



UDESC

IBIRAMA

**UNIVERSIDADE ESTADUAL DE SANTA CATARINA
CEAVI - CENTRO DE EDUCAÇÃO SUPERIOR DO ALTO VALE DO ITAJAÍ
DEPARTAMENTO DE ENGENHARIA DE SOFTWARE**

Augusto Rustick

Métodos Quantitativos - Trabalho III

Ibirama

1. PROBLEMA

O problema do Knapsack, ou problema da mochila, é bastante conhecido e estudado em otimização combinatória, tanto por sua simplicidade quanto por sua utilidade em várias áreas, como planejamento de recursos e logística. De maneira geral, esse problema se resume a escolher itens que maximizem um benefício dentro de uma capacidade limitada, o que o torna um exercício valioso para entender decisões e restrições de recursos.

A ideia central do problema pode ser explicada pela metáfora de uma mochila. Imagine que temos uma mochila com um limite de peso que ela pode carregar e um conjunto de itens disponíveis para colocar dentro dela. Cada item tem um peso e um valor associados: o peso indica o espaço que ele ocupa na mochila, enquanto o valor representa o benefício ou importância de levar aquele item. No entanto, a capacidade da mochila impõe que nem todos os itens poderão ser levados. O desafio é, então, escolher uma combinação de itens que maximize o valor total sem exceder o limite de peso.

Existem várias versões do problema do Knapsack. Na forma mais básica, chamada knapsack 0-1, cada item só pode ser incluído na mochila ou deixado de fora; ou seja, não é possível levar apenas uma parte de um item. Esse cenário traz uma decisão binária para cada item, o que torna o problema interessante e desafiante. Já na versão conhecida como knapsack fracionário, é permitido levar apenas uma fração dos itens, o que muda um pouco o foco da maximização de valor, pois itens podem ser parcialmente incluídos para otimizar o espaço.

O problema do Knapsack é classificado como NP-difícil, o que significa que ele não tem uma solução que possa ser encontrada rapidamente para todos os casos. À medida que o número de itens aumenta, o problema se torna ainda mais complexo, demandando muito tempo de processamento para ser resolvido com precisão. Por essa razão, em situações práticas, é comum o uso de métodos aproximados, como algoritmos de heurística e de aproximação, que conseguem encontrar soluções próximas do ideal em um tempo aceitável.

2. INSTÂNCIAS DO PROBLEMA

As instâncias do problema de Knapsack 0/1 utilizadas neste estudo estão organizadas em arquivos de texto simples e seguem uma convenção de nomenclatura padronizada que facilita a identificação de suas características principais. Cada arquivo tem um nome no formato kp_n_wmax , onde:

- **kp:** indica que o arquivo representa uma instância do problema de Knapsack 0/1.
- **n:** refere-se ao número total de itens disponíveis para inclusão na mochila, um dado essencial para a construção da matriz de itens e valores.
- **wmax:** representa a capacidade máxima da mochila, ou seja, o limite de peso que não deve ser ultrapassado pela soma dos pesos dos itens selecionados.

Cada arquivo segue a seguinte estrutura: A linha 1 contém dois valores inteiros, n e $wmax$, que indicam, respectivamente, o número de itens disponíveis e a capacidade máxima da mochila, enquanto nas linhas subsequentes, cada linha subsequente apresenta um par de valores inteiros, v_i e w_i

O foco deste trabalho será trabalhar nas instâncias conhecidas como “low-dimensional instances”, que podem ser encontradas na seguinte URL: http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/

3. FORMULAÇÃO MATEMÁTICA

O problema de Knapsack 0/1 é formulado matematicamente para maximizar o valor total de itens selecionados, respeitando uma restrição de capacidade de peso. No contexto das instâncias do problema 0/1 Knapsack, que seguem a estrutura detalhada anteriormente, temos uma configuração onde cada item é representado por dois parâmetros: valor e peso. A formulação do problema busca encontrar a combinação ótima desses itens que maximiza o valor total sem ultrapassar a capacidade da mochila.

3.1. PARÂMETROS

Para resolver o problema, definimos um conjunto de variáveis e uma função objetivo, além de uma restrição de capacidade. Abaixo está a formulação detalhada do problema de Knapsack 0/1, com variáveis e expressões matemáticas que traduzem as informações das instâncias.

n : Número total de itens, como especificado na primeira linha do arquivo.

W : Capacidade máxima da mochila (também fornecida na primeira linha).

V_i : Valor associado ao item i , onde $i = 1, 2, \dots, n$. Este valor é lido da segunda coluna em cada linha subsequente do arquivo.

W_i : Peso associado ao item i , onde $i = 1, 2, \dots, n$. Este valor é lido na terceira coluna de cada linha subsequente do arquivo.

