Resolução problema da mochila com algoritmo genético

Gabriel Meale

[1]Universidade Federal do ABC (UFABC)

Abstract. *This project proposes to solve the classic Knapsack Problem using Genetic Algorithms (GAs). The Knapsack Problem is a combinatorial optimization problem, where the goal is to select a set of items to put in a knapsack, maximizing the total value of the items, subject to the constraint of the knapsack's weight capacity. Genetic Algorithms are a class of optimization algorithms inspired by biological evolution, which operate with a population of candidate solutions, using genetic operators such as crossover and mutation to generate new solutions. In this project, we implemented a GA to solve the Knapsack Problem, utilizing binary chromosome representations, appropriate genetic operators, and a fitness evaluation function based on the total value of the items and the knapsack's weight capacity. Experimental results demonstrate the effectiveness of the GA in finding high-quality solutions for the Knapsack Problem, showcasing its potential as an optimization tool for real-world practical problems.*

Resumo. *Este projeto propõe a resolução do clássico Problema da Mochila usando Algoritmos Genéticos (AG). O Problema da Mochila é um problema de otimização combinatória, onde o objetivo é selecionar um conjunto de itens para colocar em uma mochila, maximizando o valor total dos itens, sujeito à restrição da capacidade de peso da mochila. Os Algoritmos Genéticos são uma classe de algoritmos de otimização inspirados na evolução biológica, que operam com uma população de soluções candidatas, usando operadores genéticos como crossover e mutação para gerar novas soluções. Neste projeto, implementamos um AG para resolver o Problema da Mochila, utilizando representações de cromossomos binários, operadores genéticos adequados e uma função de avaliação de aptidão baseada no valor total dos itens e na capacidade de peso da mochila. Os resultados experimentais demonstram a eficácia do AG em encontrar soluções de alta qualidade para o Problema da Mochila, demonstrando seu potencial como ferramenta de otimização para problemas práticos do mundo real.*

1. Introdução

O Problema da Mochila é um dos problemas mais estudados e fundamentais da teoria da otimização combinatória. Ele desafia os pesquisadores a encontrar uma seleção ótima de itens que maximizem o valor total, levando em consideração as restrições de capacidade de uma mochila. Essa combinação de eficiência e restrições é relevante em muitos contextos do mundo real, como logística, gerenciamento de recursos e planejamento de projetos.

Para abordar o Problema da Mochila, este projeto propõe a utilização de Algoritmos Genéticos (AG), uma técnica de otimização inspirada na teoria da evolução

biológica. Os AGs operam com uma população de soluções candidatas representadas por cromossomos, utilizando operadores genéticos como crossover e mutação para gerar novas soluções. Essa abordagem permite uma exploração eficiente do espaço de busca, buscando soluções de alta qualidade.

O objetivo deste projeto é implementar e aplicar um Algoritmo Genético para resolver o Problema da Mochila, considerando uma lista de itens com valores monetários e pesos associados. Será desenvolvida uma representação de cromossomo adequada, juntamente com operadores genéticos e uma função de avaliação de aptidão que leve em conta o valor total dos itens e a capacidade de peso da mochila.

Por meio deste projeto, buscamos demonstrar a eficácia dos Algoritmos Genéticos como uma ferramenta poderosa para resolver problemas de otimização combinatória, destacando sua aplicabilidade e relevância em cenários práticos do mundo real.

2. Metodologia

Nessa seção serão descritas as principais características do algoritmo e do experimento. Vale ressaltar que a implementação foi feita em Python, utilizando orientação à objetos. O código foi codificado para que seja aberto a mudanças, respeitando os principais princípios de SOLID.

2.1. Problema inicial

Os itens que serão carregados na mochila são listados pela tabela 1. Além disso, o peso máximo que a mochila suporta é de **15 kg**.

2.2. Hiperparâmetros

Para o processamento do algoritmo, podemos configurar os seguintes hiperparâmetros:

- **tamanho_populacao** - Controla o tamanho da população *(Default: 100)*.
- **max_geracoes** - Número de gerações que acontecerão no algoritmo *(Default: 100)*.
- **taxa_mutacao** - Controla a probabilidade de uma mutação ocorrer em algum gene do cromossom *(Default: 0.1)*.
- **mutacao_flag** - Sinaliza se ocorrerá mutação durante o processamento ou não *(Default: True)*.
- **crossover_flag** - Sinaliza se ocorrerá crossover durante o processamento ou não *(Default: True)*.
- **tipo_selecao** - Codifica o tipo de seleção que acontecerá aos indíviduos da população *(Default: TORNEIO)*.
- **torneio_size** - Configura o tamanho da seleção aleatória da população para encontrar o indíviduo mais apto naquela iteração do torneio*(Default: 3)*.

