O PROBLEMA DA MOCHILA

Inteligência Artifical

Maurício Júnior Santos Freire – 14A

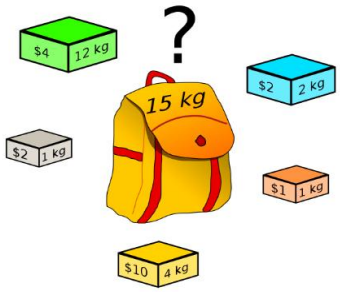
Sumário

Linguagem utilizada

Foi utilizada a linguagem JavaScript para o desenvolvimento do algoritmo.

Funcionamento

Definição dos itens disponíveis



Antes de aplicar o algoritmo, defini os itens disponíveis para colocar na mochila com os seguintes valores (como no documento do ClassRoom):

Item 1: Peso: 2, Valor: 2
Item 2: Peso: 12, valor: 4
Item 3: Peso: 1, valor: 2
Item 4: Peso: 1, valor: 1
Item 5: Peso: 4, valor: 1

Inicialização

Foi inicializado todos os dados para o funcionamento do algoritmo: a solução aleatória, os dados de temperatura e iteração, melhor valor e os dados que serão utilizados posteriormente para a formação do arquivo CSV que será utilizado para plotar os gráficos.

Função de Avaliação

A função de avaliação recebe como entrada a solução atual, que é uma lista de valores booleanos indicando quais itens estão selecionados para a mochila. Para cada item na lista, verifica-se se o valor correspondente é "true" (item selecionado) ou "false" (item não selecionado).
Em seguida, itera-se sobre todos os itens disponíveis e, para cada item selecionado, soma-se o seu valor total e o seu peso total. Se o peso total exceder a capacidade máxima da mochila, o valor total é penalizado (definido como 0), indicando que a solução é inválida.
Ao final do cálculo, a função de avaliação retorna o valor total da mochila, que representa a qualidade da solução em termos de valor máximo alcançado.

Laço principal do Simulated Annealing

O laço principal segue os seguintes passos:

1. Definir a temperatura inicial e a temperatura final para o algoritmo.
2. Definir a taxa de resfriamento, que controla como a temperatura diminui a cada iteração.
3. Dentro do laço principal, os seguintes passos são executados:
    - Gerar uma nova solução vizinha fazendo uma pequena alteração na solução atual. Isso pode envolver adicionar ou remover aleatoriamente um item da mochila.
    - Calcular o valor da nova solução vizinha usando a função de avaliação.
    - Calcular a diferença de valor entre a nova solução vizinha e a solução atual.
    - Verificar se a nova solução vizinha é melhor (ou seja, tem um valor maior) do que a solução atual. Se for, aceitar a nova solução vizinha como a nova solução atual.
    - Caso contrário, aceitar a nova solução vizinha com uma probabilidade dependente da diferença de valor e da temperatura atual. Quanto maior a temperatura, maior a probabilidade de aceitar soluções piores. Isso permite que o algoritmo escape de mínimos locais e explore diferentes áreas do espaço de solução.

