P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

**Análise Algorítmica e Otimização**

# Problema de Localização de Instalações sem Restrições de Capacidade

**Sérgio Ferreira nº8210134**
**Tiago Oliveira nº8210137**
**Bruno Duarte nº8210120**
**Paulo Coelho nº8210132**
**Carlos Fernandes nº8210121**

# Índice

# Índice de Figuras

# Índice de Tabelas

# Lista de Siglas e Acrónimos

| Sigla | Significado |
|---|---|
| **UFLP** | Problema de Localização de Instalações sem Restrições de Capacidade |
| **VNS** | Algoritmo Variable Neighborhood Search |

# 1. Introdução

O presente relatório tem como objetivo explorar e analisar o Problema de Localização de Instalações sem Restrições de Capacidade (*UFLP*) e foi realizado no seguimento do trabalho prático da unidade curricular de Análise Algorítmica e Otimização, com o intuito de aplicar e aprofundar conhecimentos sobre algoritmos de otimização e a sua aplicação em problemas reais.

Este relatório pretende não só apresentar uma pesquisa bibliográfica abrangente sobre o UFLP, mas também descrever a implementação e análise de diferentes algoritmos de resolução deste problema. A análise comparativa dos algoritmos permitirá avaliar a eficiência e a eficácia das abordagens utilizadas, fornecendo uma compreensão mais profunda sobre as melhores práticas na resolução deste tipo de problema.

## 1.1 Contextualização

O Problema de Localização de Instalações sem Restrições de Capacidade (UFLP) é um dos problemas clássicos em otimização combinatória e teoria de localização. Este problema surge em diversas áreas práticas, como logística, gestão de cadeias de abastecimento e distribuição de produtos. O UFLP aborda a decisão sobre onde localizar instalações, tais como armazéns ou centros de distribuição, de modo a minimizar os custos totais, que incluem os custos de operação das instalações, bem como os custos de transporte dos produtos até aos clientes. Este problema é amplamente estudado devido à sua aplicação prática em cenários reais, onde decisões eficientes podem resultar em economias significativas.

## 1.2 Apresentação do Caso de Estudo

Para a realização deste trabalho, foi disponibilizado um conjunto de ficheiros que contêm instâncias de teste para o Problema de Localização de Instalações sem Restrições de Capacidade (UFLP). Estes ficheiros são originários do conjunto de problemas testados por J.E. Beasley no seu artigo “Na algorithm for solving large capacitated warehouse location problems” publicado no European Journal of Operational Research, em 1988.

Os ficheiros de dados incluem um total de 40 instâncias de teste, categorizadas em diferentes conjuntos de problemas. Cada conjunto é composto por vários ficheiros que variam

em termos de número de localizações potenciais para instalações e número de clientes. Os ficheiros incluem a capacidade e o custo fixo para cada instalação, bem como a procura e o custo de alocação de cada cliente a cada instalação. Neste caso, não foram contabilizadas as capacidades das instalações, bem como também foi ignorada a demanda dos clientes.

Estes ficheiros foram utilizados para testar os algoritmos implementados no presente trabalho. Cada instância de teste permite avaliar o desempenho dos algoritmos em termos de eficiência computacional e qualidade das soluções obtidas. Os resultados ótimos conhecidos para cada uma dessas instâncias foram também utilizados como referência para validar as soluções obtidas pelos algoritmos desenvolvidos.

## 1.3 Motivação e Objetivos

O UFLP tem enorme relevância prática em áreas como logística, gestão de cadeias de abastecimento e distribuição de produtos, como mencionado anteriormente, o que o torna um tópico de grande interesse tanto para a nível académico como para indústria. A capacidade de resolver eficientemente este problema pode resultar em reduções significativas nos custos operacionais e de transporte, oferecendo uma vantagem competitiva às empresas.

Além disso, o UFLP é um problema clássico de otimização combinatória, o que proporciona uma excelente oportunidade para aplicar e testar diferentes algoritmos de otimização. Esta aplicação prática é essencial para consolidar os conhecimentos adquiridos na unidade curricular.

