# Instituto Superior de Ciências do Trabalho e da Empresa

Departamento de Matemática

TECNOLOGIAS E ARQUITETURA


## Projeto OC

Elaborado por:


**João Almeida - a127766**
**Vicente Chã - a127688**


Orientadora:


**Professora Doutora Cristiana João Silva**


2 de dezembro de 2024

# *Lista de Equações*

2.1 Função para calcular a temperatura. . . . . . . . . . . . . . . . . . . . 5
2.2 Função objetivo, onde $n$ é o número de objetos, $x_i$ o valor binário do objeto, $v_i$ o benefício do objeto, $w_i$ o peso do objeto e $W$ a capacidade da mochila. . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.3 Função para calcular a probabilidade de aceitação. . . . . . . . . . . 5
2.4 Função objetivo, onde $n$ é o número de objetos, $x_i$ é o valor binário do objeto, $v_i$ o benefício do objeto, $w_i$ o peso do objeto e $W$ a capacidade da mochila. . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

# Conteúdo

**Lista de Equações** **i**

**Conteúdo** **iii**

**1 Introdução** **1**
1.1 Objetivos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Definição do Problema da Mochila . . . . . . . . . . . . . . . . . 1
1.3 Organização do Documento . . . . . . . . . . . . . . . . . . . . . 1

**2 Desenvolvimento** **3**
2.1 *Bottom-UP Approach* . . . . . . . . . . . . . . . . . . . . . . . . 3
2.2 *Simulated Annealing* . . . . . . . . . . . . . . . . . . . . . . . . 4
2.2.1 Análise do Problema . . . . . . . . . . . . . . . . . . . . 4
2.2.2 Análise da resolução . . . . . . . . . . . . . . . . . . . . 4
2.3 *Tabu Search* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.3.1 Análise do Problema . . . . . . . . . . . . . . . . . . . . 6
2.3.2 Solução Proposta . . . . . . . . . . . . . . . . . . . . . . 6

**3 Análise dos Dados** **9**
3.1 Resultados . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

**4 Conclusões** **11**
4.1 Tecnologias Utilizadas . . . . . . . . . . . . . . . . . . . . . . . . 11
4.2 Conclusão . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

**Bibliografia** **13**

*Capítulo*

# 1

# *Introdução*

## 1.1 Objetivos

O objetivo deste trabalho é implementar alguns métodos de otimização a problemas de otimização conhecidos e comparar os resultados de cada um. No caso deste trabalho o problema a ser resolvido é o problema da mochila, ou *knapsack problem*, para a sua resolução utilizamos três algoritmos diferentes: o *Bottom-UP Approach*(devolve sempre a solução ótima), o *Simulated Annealing* e o *Tabu Search*, os quais iremos explicar com mais detalhe no capítulo 2.

## 1.2 Definição do Problema da Mochila

O problema da mochila consiste conseguir organizar vários objetos com benefício **n** e peso **w** numa mochila com capacidade **W**, sendo que o objetivo é colocar os objetos na mochila de tal forma que a soma dos benefícios seja máxima.

Existem várias meta heurísticas para se aproximar da solução exata, no âmbito dos objetivos deste trabalho foram testados e implementados três métodos, duas meta heurísticas e um método para obter a solução ótima, todos são detalhadamente descritos nos próximos capítulos.

## 1.3 Organização do Documento

De modo a refletir o trabalho feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – Apresenta o projeto, os seus objetivos e a definição do problema a ser explorado, delineia também a respetiva organização do documento

2. O segundo capítulo – **Desenvolvimento** – Expõe e explica os vários métodos e algoritmos utilizados e os seus mecanismos.

3. O terceiro capítulo – **Análise dos Dados** – Aqui faz-se uma análise e comparação dos resultados obtidos através dos vários métodos utilizados.

4. O quarto capítulo – **Conclusões** – Descreve uma reflexão final sobre o trabalho, bem como as tecnologias utilizadas durante do desenvolvimento da aplicação.

*Capítulo*

# 2

# Desenvolvimento

## 2.1 *Bottom-UP Approach*

Utilizando o *Bottom-UP Approach*[1], construímos uma tabela em que cada célula $m[i][k]$ representa o valor máximo que pode ser obtido usando os primeiros $i$ objetos e uma capacidade de mochila de $k$.

