# Trabalho 1 - Heurísticas e Metaheurísticas

```
#define TRABALHO 1
#define PROFESSOR "Guilherme Pena"
#define ESTUDANTE "Gabriel de Paula"
```

---

## 🧠 Metaheurísticas

Três metaheurísticas foram desenvolvidas para resolver dois problemas clássicos de otimização combinatória: o TSP (Problema do Caixeiro Viajante) e o KP (Problema da Mochila).

### 1) Simulated Annealing

Um algoritmo de otimização inspirado no processo de resfriamento de metais, que busca minimizar uma função de custo explorando soluções de forma probabilística, aceitando até mesmo soluções piores temporariamente para evitar ficar preso em ótimos locais.

| Parâmetros | Temperatura Inicial | Temperatura Mínima | Coeficiente de Resfriamento | Iterações |
|---|---|---|---|---|
| **Constantes** | T_INICIAL | T_MINIMO | A | SA_ITERACOES |

### 2) Busca Tabu

Uma técnica de otimização iterativa que utiliza uma lista de movimentos proibidos (tabu) para evitar ciclos e explorar o espaço de soluções de forma mais eficiente, frequentemente usada em problemas de otimização combinatória.

| Parâmetros | Iterações | Tamanho da Lista Tabu |
|---|---|---|
| **Constantes** | ITERACOES | LISTA_TABU_MAX |

### 3) GRASP

Um algoritmo de busca heurística que combina duas fases: uma construção gulosa, que constrói uma solução de forma iterativa, e uma fase de aprimoramento, que tenta melhorar a solução utilizando técnicas de busca local.

| Parâmetros | Iterações | Coeficiente de Aceitação |
|---|---|---|
| **Constantes** | ITERACOES | A |

## ⚙️ Compilação

Compilar todos os executáveis em modo otimizado:

```
make
```

Compilar todos os executáveis em modo otimizado, imprimindo todas as soluções aceitas:

```
make debug
```

---

## 🛠️ Execução

Programas gerados:

| Algoritmo | Problema | Programa |
|---|---|---|
| Simulated Annealing | TSP | sa-tsp |
| Simulated Annealing | KP | sa-kp |
| Busca Tabu | TSP | bt-tsp |
| Busca Tabu | KP | bt-kp |
| GRASP | TSP | grasp-tsp |
| GRASP | KP | grasp-kp |

Para executar um programa é necessário fornecer os valores numéricos dos parâmetros conforme indicado anteriormente.

```
./bin/<programa> <parametro1> <...> <parametroN>
```

Exemplo de como executar um programa 50 vezes para gerar relatórios:

```
for i in {1..50}; do ./bin/<programa> <parametro1> <...> <parametroN>; done
```

# 📊 Resultados

Foi realizada uma análise detalhada dos parâmetros de execução de todos os programas utilizando Grid Search, que explora o produto cartesiano de todas as combinações possíveis de parâmetros. Além do valor obtido, foi dada ênfase à análise do tempo decorrido.

Para algoritmos baseados em aleatoriedade, como Simulated Annealing e GRASP, foi calculada a média dos resultados obtidos em 10 execuções, garantindo mais consistência nos dados.

> Solução ótima TSP (min) = **426**
>
> Solução ótima KP (max) = **9147**

---

## Simulated Annealing

1. T_INICIAL = [100, 1000, 10000]
2. T_MINIMO = [0.1, 0.01, 0.001]
3. A = [0.5, 0.75, 0.9, 0.999]
4. SA_ITERACOES = [5, 10, 50, 100]

#### sa-tsp

É possível identificar uma característica comum entre os melhores resultados obtidos: um coeficiente de redução de temperatura próximo de 1 combinado com o número máximo de iterações.

| Parâmetros | Valor médio | Tempo médio |
|---|---|---|
| (100, 0.1, 0.999, 100) | **453.38** | 0.6234 s |
| (1000, 0.001, 0.999, 100) | 457.02 | 1.2246 s |
| (10000, 0.1, 0.999, 100) | 458.39 | 1.0618 s |
| (10000, 0.01, 0.999, 100) | 455.42 | 1.2613 s |
| (10000, 0.001, 0.999, 100) | 459.19 | 1.4437 s |

No entanto, também é possível alcançar bons resultados com tempos médios de execução menores ao reduzir o valor da temperatura inicial, desde que o coeficiente de redução permaneça próximo de 1 e o número de iterações continue elevado.

| Parâmetros | Valor médio | Tempo médio |
|---|---|---|
| (100, 0.1, 0.999, 50) | 469.69 | 0.3269 |
| (100, 0.1, 0.999, 100) | **453.38** | 0.6234 |
| (100, 0.01, 0.999, 50) | 468.25 | 0.4218 |
| (100, 0.01, 0.999, 100) | 463.18 | 0.8081 |
| (100, 0.001, 0.999, 50) | 469.77 | 0.5100 |

**sa-kp**

Houveram 26 combinações de parâmetros que conseguiram uma média igual à solução ótima e muitas outras se aproximaram. Importante destacar que todos possuem o coeficiente de redução próximo de 1.