Os principais objetivos deste trabalho passam por:

- **Compreender a fundo o UFLP**.
- **Implementar e comparar diferentes algoritmos de resolução do UFLP**: Abrange tanto algoritmos exatos quanto heurísticos e metaheurísticos, avaliando a sua eficiência e eficácia.
- **Avaliar a performance dos algoritmos**: Medir o tempo de execução e a qualidade das soluções obtidas, comparando-os com os resultados ótimos conhecidos.
- **Propor melhorias e novas abordagens**: Com base na análise dos resultados, sugerir possíveis melhorias nos algoritmos existentes ou desenvolver novas abordagens para resolver o UFLP.

A realização destes objetivos permitirá uma compreensão das técnicas de otimização aplicadas ao UFLP e contribuirá para o desenvolvimento de soluções mais eficientes para problemas de localização de instalações em contextos reais.

## 1.4 Estrutura do Relatório

Este relatório está organizado da seguinte forma: No Capítulo 2, é apresentada a pesquisa bibliográfica sobre o Problema de Localização de Instalações sem Restrições de Capacidade

(*UFLP*), começando pela definição e formulação do problema, seguida por uma revisão dos algoritmos exatos e heurísticos utilizados para a sua resolução. No Capítulo 3, detalha-se a implementação de vários algoritmos, incluindo o Algoritmo *Swap*, Algoritmo *Switch*, *Variable Neighborhood Search* (VNS) e *Branch and Bound*, abrangendo a descrição, implementação, vantagens, desvantagens e resultados obtidos para cada um. No Capítulo 4, realiza-se uma análise do desempenho dos algoritmos implementados, discutindo a eficácia de cada um em termos de precisão e tempo de execução, culminando numa conclusão comparativa que orienta sobre a escolha do algoritmo mais adequado conforme o contexto e as necessidades específicas do problema.

# 2 Pesquisa bibliográfica sobre o Problema de Localização de Instalações sem Restrições de Capacidade

## 2.1 Definição e Formulação do Problema

O Problema de Localização de Instalações sem Restrições de Capacidade (*UFLP*), como referido anteriormente, é um problema de otimização combinatória onde o objetivo é determinar a localização de um conjunto de instalações de forma a minimizar os custos locais (ou maximizar os lucros) que incluem tanto os custos fixos de operação das instalações quanto os custos variáveis de transporte dos produtos até aos clientes [1].

**Definição Formal**

De acordo com a literatura, o UFLP pode ser descrito da seguinte forma:

- **Conjunto de Clientes**: I = {1, 2, …, n}, cada um com uma demanda específica por uma mercadoria.
- **Conjunto de Potenciais Localizações**: J = {1, 2, …, m}, onde as instalações podem ser localizadas.
- **Custos Fixos**: f*j* representa o custo fixo de abrir uma instalação na localização *j.*
- **Lucros ou Custos de Transporte(C*ij*)**: Lucro ou custo associado a satisfazer a demanda do cliente *i* a partir da instalação *j.*

O objetivo é encontrar um subconjunto *S* de localizações que minimize o custo total ou maximize o lucro total, assegurando que toda a demanda dos clientes seja atendida.

**Formulação Matemática**

Para formular o UFLP como um problema de programação linear, utilizamos as seguintes variáveis de decisão:

- X*j* = 1 se a instalação está aberta, X*j* = 0 caso contrário.
- Y*ij* = 1 se a demanda do cliente *i* é satisfeita pela instalação *j*, Y*ij* = 0 caso contrário.