Abaixo está um exemplo da tabela de programação dinâmica para uma mochila de capacidade 7 e 5 objetos:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Empty | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $v_1 = 2, w_1 = 3$ | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 |
| $v_2 = 2, w_2 = 1$ | 0 | 2 | 2 | 2 | 4 | 4 | 4 | 4 |
| $v_3 = 4, w_3 = 3$ | 0 | 2 | 2 | 4 | 6 | 6 | 6 | 8 |
| $v_4 = 5, w_4 = 4$ | 0 | 2 | 2 | 4 | 6 | 7 | 7 | 9 |
| $v_5 = 3, w_5 = 2$ | 0 | 2 | 3 | 5 | 6 | 7 | 9 | 10 |

- **Cálculo linha a linha:** Cada linha subsequente representa a inclusão de um novo objeto $i$, com valor $v_i$ e peso $w_i$. O valor de $m[i][k]$ é determinado da seguinte forma:
  - **Caso 1: O objeto $i$ não está incluído.** Se $w_i > k$, o objeto não pode ser incluído e herdamos o valor da linha anterior:

    $$m[i][k] = m[i-1][k]$$

  - **Caso 2: O objeto $i$ está incluído.** Se $w_i \leq k$, consideramos duas possibilidades:
    * Excluir objeto $i$, resultando no valor $m[i-1][k]$.

* Incluir o objeto $i$, adicionando o seu valor $v_i$ ao valor máximo que se pode obter com a capacidade restante $k - w_i$:

$$m[i][k] = v_i + m[i-1][k-w_i]$$

O valor final é o máximo destas duas opções.

## 2.2 *Simulated Annealing*

### 2.2.1 Análise do Problema

O *Simulated Annealing*[2] é um algoritmo de otimização que é inspirado pelo processo de *annealing* de metais. Onde é feito o aquecimento dos metais a temperaturas elevadíssimas durante um período fixo de tempo para depois ser arrefecido lentamente para remover defeitos, isto torna o metal mais resistente, maleável, flexível, etc., sendo que o tempo e condições do processo de arrefecimento dependem de metal para metal.[3]

No caso deste trabalho o *Simulated Annealing* é utilizado de forma semelhante, este tenta encontrar a melhor solução através de uma exploração dos dados durante o período de "arrefecimento"para tentar encontrar o melhor, enquanto apresenta "temperaturas"elevadas este método tem uma maior hipótese de optar por explorar mais os dados mesmo se tiver de escolher soluções piores, e enquanto a "temperatura"desce a probabilidade de este explorar mais os dados vai diminuindo, o que permite que o algoritmo se foque mais nos melhores resultados que encontra.

### 2.2.2 Análise da resolução

O algoritmo começa com a declaração de todas as variáveis iniciais, que neste caso são: A capacidade da mochila, o valor de cada objeto, o peso da cada objeto, o número de objetos, a temperatura inicial, a taxa de arrefecimento e o número de iterações, após isso é criada a solução atual, que é uma solução aleatória ao problema, e a melhor solução, que como está no início do algoritmo, vai ser uma cópia da solução atual.

Depois disso começa o algoritmo em si, e como referido anteriormente, este algoritmo funciona com base na temperatura, grande parte da exploração deste método vem da frequência e intensidade com que a temperatura desce, no caso deste trabalho a função de arrefecimento da temperatura por foi declarada da seguinte forma para uma melhor adaptação à realidade:

Onde a cada iteração a temperatura é reduzida para uma percentagem dela, sendo que essa percentagem está sempre entre 95% e 99%. Para assim o

$$Temperatura = Temperatura \cdot TaxaArrefecimento \tag{2.1}$$

Função para calcular a temperatura.

algoritmo conseguir ter bastantes oportunidades de fazer uma ampla procura no início do programa e no fim focar-se nos melhores valores.

Depois do arrefecimento da temperatura é calculado o vizinho, onde este é apenas uma cópia da solução temporária com apenas um dos seus objetos trocados, após a sua declaração calculamos o seu valor através da função apresentada em 2.2. Como o objetivo deste problema é maximizar o benefício total de cada objeto sem que o seu peso total ultrapasse a capacidade da mochila, calculamos o benefício e o peso total de todos os objetos do vizinho, se o peso total for maior que a capacidade da mochila o vizinho é penalizado com uma avaliação negativa do seu benefício, se os pesos não ultrapassarem a capacidade da mochila é calculado o benefício total dos seus objetos para comparação futura.

$$\begin{cases} \sum_{i=1}^{n} x_i \cdot v_i & \text{, se } \sum_{i=1}^{n} x_i \cdot w_i < W \\ -1 & \text{, se } \sum_{i=1}^{n} x_i \cdot w_i > W \end{cases} \tag{2.2}$$

Função objetivo, onde $n$ é o número de objetos, $x_i$ o valor binário do objeto, $v_i$ o benefício do objeto, $w_i$ o peso do objeto e $W$ a capacidade da mochila.