| Parâmetros | Valor médio | Tempo médio |
|---|---|---|
| (100, 0.1, 0.999, 50) | 9147.00 | 0.0533 |
| (100, 0.1, 0.999, 100) | 9147.00 | 0.0748 |
| (100, 0.01, 0.999, 10) | 9147.00 | 0.0183 |
| (100, 0.01, 0.999, 50) | 9147.00 | 0.0530 |
| (100, 0.01, 0.999, 100) | 9147.00 | 0.0954 |
| (100, 0.001, 0.999, 50) | 9147.00 | 0.0691 |
| (100, 0.001, 0.999, 100) | 9147.00 | 0.1213 |
| (1000, 0.1, 0.999, 50) | 9147.00 | 0.0561 |
| (1000, 0.1, 0.999, 100) | 9147.00 | 0.0963 |
| (1000, 0.01, 0.999, 5) | 9147.00 | 0.0150 |
| (1000, 0.01, 0.999, 50) | 9147.00 | 0.0555 |
| (1000, 0.01, 0.999, 100) | 9147.00 | 0.1127 |
| (1000, 0.001, 0.999, 10) | 9147.00 | 0.0312 |
| (1000, 0.001, 0.999, 50) | 9147.00 | 0.0846 |
| (1000, 0.001, 0.999, 100) | 9147.00 | 0.1342 |
| (10000, 0.1, 0.999, 5) | 9147.00 | 0.0102 |
| (10000, 0.1, 0.999, 10) | 9147.00 | 0.0176 |
| (10000, 0.1, 0.999, 50) | 9147.00 | 0.0666 |
| (10000, 0.1, 0.999, 100) | 9147.00 | 0.1096 |
| (10000, 0.01, 0.999, 5) | 9147.00 | 0.0169 |
| (10000, 0.01, 0.999, 10) | 9147.00 | 0.0317 |
| (10000, 0.01, 0.999, 50) | 9147.00 | 0.0811 |
| (10000, 0.01, 0.999, 100) | 9147.00 | 0.1388 |
| (10000, 0.001, 0.999, 5) | 9147.00 | 0.0191 |
| (10000, 0.001, 0.999, 10) | 9131.30 | 0.0295 |
| (10000, 0.001, 0.999, 50) | 9147.00 | 0.0804 |
| (10000, 0.001, 0.999, 100) | 9147.00 | 0.1551 |

## Busca Tabu

1. ITERACOES = [10, 100, 1000, 10000]
2. LISTA_TABU_MAX = [1, 5, 10, 50, 100]

### bt-tsp

Por ser um algoritmo determinístico iterativo que visa melhorar uma solução prévia, é compreensível que mudanças drásticas não existam.

Contudo, o resultado é satisfatório. A solução inicial, gerada usando uma abordagem gulosa que seleciona sempre o vértice mais próximo, tem seu valor na função objetivo de `513.61`.

| Parâmetros | Valor | Tempo |
|---|---|---|
| (10, 1) | 469.32 | 0.0720 |
| (10, 5) | 469.32 | 0.0733 |
| (10, 10) | 469.32 | 0.0413 |
| (10, 50) | 469.32 | 0.0721 |
| (10, 100) | 469.32 | 0.0727 |
| (100, 1) | 469.32 | 0.4065 |
| (100, 5) | 469.32 | 0.3931 |
| (100, 10) | 469.32 | 0.4067 |
| (100, 50) | **466.23** | 0.3934 |
| (100, 100) | **466.23** | 0.4385 |
| (1000, 1) | 469.32 | 3.7328 |
| (1000, 5) | 469.32 | 3.5303 |
| (1000, 10) | 469.32 | 3.5782 |
| (1000, 50) | **466.23** | 3.8443 |
| (1000, 100) | **466.23** | 4.1113 |
| (10000, 1) | 469.32 | 37.2412 |
| (10000, 5) | 469.32 | 37.6833 |
| (10000, 10) | 469.32 | 35.7464 |
| (10000, 50) | **466.23** | 39.6062 |
| (10000, 100) | **466.23** | 41.8037 |

Nota-se também que aumentar consideravelmente o número de iterações para 10000 não tornou o resultado melhor que 100 iterações, cujo tempo de execução foi disparadamente menor.

**bt-kp**

Utilizando uma das melhores heurísticas para a construção da solução inicial (seleção dos itens com maior custo-benefício), os resultados obtidos foram surpreendentemente positivos. Todas as combinações testadas alcançaram o valor de `8929`, partindo de uma solução inicial de `8817`.