2.3. Cromossomo

O cromossomo será representado como um vetor de 20 posições, respectivas ao indíce de cada item a ser adicionado na mochila. Cada posição será codificada por um bit (1 ou 0). Caso o indíviduo da população considere que aquele item deva entrar na mochila, será codificado como 1, caso contrário, 0. Um exemplo da representação se encontra na tabela 2.

Table 1. Itens e suas descrições

Item	Valor (R$)	Peso (kg)
Barraca	150,00	3,5
Saco de dormir	100,00	2,0
Isolante térmico	50,00	0,5
Colchão inflável	80,00	1,0
Lanterna	30,00	0,2
Kit de primeiros socorros	20,00	0,5
Repelente de insetos	15,00	0,1
Protetor solar	20,00	0,2
Canivete	10,00	0,1
Mapa e bússola 25,00	0,3	
Garrafa de água 15,00	1,8	
Filtro de água	50,00	0,5
Comida (ração liofilizada)	50,00	3,0
Fogão de camping	70,00	1,5
Botijão de gás	30,00	1,2
Prato, talheres e caneca	20,00	0,5
Roupas (conjunto)	80,00	1,5
Calçados (botas)	120,00	2,0
Toalha	20,00	0,5
Kit de higiene pessoal	30,00	0,5

Table 2. Representação do cromossomo

Cromossomo	
Índice do Gene	Valor
1	1
2	0
3	1
4	0
5	1
6	0
7	1
8	1
9	0
10	1

2.4. Seleção

2.4.1. Torneio

Para realizar a seleção, foi utilizado o algoritmo de torneio. A primeira etapa consiste em escolher aleatoriamente na população *torneio_size* indíviduos aleatórios da população. Logo em seguida, basta escolher qual deles retorna o maior *fitness*. Em específico nesse caso, a iteração acontece duas vezes (já que são dois pais). Nessa abordagem, foi realizado a reprodução biparental, ou seja, a próxima geração inteira será gerada por esses dois pais selecionados.

2.5. Fitness

A função de Fitness implementada é extremamente simples. Em geral, quando um gene do cromossomo sinaliza que um item deve ser adicionado a mochila, o preço desse item e seu respectivo peso são armazenados em duas variáveis. Após o processamento de cada gene, é avaliado o valor total dos itens. Caso o peso não tenha ultrapassado a capacidade da mochila, esse valor total é retornado. Caso contrário, a função retorna 0 (como forma de penalização para soluções inviáveis ao problema), como ilustrado no algoritmo 1.

Algorithm 1 Calcular Fitness

Require: Um indivíduo $individuo$
Ensure: O valor de aptidão do indivíduo

```
valor_total ← 0
peso_total ← 0
for cada item em ITENS_MOCHILA do
    if individuo.cromossomo[0] == 1 then
        valor_total ← valor_total + item["valor"]
        peso_total ← peso_total + item["peso"]
    end if
end for
if peso_total > CAPACIDADE_MOCHILA then
    return 0
end if
return valor_total
```

2.6. Mutação

A mutação do algoritmo consistem em, baseado na *taxa_mutacao*, mudar aleatoriamente alguns bits do vetor cromossomo.

2.7. Crossover

O crossover do algoritmo consiste em, primeiramente, escolher aleatoriamente um ponto de corte no cromossomo. Após isso, os cromossomos dos pais 1 e 2 são cortados nesse ponto gerando sub-vetores A1, B1, A2, B2. Por fim, B1 é combinado com A2 e, B2 é combinado com A1.

2.8. Geração populacional

Depois da operação de seleção (que deve escolher dois indivíduos candidatos após duas iterações do torneio), a nova população é gerada a partir apenas desses dois pais. Ou seja, ocorrerão $tamanho\_populacao/2$ iterações, realizando operações de crossover e mutação nesses dois pais, o que gerará uma nova população de 100 indivíduos.

Todo esse processo é repetido de acordo com a quantidade de *max_geracoes* configurado no algoritmo.