Após o cálculo do benefício do vizinho é feito o cálculo da probabilidade de aceitação, demonstrado na equação 2.3, este calculo é feito para permitir que o algoritmo possa aceitar soluções piores que a atual para o propósito de melhorar a sua procura no espaço do conjunto dos dados. Este cálculo depende muito da temperatura atual, uma vez que quando a temperatura é maior a probabilidade de aceitação é maior.

$$\begin{cases} \min\left(1, \exp\left(\frac{ValorVizinho - ValorAtual}{Temperatura}\right)\right) & \text{, se } temperatura < 0.01 \\ 0 & \text{, se } temperatura < 0.01 \end{cases} \tag{2.3}$$

Função para calcular a probabilidade de aceitação.

No final é verificado se o benefício do vizinho é maior que o benefício da solução atual e é gerado um número aleatório para comparar com a probabilidade de aceitação, se o benefício do vizinho for maior e/ou a probabilidade de aceitação for maior que o número aleatório, a solução atual toma o valor do vizinho e é verificado se este é maior que a melhor solução obtida, se sim, a melhor solução toma o valor do vizinho e o programa continua até concluir todas as iterações.

## 2.3 *Tabu Search*

### 2.3.1 Análise do Problema

O *Tabu Search*[4] é um algoritmo de otimização que utiliza busca de vizinhos e uma lista tabu para evitar que a busca revisite soluções já exploradas. Este é um método com inspiração na necessidade de balançar *exploitation vs exploration* em grandes espaços de busca, permitindo o escape de mínimos locais e indo ao encontro de candidatos a mínimos globais ao longo do processo. A lista tabu armazena movimentos proibidos temporariamente, garantindo que o algoritmo explore novas áreas do espaço de soluções.

### 2.3.2 Solução Proposta

Da mesma forma, o algoritmo inicia com a declaração de variáveis como a capacidade da mochila, os valores e pesos dos objetos, o número total de objetos, o tamanho da lista tabu(como um fator do número de objetos) e o número de iterações, até parar. A solução inicial é gerada aleatoriamente, e o algoritmo começa com a solução atual e a melhor solução sendo iguais a essa solução inicial.

A função objetivo utilizada para avaliar cada solução é representada pela equação 2.4. Nesta função, o objetivo é maximizar o benefício total dos objetos selecionados, penalizando soluções onde o peso total ultrapasse a capacidade da mochila.

$$\sum_{i=1}^{n}(v_i \cdot x_i) \cdot \left[1 - \max\left(0, \sum_{i=1}^{n}(w_i \cdot x_i) - W\right)\right] \tag{2.4}$$

Função objetivo, onde $n$ é o número de objetos, $x_i$ é o valor binário do objeto, $v_i$ o benefício do objeto, $w_i$ o peso do objeto e $W$ a capacidade da mochila.

O algoritmo gera a partir da primeira solução aleatória vários vizinhos que diferem da última iteração em apenas uma alteração binária, caso um vizinho não pertença à lista tabu, é considerado como solução candidata, cada candidato é avaliado com base na função objetivo, o melhor é encontrado e o processo continua da mesma forma com esse encontrado.

Inicialmente utilizávamos todos os vizinhos possíveis, mas usar apenas uma pequena amostra dos vizinhos torna o algoritmo muito mais eficiente.

Em cada iteração, o melhor vizinho é selecionado entre as soluções candidatas, e a solução atual é atualizada. Se a solução atual tiver um valor maior que a melhor solução registada até o momento, esta também é atualizada. A lista tabu é ajustada dinamicamente: novas soluções são adicionadas, e as mais antigas são removidas quando o tamanho máximo da lista é atingido.

Ao longo das iterações, o algoritmo mantém o registo do valor da solução atual, da melhor solução encontrada e da média das soluções na lista tabu, para fins de análise. O processo continua até atingir o número máximo de iterações, quando retorna a melhor solução encontrada.

Este processo assegura que o algoritmo realiza uma procura eficaz e eficiente no espaço de soluções, explorando diferentes regiões e evitando voltar a soluções já exploradas.