Para avaliar o desempenho em condições menos favoráveis, o algoritmo foi testado com uma das piores heurísticas para a solução inicial (seleção dos itens com maior peso). Mesmo assim, houve uma melhora significativa nos resultados, considerando que a solução inicial gerada por essa heurística mais fraca foi de apenas `794`.

| Parâmetros | Valor médio | Tempo médio |
|---|---|---|
| (10, 1) | 4787 | 0.0029 |
| (10, 5) | 4787 | 0.0018 |
| (10, 10) | 4787 | 0.0032 |
| (10, 50) | 4787 | 0.0024 |
| (10, 100) | 4787 | 0.0036 |
| (100, 1) | 4787 | 0.0040 |
| (100, 5) | **7967** | 0.0061 |
| (100, 10) | **7967** | 0.0065 |
| (100, 50) | **7967** | 0.0033 |
| (100, 100) | **7967** | 0.0048 |
| (1000, 1) | 4787 | 0.0100 |
| (1000, 5) | **7967** | 0.0115 |
| (1000, 10) | **7967** | 0.0280 |
| (1000, 50) | **7967** | 0.0152 |
| (1000, 100) | **7967** | 0.0165 |
| (10000, 1) | 4787 | 0.0872 |
| (10000, 5) | **7967** | 0.0953 |
| (10000, 10) | **7967** | 0.1380 |
| (10000, 50) | **7967** | 0.1559 |
| (10000, 100) | **7967** | 0.1886 |

Assim como no caso do TSP, a Busca Tabu foi capaz de encontrar melhor solução dos testes já com 100 iterações e com valores tamanhos reduzidos de lista tabu, mostrando como nem sempre o exagero traz benefícios

## GRASP

1. ITERACOES = [10, 100, 1000, 10000]
2. A = [0.0625, 0.125, 0.25, 0.5, 0.75]

**grasp-tsp**

Quanto maior o coeficiente de aceitação, mais aleatória é a solução e, por consequência, piores são os resultados obtidos, já que sobra mais para a busca local fazer as melhorias.

| Parâmetros | Valor médio | Tempo médio |
|---|---|---|
| (10, 0.0625) | 501.85 | 0.0235 |
| (10, 0.125) | 535.50 | 0.0261 |
| (10, 0.25) | 738.41 | 0.0239 |
| (10, 0.5) | 963.40 | 0.0591 |
| (10, 0.75) | 1221.38 | 0.0561 |
| (100, 0.0625) | 475.42 | 0.2362 |
| (100, 0.125) | 542.90 | 0.2361 |
| (100, 0.25) | 678.72 | 0.2376 |
| (100, 0.5) | 940.46 | 0.2432 |
| (100, 0.75) | 1197.48 | 0.2468 |
| (1000, 0.0625) | 465.58 | 2.0723 |
| (1000, 0.125) | 518.64 | 2.0372 |
| (1000, 0.25) | 645.72 | 2.0400 |
| (1000, 0.5) | 898.12 | 2.0668 |
| (1000, 0.75) | 1060.18 | 2.0611 |
| (10000, 0.0625) | **451.32** | 19.7353 |
| (10000, 0.125) | 490.11 | 20.2872 |
| (10000, 0.25) | 627.47 | 20.3972 |
| (10000, 0.5) | 864.70 | 20.3062 |
| (10000, 0.75) | 1076.18 | 20.5717 |

Soluções aceitáveis foram encontradas mesmo com baixo número de iterações, porém todas as melhores soluções foram obtidas justamente com o menor coeficiente de aceitação, demonstrando que a intenção desse algoritmo é gerar pequenas perturbações na construção dos caminhos e não depositar todas as esperanças na aleatoriedade.

**grasp-kp**

Novamente é possível observar a correlação do coefiente de aceitação com o valor médio encontrado. Apesar disso, os resultados não foram tão consistentes quanto no TSP, mostrando que problemas distintos podem apresentar diferentes aspectos para um mesmo algoritmo.

| Parâmetros | Valor médio | Tempo médio |
|---|---|---|
| (10, 0.0625) | 8693.00 | 0.0143 |
| (10, 0.125) | 8563.00 | 0.0153 |
| (10, 0.25) | 8693.00 | 0.0162 |
| (10, 0.5) | 7216.00 | 0.0174 |
| (10, 0.75) | 5941.00 | 0.0188 |
| (100, 0.0625) | **8900.00** | 0.0679 |
| (100, 0.125) | 8693.00 | 0.0460 |
| (100, 0.25) | 8658.00 | 0.0778 |
| (100, 0.5) | 7428.00 | 0.0796 |
| (100, 0.75) | 5639.00 | 0.0848 |
| (1000, 0.0625) | **8900.00** | 0.4504 |
| (1000, 0.125) | 8817.00 | 0.4566 |
| (1000, 0.25) | **8900.00** | 0.4315 |
| (1000, 0.5) | 7516.00 | 0.4847 |
| (1000, 0.75) | 6124.00 | 0.5193 |
| (10000, 0.0625) | 8693.00 | 4.2622 |
| (10000, 0.125) | 8693.00 | 4.2184 |
| (10000, 0.25) | 8693.00 | 4.2933 |
| (10000, 0.5) | 7437.00 | 4.4120 |
| (10000, 0.75) | 5410.00 | 4.8726 |

Houve uma combinação diferente das outras, beneficiada pela aleatoriedade, que com coefiente 0.25 conseguiu encontrar a mesma solução que foi a melhor, mostrando que é realmente importante testar diferentes parâmetros.