3. Resultados e Observações

3.1. Processamento com parâmetros *default*

Podemos observar, baseado nas tabelas 3 e 4 que a melhor solução da última geração (tabela 4) trouxe um valor de otimização satisfatório, com um peso de 13.7kg. Entretanto, podemos observar pela tabela 3 que existiu uma geração que otimizou ainda mais o valor (geração 30).

Dessa forma, foi feito um teste com o primeiro ajuste de hiperparâmetros (*max_geracoes*=30). Com isso, tivemos os resultados presentes nas tabelas 5 e 6. Agora, é possível observar resultados ainda melhores e, de certa forma, bem perto do limite de otimização (15kg).

Table 3. Melhores valores ao longo das gerações

Geração	Melhor valor
0	R$670
10	R$800
20	R$795
30	R$815
40	R$735
50	R$800
60	R$745
70	R$795
80	R$765
90	R$725

Table 4. Melhor solução encontrada

Melhor solução	Valor	Peso
$[1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0]$	R$800	13.7kg

3.2. Testes de *tuning*

Durante os testes de *tuning* de hiperparâmetros, foi constado que o parâmetro de tamanho do torneio era extremamente sensível para otimizar a solução. O que faz sentido, já que ele passa a considerar em um momento a população inteira (quando o tamanho do torneio é igual ao tamanho da população), o que pode restringir a variedade genética. Por isso,

Table 5. Melhores valores ao longo das 30 primeiras gerações

Geração	Melhor valor
0	R$725
10	R$830
20	R$775

Table 6. Melhor solução encontrada após 30 gerações

Melhor solução	Valor	Peso
$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1]$	R$870	14.9kg

pode existir o risco de estagnarmos em um máximo local e não conseguir achar soluções melhores. Um tamanho menor do torneio favorece a exploração, permitindo que soluções potencialmente melhores sejam descobertas, enquanto um tamanho maior do torneio favorece a explotação, concentrando-se em torno de soluções já conhecidas. Esse parâmetro deve ser ajustado com extremo cuidado.

Além disso, durante os testes, foi possível constatar que, nesse exemplo, aumentar significativamente a taxa de mutação tornava mais difícil a tarefa de encontrar uma solução ótima. Isso provavelmente é ocasionado porque aumentar a taxa de mutação faz a probabilidade de boas soluções serem descartadas, e também é mais difícil convergir para uma solução ótima, já que a busca é menos direcionada.

Por fim, vale dizer que aumentar a população ou aumentar as gerações, tende a resultar em achar alguma solução boa. Isso porque o algoritmo acabará iterando várias vezes e, apesar de a população inicial poder ser pequena, com a taxa de gerações alta, a variabilidade genética que tende a aumentar, e vice-versa. Os únicos cenários ruins do ponto de vista de solução, são poucas amostragens iniciais e/ou poucas gerações populacionais.

3.3. Teste com hiperparâmetros otimizados

Valores atribuidos aos hiperparâmetros:

- **tamanho_populacao** - 300.
- **max_geracoes** - 300.
- **taxa_mutacao** - 0.25.
- **torneio_size** - 3.

Os resultados do processamento podem ser visualizados nas tabelas 7 e 8. Em geral, ter uma população grande favorece muito desempenho do algoritmo, mas tudo indica que não é possível ter resultados satisfatórios se a taxa geracional for baixa. Foi possível obter novamente o valor de **R$870**. Em nenhum dos testes executados esse valor aumentou, o que nos leva a concluir que, à princípio, esse é o valor ótimo do problema, quando abordado dessa forma.

4. Trabalhos Futuros

- Exploração da aplicação com outros algoritmos genéticos mais complexos
- Implementação com um método de seleção diferente.

Table 7. Melhores valores ao longo das gerações (pós ajuste)

Geração	Melhor valor
0	R$740
10	R$780
20	R$740
30	R$775
40	R$790
50	R$765
60	R$790
70	R$765
80	R$740
90	R$755
100	R$805
110	R$825
120	R$805
130	R$830
140	R$775
150	R$815
160	R$765
170	R$770
180	R$735
190	R$755
200	R$805
210	R$750
220	R$810
230	R$800
240	R$780
260	R$805
270	R$815
280	R$785
290	R$805

Table 8. Melhor solução encontrada (pós ajuste)

Melhor solução	Valor	Peso
$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1]$	R$870	14.7kg

References

Puertas de Araújo, Hugo. (Q1.2024). Aulas de Computação Evolutiva e Conexionista. Curso de Bacharelado em Ciência da Computação, Universidade Federal do ABC.