3.2. VARIÁVEL DE DECISÃO

X_i : Variável binária associada ao item i , sendo que a variável assume 1 ou 0, para informar se o item será incluído na mochila

3.3. FUNÇÃO OBJETIVO

O objetivo do problema é maximizar o valor total dos itens que são incluídos na mochila. Isso pode ser expresso pela função objetivo:

$$\text{MAXIMIZAR } Z = \sum_{i=1}^n V_i * X_i$$

Onde $V_i * X_i$ representa o valor do item i se ele for selecionado para a mochila, o que faz com que a função objetivo, calcule a soma dos valores de todos os itens selecionados, com o objetivo de maximizar esse total.

3.4. RESTRIÇÃO

Para respeitar a limitação de capacidade da mochila, impomos uma restrição sobre o peso total dos itens selecionados. Essa restrição assegura que a soma dos pesos dos itens escolhidos não exceda a capacidade máxima W . Temos então a seguinte formulação

$$\sum_{i=1}^n W_i * X_i \leq W$$

Onde: $W_i * X_i$ é o peso do item i se ele for incluído na mochila (ou zero, caso contrário). Essa restrição limita o somatório total dos pesos dos itens selecionados para que não ultrapasse a capacidade W da mochila.

4. RESULTADOS

Implementei o modelo para resolver o problema de Knapsack 0/1 e consegui alcançar as soluções ótimas para as instâncias analisadas. Abaixo, apresento um comparativo detalhado, que mostra os valores totais obtidos e o desempenho do algoritmo em termos de precisão e tempo de execução em relação às soluções ótimas esperadas.

Esse comparativo ajuda a validar a implementação e a eficiência do método aplicado, permitindo observar como a solução se comporta em cada instância, especialmente em termos de convergência para o valor ótimo e consumo de recursos computacionais.

NOME DA INSTÂNCIA	SOLUÇÃO PRÓPRIA	SOLUÇÃO ÓTIMA	TEMPO DE EXECUÇÃO	ITENS ESCOLHIDOS
f1_l-d_kp_10_269	295	295	00:00:00.042	[2, 3, 4, 8, 9, 10]
f2_l-d_kp_20_878	1024	1024	00:00:00.041	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17, 19, 20]
f3_l-d_kp_4_20	35	35	00:00:00.041	[1, 2, 4]
f4_l-d_kp_4_11	23	23	00:00:00.042	[2, 4]
f5_l-d_kp_15_375	481.06937	481.0694	00:00:00.042	[3, 5, 7, 8, 10, 11, 12, 14, 15]
f6_l-d_kp_10_60	52	52	00:00:00.042	[3, 5, 6, 7, 8, 9, 10]
f7_l-d_kp_7_50	107	107	00:00:00.042	[1, 4]
f8_l-d_kp_23_10000	9767	9767	04:46:44.120	[1, 2, 3, 4, 5, 6, 7, 8, 10, 16, 17]
f9_l-d_kp_5_8	130	130	00:00:00.042	[1, 2, 3, 4]
f10_l-d_kp_20_879	1025	1025	00:00:00.111	[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 16, 18, 19, 20]

As instâncias 5 e 8 são as únicas que julgo como necessidade de algum comentário em especial.

Primeiramente, para a f5, eu resolvi optar por formatar com uma casa decimal a mais do que a disponível no site das instâncias. A julgar pelo fato que todas as outras tiveram as suas soluções ótimas encontradas, acredito que apenas foi feito um

arredondamento com 4 casas decimais, enquanto eu usei cinco, mas não consegui encontrar qual seria o valor exato para saber se até nas casas decimais o meu resultado bateu com as soluções ótimas encontradas. Não descarto a possibilidade de que a minha solução possa estar diferente no que se diz respeito aos decimais.

Quanto à instância f8, o tempo de execução de quase 5 horas me deixou surpreso. Isso se dá devido ao fato que a solução ótima foi encontrada em aproximadamente 20 minutos, e todo restante do tempo foi apenas do GLPK buscando por alguma solução melhor. Perto do tempo de execução comentado, apareceu a seguinte informação no terminal:

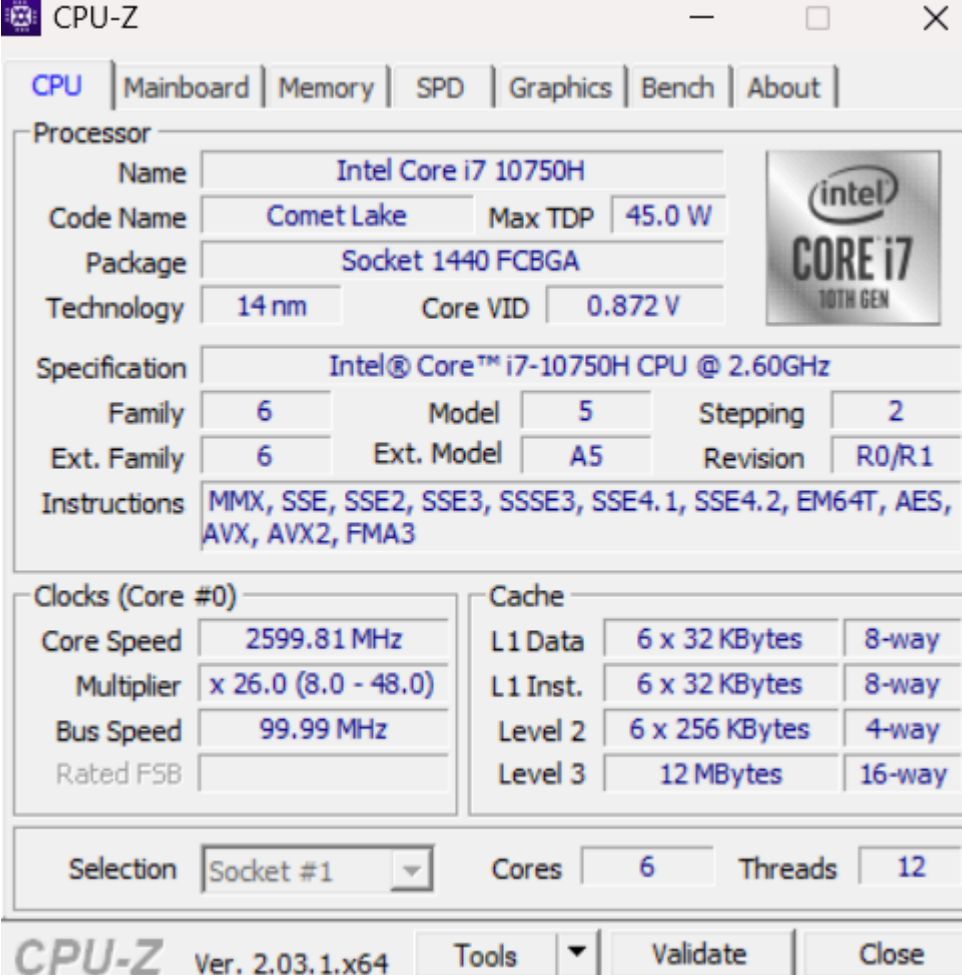
```
GLPK Simplex Optimizer, v4.65
1 row, 23 columns, 23 non-zeros
* 0: obj = -0.00000000e+00 inf = 0.000e+00 (23)
* 13: obj = 1.000049180e+04 inf = 0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
Integer optimization begins...
Long-step dual simplex will be used
+ 13: mip = not found yet <= +inf (1; 0)
Solution found by heuristic: 9744
Solution found by heuristic: 9745
Solution found by heuristic: 9767
+ 27643: mip = 9.767000000e+03 <= 1.000000000e+04 2.4% (21736; 5804)
+ 43798: mip = 9.767000000e+03 <= 1.000000000e+04 2.4% (34087; 9558)
+ 54208: mip = 9.767000000e+03 <= 1.000000000e+04 2.4% (42023; 11987)
```

Então o valor da solução ótima, de 9767, já havia aparecido muito cedo, apesar de que o programa continuou a tentar buscar por soluções melhores.

Com exceção dessas duas instâncias em particular, as demais não existe muito o que ser comentado. Todas elas conseguiram encontrar a solução ótima já encontrada, bem como com tempos de execução ínfimos, então funcionou bem a minha implementação no GLPK.

5. INFORMAÇÕES ADICIONAIS

Os testes foram feitos em um computador com as seguintes especificações:



The screenshot displays the CPU-Z application window, which provides detailed information about the system's processor. The 'CPU' tab is selected, showing the following data:

Processor			
Name	Intel Core i7 10750H		
Code Name	Comet Lake	Max TDP	45.0 W
Package	Socket 1440 FCBGA		
Technology	14 nm	Core VID	0.872 V
Specification	Intel® Core™ i7-10750H CPU @ 2.60GHz		
Family	6	Model	5
Ext. Family	6	Ext. Model	A5
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, FMA3		

Clocks (Core #0)		Cache	
Core Speed	2599.81 MHz	L1 Data	6 x 32 KBytes
Multiplier	x 26.0 (8.0 - 48.0)	L1 Inst.	6 x 32 KBytes
Bus Speed	99.99 MHz	Level 2	6 x 256 KBytes
Rated FSB		Level 3	12 MBytes

Selection	Cores	Threads
Socket #1	6	12

The bottom of the window shows the CPU-Z logo, version 2.03.1.x64, and buttons for 'Tools', 'Validate', and 'Close'.

CPU-Z

CPU

Mainboard

Memory

SPD

Graphics

Bench

About

General

Type

DDR4

Channel #

Single

Size

8 GBytes

DC Mode

Uncore Frequency

800.0 MHz

Timings

DRAM Frequency

FSB:DRAM

3:44

CAS# Latency (CL)

22.0 clocks

RAS# to CAS# Delay (tRCD)

21 clocks

RAS# Precharge (tRP)

21 clocks

Cycle Time (tRAS)

47 clocks

Row Refresh Cycle Time (tRFC)

514 clocks

Command Rate (CR)

2T

DRAM Idle Timer

Total CAS# (tRDRAM)

Row To Column (tRCD)

CPU-Z

Ver. 2.03.1.x64

Tools

Validate

Close