ignorada ficando sempre preso naquela solução sub-ótima.

**4. Análise de Complexidade entre Algoritmos Genético e Convencionais**

**Algoritmo Genético**: **O(P\* G \* C)**

* P é o tamanho da população.
* G é o número de gerações.
* C é o custo de avaliar a função fitness, aproximadamente O(n), sendo n o número de itens.

A complexidade de um algoritmo genético depende principalmente do tamanho da população e do número de gerações necessárias para convergir para uma solução aceitável.

**Brute Force: O(2^n)**

* n é o número de itens.

O algoritmo de força bruta verifica todas as possíveis combinações de inclusão/exclusão de cada item na mochila.A complexidade cresce exponencialmente com o número de itens.

**Programação Dinâmica: O(n \* W)**

* n é o numero de itens
* W é a capacidade da mochila

A programação dinâmica resolve o problema dividindo-o em subproblemas e armazenando as soluções intermediárias em uma tabela. Para cada item, o algoritmo considera todas as capacidades possíveis de 0 até W.

**5. Conclusão**

Embora o Algoritmo Genético seja o único entre os métodos estudados que não garante uma solução ótima em todas as situações, ele se destaca especialmente quando o espaço de busca é grande.Sendo capaz de identificar soluções de alta qualidade em um tempo razoável ao ajustar parâmetros, como tamanho da população, número de gerações, taxa de mutação e elitismo, podemos alcançar a solução ideal de maneira mais eficiente do que com algoritmos convencionais.