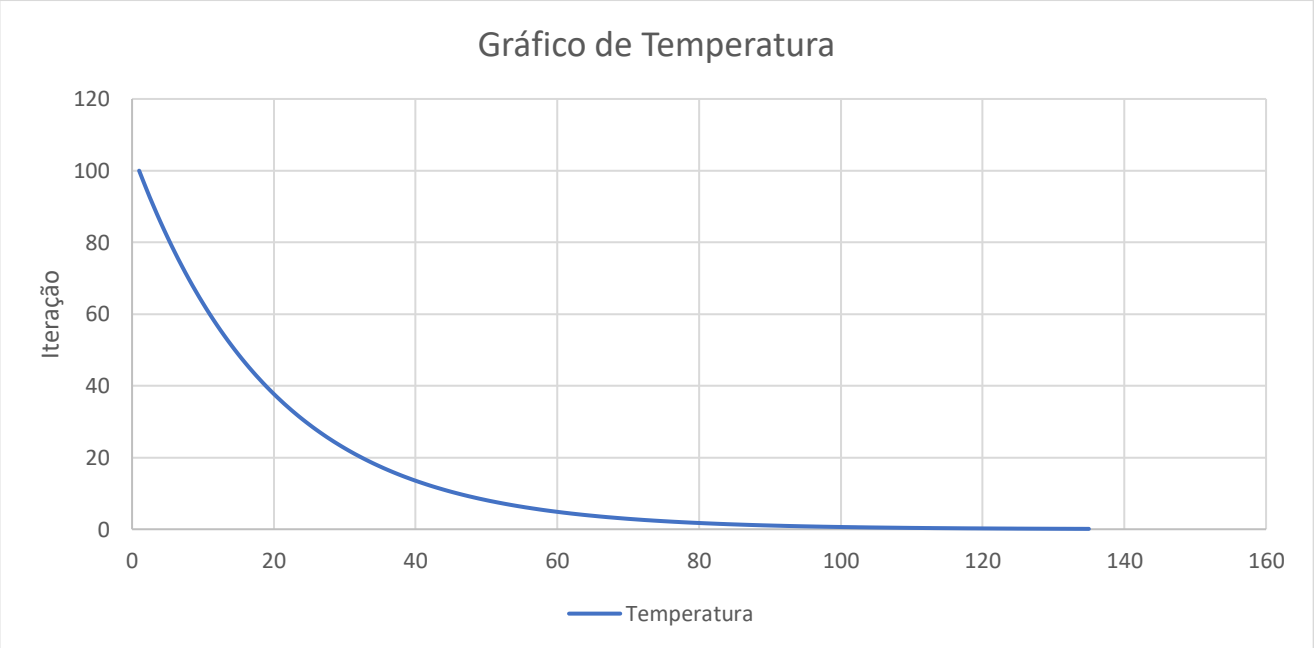
- Popular os dados para a construção do CSV.
- Repetir esses passos até que a temperatura atinja o valor final.
- A temperatura é reduzida a cada iteração, geralmente multiplicando-a pela taxa de resfriamento definida. Isso faz com que o algoritmo comece a aceitar menos soluções piores à medida que avança, aumentando a probabilidade de convergir para uma solução ótima.

Ao final do laço principal ele retorna a melhor solução, o melhor valor e os dados para a construção do CSV.

Rodando o algoritmo

1. Certifique-se de ter o Node.js instalado em sua máquina. Você pode baixá-lo em https://nodejs.org.
2. Clone o repositório ou faça o download dos arquivos.
3. Abra um terminal e navegue até o diretório onde os arquivos estão localizados.
4. Execute o código do arquivo index.js com o Node.js através do seguinte comando "node index.js"
5. Você verá no console a melhor solução encontrada para o problema da mochila, juntamente com o valor da solução e os arquivos CSV gerados para realizar a construção do gráfico.

Gráficos



No algoritmo do Simulated Annealing, é esperado que a função de avaliação seja maximizada, já que o objetivo é encontrar uma solução com o maior valor possível. Portanto, os valores do gráfico de erro devem aumentar ao longo das iterações, refletindo a melhoria gradual na qualidade da solução encontrada.

Possíveis outras estratégias para resolver o problema

1. Algoritmo Genético: Utiliza uma abordagem inspirada na evolução biológica, onde uma população de soluções é gerada e evolui ao longo das gerações através de operadores genéticos como seleção, recombinação e mutação.
2. Programação Dinâmica: Divide o problema em subproblemas menores e constrói a solução final a partir da combinação ótima das soluções dos subproblemas. É eficiente para problemas menores com uma capacidade relativamente pequena.
3. Heurísticas Construtivas: Constroem uma solução passo a passo, selecionando e adicionando itens à mochila com base em critérios específicos. Exemplos de heurísticas construtivas incluem a Greedy Heuristic, que seleciona itens de acordo com a relação valor-peso, e a Heurística do Maior Valor, que seleciona os itens de maior valor até que a capacidade seja atingida.
4. Algoritmo Branch and Bound: Explora o espaço de solução de forma exaustiva, mas com estratégias de poda para evitar a avaliação de soluções inviáveis ou soluções que claramente não levarão à solução ótima. É eficiente para problemas menores com um número limitado de itens.
5. Algoritmos Metaheurísticos: São algoritmos gerais de busca que podem ser aplicados a uma ampla variedade de problemas, incluindo o Problema da Mochila. Exemplos de metaheurísticas são o Simulated Annealing, que foi abordado anteriormente, e o Algoritmo de Busca Tabu, que mantém uma lista de movimentos proibidos para diversificar a busca.

Link do repositório

O repositório está localizado no GitHub: Repositório.