A formulação de programação linear é:

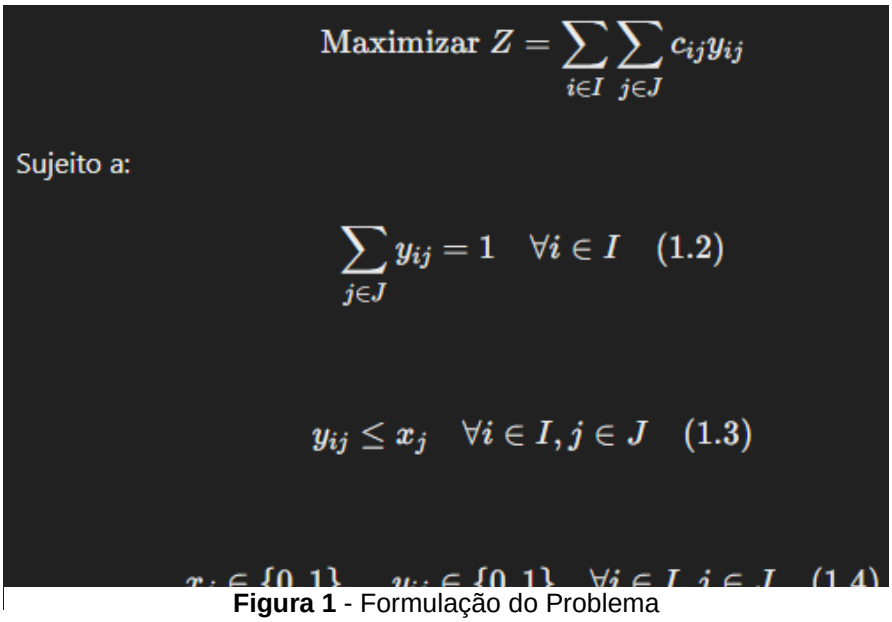


**Figura 1** - Formulação do Problema

De acordo com a Figura 1, as restrições (1.2) garantem que a demanda de cada cliente é satisfeita, assegurando que cada cliente é servido por exatamente uma instalação. As restrições (1.3) garantem que os clientes são atendidos apenas por instalações abertas, ou seja, a demanda do cliente *i* só pode ser satisfeita pela instalação *j* se esta estiver aberta. Finalmente, as restrições (1.4) definem que as variáveis X*j* e Y*ij* são binárias, significando que uma instalação ou está aberta ou não, e que a demanda de um cliente ou é satisfeita por uma instalação ou não [1].

## 2.2 Algoritmos para Resolução do UFLP

Os algoritmos utilizados para a resolução de problemas de otimização combinatória podem ser classificados em duas categorias: algoritmos exatos e algoritmos heurísticos.

### 2.2.1 Algoritmos Exatos

Os algoritmos exatos garantem a obtenção da solução ótima, mas o tempo computacional necessário para encontrar a solução ótima é proporcional à dimensão do problema. Portanto, os algoritmos exatos têm limitações significativas quando se trata de resolver em tempo útil problemas de grande dimensão. Exemplos conhecidos de algoritmos exatos incluem o método "*Simplex*" e o "*Branch and Bound"* [2].

### 2.2.2 Algoritmos Heurísticos

Os algoritmos heurísticos podem ser divididos em três grupos distintos: construtivos, de melhoramento (ou pesquisa local) e meta heurísticos.

- **Algoritmos Construtivos**: Estes algoritmos constroem uma solução de forma incremental, escolhendo em cada iteração um componente para adicionar à solução até que uma solução admissível seja encontrada [2].
- **Algoritmos de Melhoramento ou Pesquisa Local**: Estes algoritmos perturbam a solução corrente para gerar uma nova solução de melhor qualidade utilizando uma estrutura de vizinhança. O processo continua até que não seja possível melhorar mais a solução com a estrutura de vizinhança escolhida, resultando num ótimo local [2].
- **Algoritmos Metaheurísticos**: Para superar a limitação dos métodos de melhoramento que podem ficar presos em ótimos locais, surgem os algoritmos metaheurísticos. Estes algoritmos conseguem escapar da vizinhança de um ótimo local, mesmo que a solução corrente piore temporariamente, com o objetivo de explorar outras partes do espaço de soluções e encontrar o ótimo global. Exemplos de algoritmos metaheurísticos incluem *Simulated Annealing*, Algoritmos Genéticos, e GRASP (*Greedy Randomized Adaptive Search Procedures*) [2].