*Capítulo*

# 3

# *Análise dos Dados*

## 3.1 Resultados

Após a implementação de todos os métodos foram testadas as suas eficiências e eficácias em vários conjuntos de dados, o que é demonstrado na tabela 3.1, onde a coluna Tempo é o tempo de execução de cada algoritmo dependendo do conjunto de dados, e a coluna Derivação é a diferença entre o melhor resultado possível e o resultado obtido pelo algoritmo em questão.

| | **Tempo (ms)** | | | **Derivação** | | |
|---|---|---|---|---|---|---|
| | **M** | **S.A.** | **T.S.** | **M** | **S.A.** | **T.S.** |
| **pequeno** | 0.0008 | 0.0005 | 0.0020 | 0 | -40.7 | -21.0 |
| **médio** | 0.0493 | 0.0109 | 0.0055 | 0 | -355.1 | -184.9 |
| **grande** | 5.7584 | 0.6964 | 0.8876 | 0 | -1243.9 | -846.2 |
| **gigante** | 21.796 | 18.451 | 9.2404 | 0 | -1768.6 | -1004.56 |

Tabela 3.1: Tempo de execução e desvio ao resultado dos vários métodos

A tabela 3.1 apresenta todos os métodos utilizados neste trabalho, o *Bottom-UP Approach*, **M**, o *Simulated Annealing*, **S.A.** e o *Tabu Search*, **T.S.**. Estes métodos são comparados pelo seu desempenho em vários conjuntos de dados.

O conjunto pequeno, que são dados correspondentes a uma mochila com capacidade 50, 12 objetos com benefícios entre 10 e 60 e pesos entre 3 e 30. Ao observar a tabela podemos concluir que para um conjunto de dados pequeno o *Tabu search* é ligeiramente pior em termos de tempo de execução, mas melhor em resultados, no entanto, como o *Bottom-Up* dá sempre o melhor resultado, faz mais sentido utilizar este método para conjuntos de dados pequenos.

Para conjuntos de dados médios, mochila com capacidade de 900, 50 objetos com benefícios entre 10 e 150 e pesos entre 1 e 50. Podemos observar que a complexidade do *Bottom-Up* está a começar a mostrar-se, com o pior tempo de execução, em termos de resultados este continua a ser o melhor, e como a quantidade de dados aumentou o *Tabu* e o *Annealing* têm uma maior derivação do valor máximo. Mas à semelhança dos outros dados, ainda é preferível usar o *Bottom-Up*, pois este dá sempre o melhor resultado em tempo aceitável.

Para o próximo conjunto de dados foi utilizada uma mochila com capacidade 11000, 500 objetos com benefícios entre 10 e 150 e pesos entre 1 e 50, agora sim dá para ver a complexidade do *Bottom-Up*, com quase 5.7 segundos de tempo de execução comparados aos 0.7 e 0.9 segundos dos outros métodos, sendo que as derivações dos resultados continuam com o mesmo padrão. A partir deste ponto o *Bottom-Up* para de ser o melhor método para a pesquisa, sendo agora o *Tabu* a escolha ideal.

Para concluir temos o último conjunto de dados onde a mochila tem capacidade 20000, com 1000 objetos com benefícios entre 10 e 150 e pesos entre 1 e 50. Podemos observar o mesmo padrão que observamos no conjunto anterior, contudo agora o *Annealing* também sofre com a quantidade de dados utilizados, com um tempo de execução impressionante de 18.4 segundos, quase tanto quanto o *Bottom-Up*. Em termos de resultados, todos os algoritmos continuam com o mesmo padrão.

Ao fim da análise desta tabela podemos claramente concluir que o *Tabu search* é o melhor algoritmos em termos de tempo de execução comparado a resultados, no entanto, só será útil ser utilizado com grades conjuntos de dados, pois em conjuntos pequenos o *Bottom-Up*, apesar de ser mais demorado, devolve melhores resultados.

*Capítulo*

# 4

# *Conclusões*

## 4.1 Tecnologias Utilizadas

Para realizar este trabalho foi necessário a utilização de várias bibliotecas externas à biblioteca padrão do Python: *time* que fornece várias funções relacionadas ao tempo.*numpy*, utilizada para trabalhar com listas e funções matemáticas. *radom*, utilizada para gerar números aleatórios. *math*, utilizada para funções e constantes matemáticas. *matplotlib*, utilizada para gerar gráficos dos dados obtidos.

## 4.2 Conclusão

Com este trabalho explorámos várias formas de resolver o problema da mochila e comparamo-los todos para ver qual dele era o mais eficiente e eficaz em vários conjuntos de dados.

# Bibliografia

[1] WilliamFiset. 0/1 Knapsack problem | Dynamic Programming, 2017. Link: https://www.youtube.com/watch?v=cJ21moQpofY.

[2] Paulo Moura Oliveira. Simulated Annealing (Em Português), 2018. Link: https://www.youtube.com/watch?v=w2rBcPo88XM.

[3] Training within Industry (TWI). What is Annealing? A Complete Process Guide, 2024. Link: https://www.twi-global.com/technical-knowledge/faqs/what-is-annealing.

[4] Marcos Castro de Souza. Problema da Mochila Inteira, 2013. Link: https://github.com/marcoscastro/mochila_inteiro-busca_tabu.