# 3 Implementação de algoritmos para a resolução do Problema de Localização de Instalações sem Restrições de Capacidade

A otimização do Problema de Localização de Instalações sem Restrições de Capacidade é essencial para reduzir custos operacionais e melhorar a eficiência em diversas áreas. Neste segmento do relatório, exploramos a implementação e o desempenho de vários algoritmos, desde heurísticas a métodos exatos, para encontrar soluções eficazes e eficientes. Avaliamos as abordagens com base na qualidade das soluções geradas e na eficiência computacional.

## 3.1 Algoritmo Swap

O Algoritmo Swap Heuristic é uma técnica heurística usada na resolução do Problema de Localização de Instalações sem Restrições de Capacidade (UFLP). Este algoritmo baseia-se

na ideia de melhorar progressivamente uma solução inicial através da troca sistemática do estado de abertura de pares de instalações. A meta é reduzir o custo total associado à configuração das instalações e à alocação de clientes a essas instalações.

### 3.1.1 Descrição do Algoritmo

O processo inicia-se com uma configuração inicial onde as instalações são alternadamente definidas como abertas ou fechadas. Esta configuração inicial é arbitrariamente escolhida para garantir uma distribuição equilibrada e facilitar a exploração do espaço de solução. Cada iteração do algoritmo envolve avaliar todas as combinações possíveis de trocas entre pares de instalações (uma aberta e outra fechada) e aplicar a troca que resulta na maior redução de custo. O critério de paragem ocorre quando uma passagem completa pelas instalações não resulta em nenhuma melhoria de custo, indicando que um mínimo local foi possivelmente alcançado.
O custo total é calculado somando-se os custos fixos das instalações abertas e os custos de alocação dos clientes às instalações mais próximas que estejam abertas.

### 3.1.2 Implementação do Algoritmo

**Solução Inicial:** A solução inicial é gerada de forma aleatória, mas controlada, para garantir diversidade nas configurações iniciais sem comprometer a eficiência computacional. Inicialmente, as instalações são atribuídas aleatoriamente como abertas ou fechadas, com aproximadamente metade das instalações abertas. Este método proporciona uma base equilibrada sobre a qual o algoritmo pode começar a explorar o espaço de soluções.

```
private static void initializeSolution(ProblemInstance instance) {
    for (int i = 0; i < instance.getFacilities().size(); i++) {
        instance.getFacilities().get(i).setOpen(i % 2 == 0);
    }
}
```

**Figura 2 -** Solução inicial - Excerto de código

**Mecanismo de Atuação:** Durante cada iteração do algoritmo, todas as possíveis trocas de estado entre pares de instalações (uma aberta e outra fechada) são consideradas. Para cada par, o algoritmo calcula o custo total se o estado das duas instalações fosse trocado e compara com o custo atual. Se a troca resultar numa redução do custo total, a troca é efetuada.

```
for (int i = 0; i < facilities.size(); i++) {
    for (int j = i + 1; j < facilities.size(); j++) {
        if (trySwap(currentInstance, i, j)) {
            improvement = true;
        }
    }
}
```

**Figura 3 -** trySwap - Excerto de código 1

**Cálculo do Custo e Avaliação da Troca:** A função `trySwap` desempenha o papel de decidir se uma troca deve ser realizada:

```
private static boolean trySwap(ProblemInstance instance, int indexFacility1, int indexFacility2) {
    Facility facility1 = instance.getFacilities().get(indexFacility1);
    Facility facility2 = instance.getFacilities().get(indexFacility2);

    // Calcular o custo antes da troca
    double currentCost = calculateTotalCost(instance);

    // Trocar o estado das instalações
    toggleFacility(facility1);
    toggleFacility(facility2);

    // Calcular o custo após a troca
    double newCost = calculateTotalCost(instance);

    if (newCost < currentCost) {
        return true; // A troca reduz o custo total, então mantém a troca
    } else {
        // Desfazer a troca se não reduzir o custo
        toggleFacility(facility1);
        toggleFacility(facility2);
        return false;
    }
}
```

**Figura 4 -** trySwap - Excerto de código 2

**Critério de Paragem:**

O algoritmo termina as iterações quando uma passagem completa por todas as combinações possíveis de pares de instalações não resulta em nenhuma redução do custo total. Este critério de paragem garante que o algoritmo só termina quando se atinge um estado em que não são possíveis mais melhorias, indicando que uma solução localmente ótima foi provavelmente alcançada

### 3.1.3 Vantagens

- **Simplicidade de Implementação:** O algoritmo *Swap* é relativamente simples de implementar e compreender.

- **Flexibilidade:** É aplicável a uma vasta gama de problemas de otimização, podendo ser facilmente ajustado para diferentes configurações de problemas.

### 3.1.4 Desvantagens

- **Possibilidade de convergência para mínimos locais:** O algoritmo pode ficar preso em mínimos locais, especialmente em instâncias complexas com muitas instalações, onde pode não ser capaz de explorar todas as configurações ótimas.
- **Dependência da Solução Inicial:** A qualidade da solução final pode depender da solução inicial.

### 3.1.5 Resultados

**Tabela 1 -** Resultados obtidos – Método: *Swap*

| Nome Ficheiro | m | n | Solução Ótima | Solução Obtida | Desvio (%) | Tempo de execução (ms) |
|---|---|---|---|---|---|---|
| Cap71.txt | 16 | 50 | 932615,750 | 933568,900 | 0,102 | 25 |
| cap72 | 16 | 50 | 977799,400 | 978876,300 | 0,110 | 26 |
| cap73 | 16 | 50 | 1010641,450 | 1010808,162 | 0,016 | 29 |
| cap74 | 16 | 50 | 1034976,975 | 1034976,975 | 0,000 | 29 |
| cap101 | 25 | 50 | 796648,437 | 796648,438 | 0,000 | 33 |
| cap102 | 25 | 50 | 854704,200 | 854704,200 | 0,000 | 29 |
| cap103 | 25 | 50 | 893782,112 | 894008,138 | 0,025 | 32 |
| cap104 | 25 | 50 | 928941,750 | 929460,975 | 0,056 | 29 |
| cap131 | 50 | 50 | 793439,562 | 793439,563 | 0,000 | 46 |
| cap132 | 50 | 50 | 851495,325 | 851495,325 | 0,000 | 45 |
| cap133 | 50 | 50 | 893076,712 | 894801,163 | 0,193 | 40 |
| cap134 | 50 | 50 | 928941,750 | 929460,975 | 0,056 | 45 |
| capa | 100 | 1000 | 17156454,478 | 17156454,478 | 0,000 | 2127 |
| capb | 100 | 1000 | 12979071,582 | 13193415,967 | 1,625 | 2000 |
| capc | 100 | 1000 | 11505594,329 | 11509361,660 | 0,033 | 2035 |
| KCAPMO1 | 100 | 100 | 1156,909 | 1156,909 | 0,000 | 239 |
| KCAPMO2 | 100 | 100 | 1227,667 | 1234,302 | 0,538 | 230 |
| KCAPMO3 | 100 | 100 | 1286,369 | 1294,996 | 0,666 | 241 |
| KCAPMO4 | 100 | 100 | 1177,880 | 1186,336 | 0,713 | 171 |

| | | | | | | |
|---|---|---|---|---|---|---|
| KCAPMO5 | 100 | 100 | 1147,595 | 1147,595 | 0,000 | 242 |
| KCAPMP1 | 200 | 200 | 2460,101 | 2477,842 | 0,716 | 3338 |
| KCAPMP2 | 200 | 200 | 2419,325 | 2419,325 | 0,000 | 3603 |
| KCAPMP3 | 200 | 200 | 2498,151 | 2521,701 | 0,934 | 3282 |
| KCAPMP4 | 200 | 200 | 2633,561 | 2633,561 | 0,000 | 3804 |
| KCAPMP5 | 200 | 200 | 2290,164 | 2292,061 | 0,083 | 4564 |
| KCAPMQ1 | 300 | 300 | 3591,273 | 3601,477 | 0,283 | 15813 |
| KCAPMQ2 | 300 | 300 | 3543,662 | 3543,662 | 0,000 | 15096 |
| KCAPMQ3 | 300 | 300 | 3476,806 | 3476,806 | 0,000 | 20110 |
| KCAPMQ4 | 300 | 300 | 3742,474 | 3756,766 | 0,380 | 21551 |
| KCAPMQ5 | 300 | 300 | 3751,326 | 3780,799 | 0,780 | 16215 |
| KCAPMR1 | 500 | 500 | 2349,856 | 2378,254 | 1,194 | 80128 |
| KCAPMR2 | 500 | 500 | 2344,757 | 2349,710 | 0,211 | 165811 |
| KCAPMR3 | 500 | 500 | 2183,235 | 2183,235 | 0,000 | 122409 |
| KCAPMR4 | 500 | 500 | 2433,110 | 2457,233 | 0,982 | 118392 |
| KCAPMR5 | 500 | 500 | 2344,353 | 2369,653 | 1,068 | 154705 |

## 3.2 Algoritmo Switch

O Algoritmo Switch utiliza uma abordagem heurística para o Problema de Localização de Instalações sem Restrições de Capacidade (UFLP), centrando-se na modificação do estado de abertura de uma única instalação por iteração para explorar reduções no custo total.

### 3.2.1 Descrição do Algoritmo

Inicia-se com uma configuração inicial onde as instalações são alternadamente definidas como abertas ou fechadas, semelhante ao processo utilizado no Algoritmo Swap. O objetivo é percorrer cada instalação individualmente, avaliando o impacto no custo total de alternar o seu estado. O processo continua até que não seja possível encontrar reduções adicionais significativas no custo, sugerindo a obtenção de uma solução localmente ótima.

### 3.2.2 Implementação do Algoritmo

**Solução Inicial:** A configuração inicial segue o mesmo princípio do Algoritmo Swap, promovendo uma base inicial diversificada que permite uma ampla exploração inicial do espaço de soluções.

**Mecanismo de Atuação:** A implementação prática do algoritmo considera cada instalação individualmente, calculando o impacto no custo total de abrir ou fechar a instalação em questão. O código a seguir ilustra este processo:

```
private static boolean trySwitch(ProblemInstance instance, int indexFacility) {
    Facility facility = instance.getFacilities().get(indexFacility);

    // Calcular o custo antes da troca
    double currentCost = calculateTotalCost(instance);

    // Trocar o estado da instalação
    toggleFacility(facility);

    // Calcular o custo após a troca
    double newCost = calculateTotalCost(instance);

    if (newCost < currentCost) {
        return true; // A troca reduz o custo total, então mantém a troca
    } else {
        // Desfazer a troca se não reduzir o custo
        toggleFacility(facility);
        return false;
    }
}
```

**Figura 5 -** *trySwitch* - Excerto de Código

**Critério de Paragem:** Similar ao Swap, o algoritmo termina as iterações quando uma passagem completa por todas as instalações não resulta em melhorias no custo total.

### 3.2.3 Vantagens

- **Simplicidade de Implementação:** Este algoritmo é relativamente simples de implementar, seguindo uma lógica direta de avaliação de custos para cada instalação individualmente.
- **Rápida Identificação de Melhorias:** Capaz de identificar rapidamente mudanças efetivas que resultam em economias significativas.

### 3.2.4 Desvantagens

- **Limitação na Exploração de Soluções:** Não explora todas as combinações possíveis como no Swap, o que pode resultar em soluções sub ótimas.
- **Dependência da Solução Inicial:** A qualidade da solução final pode ser fortemente influenciada pela configuração inicial, como observado no Swap.

### 3.2.5 Resultados

**Tabela 2 -** Resultados obtidos – Método: *Switch*

| Nome Ficheiro | m | n | Solução Ótima | Solução Obtida | Desvio (%) | Tempo de execuçao (ms) |
|---|---|---|---|---|---|---|
| cap71.txt | 16 | 50 | 932615,750 | 932615,750 | 0,000 | 17 |
| cap72 | 16 | 50 | 977799,400 | 977799,400 | 0,000 | 18 |
| cap73 | 16 | 50 | 1010641,450 | 1010641,450 | 0,000 | 18 |
| cap74 | 16 | 50 | 1034976,975 | 1037717,075 | 0,264 | 16 |
| cap101 | 25 | 50 | 796648,437 | 800004,975 | 0,420 | 19 |
| cap102 | 25 | 50 | 854704,200 | 854704,200 | 0,000 | 20 |
| cap103 | 25 | 50 | 893782,112 | 893782,113 | 0,000 | 26 |
| cap104 | 25 | 50 | 928941,750 | 934586,975 | 0,604 | 19 |
| cap131 | 50 | 50 | 793439,562 | 801723,213 | 1,033 | 27 |
| cap132 | 50 | 50 | 851495,325 | 858821,213 | 0,853 | 26 |
| cap133 | 50 | 50 | 893076,712 | 898830,850 | 0,640 | 27 |
| cap134 | 50 | 50 | 928941,750 | 929477,563 | 0,058 | 24 |
| capa | 100 | 1000 | 17156454,478 | 19029558,849 | 9,843 | 66 |
| capb | 100 | 1000 | 12979071,582 | 14106721,112 | 7,994 | 69 |
| capc | 100 | 1000 | 11505594,329 | 11996381,347 | 4,091 | 83 |
| KCAPMO1 | 100 | 100 | 1156,909 | 1166,529 | 0,825 | 38 |
| KCAPMO2 | 100 | 100 | 1227,667 | 1238,181 | 0,849 | 36 |
| KCAPMO3 | 100 | 100 | 1286,369 | 1378,510 | 6,684 | 38 |
| KCAPMO4 | 100 | 100 | 1177,880 | 1258,518 | 6,407 | 37 |
| KCAPMO5 | 100 | 100 | 1147,595 | 1159,473 | 1,024 | 36 |
| KCAPMP1 | 200 | 200 | 2460,101 | 2468,198 | 0,328 | 113 |
| KCAPMP2 | 200 | 200 | 2419,325 | 2419,325 | 0,000 | 99 |
| KCAPMP3 | 200 | 200 | 2498,151 | 2519,447 | 0,845 | 80 |
| KCAPMP4 | 200 | 200 | 2633,561 | 2685,286 | 1,926 | 85 |
| KCAPMP5 | 200 | 200 | 2290,164 | 2446,709 | 6,398 | 71 |
| KCAPMQ1 | 300 | 300 | 3591,273 | 3614,192 | 0,634 | 239 |
| KCAPMQ2 | 300 | 300 | 3543,662 | 3654,766 | 3,040 | 234 |
| KCAPMQ3 | 300 | 300 | 3476,806 | 3498,521 | 0,621 | 234 |

| | | | | | | |
|---|---|---|---|---|---|---|
| KCAPMQ4 | 300 | 300 | 3742,474 | 3844,345 | 2,650 | 203 |
| KCAPMQ5 | 300 | 300 | 3751,326 | 3846,041 | 2,463 | 167 |
| KCAPMR1 | 500 | 500 | 2349,856 | 2372,370 | 0,949 | 997 |
| KCAPMR2 | 500 | 500 | 2344,757 | 2401,228 | 2,352 | 991 |
| KCAPMR3 | 500 | 500 | 2183,235 | 2207,000 | 1,077 | 1051 |
| KCAPMR4 | 500 | 500 | 2433,110 | 2532,259 | 3,915 | 1117 |
| KCAPMR5 | 500 | 500 | 2344,353 | 2423,694 | 3,274 | 882 |

## 3.3 Algoritmo Variable Neighborhood Search (VNS)

O *Variable Neighborhood Search* (VNS) é uma técnica meta-heurística sofisticada que procura superar as limitações das heurísticas tradicionais ao explorar sistematicamente múltiplas vizinhanças para encontrar soluções globalmente eficientes para o Problema de Localização de Instalações sem Restrições de Capacidade (UFLP).

### 3.3.1 Descrição do Algoritmo

O VNS começa com uma solução inicial, que pode ser gerada de forma aleatória ou derivada de um método heurístico simples, e progride através de mudanças estratégicas dentro de diversas vizinhanças. A cada passo, o algoritmo avalia se explorar uma nova vizinhança produz uma solução melhor, incrementando progressivamente o raio de pesquisa se nenhuma melhoria for encontrada dentro da vizinhança atual.

### 3.3.2 Implementação do Algoritmo

**Solução Inicial e Escolha do Número de Vizinhanças:** A solução inicial para o VNS geralmente é gerada aleatoriamente ou por meio de uma heurística simples, e o número de vizinhanças (*maxNeighborhood*) é um parâmetro crucial que define a profundidade e amplitude da pesquisa. A escolha do número de vizinhanças depende da complexidade do problema e do equilíbrio desejado entre tempo de execução e qualidade da solução. Tipicamente, mais vizinhanças permitem uma exploração mais abrangente, mas aumentam o tempo computacional.

Neste caso a solução inicial é gerada aleatoriamente para cada instalação, decidindo se estará aberta ou fechada, proporcionando uma base diversificada para a procura inicial.

```java
private static ArrayList<Facility> initializeSolution(ProblemInstance instance) {
    ArrayList<Facility> solution = new ArrayList<>();
    Random random = new Random();
    for (Facility facility : instance.getFacilities()) {
        Facility newFacility = new Facility(facility.getId(), facility.getFixedCost());
        newFacility.setOpen(random.nextBoolean());
        solution.add(newFacility);
    }
    return solution;
}
```

**Figura 6 -** Solução inicial VNS - Excerto de código

**Mecanismo de Atuação:** O VNS executa uma série de iterações onde explora diferentes vizinhanças. A função `shake` é usada para introduzir mudanças aleatórias na solução atual, ajustando o estado de abertura de um número definido de instalações. A `localSearch` procura melhorias locais continuamente até que nenhuma melhoria possa ser feita na vizinhança atual.

```java
private static ArrayList<Facility> shake(ProblemInstance instance, ArrayList<Facility> solution, int k) {
    Random random = new Random();
    for (int i = 0; i < k; i++) {
        int index = random.nextInt(solution.size());
        Facility facility = solution.get(index);
        facility.setOpen(!facility.isOpen());
    }
    return solution;
}

private static ArrayList<Facility> localSearch(ProblemInstance instance, ArrayList<Facility> solution) {
    boolean improvement = true;
    while (improvement) {
        improvement = false;
        double currentCost = calculateTotalCost(instance, solution);

        for (Facility facility : solution) {
            facility.setOpen(!facility.isOpen());
            double newCost = calculateTotalCost(instance, solution);

            if (newCost < currentCost) {
                currentCost = newCost;
                improvement = true;
            } else {
                facility.setOpen(!facility.isOpen());
            }
        }
    }
    return solution;
}
```

**Figura 7 -** *shake* e *localSearch* VNS - Excerto de código

**Critério de Paragem:** O VNS termina após um número predefinido de iterações sem melhorias, garantindo que a solução não pode mais ser otimizada com as vizinhanças definidas.

### 3.3.3 Vantagens:

- **Eficácia em escapar de mínimos locais:** Graças à sua estratégia de múltiplas vizinhanças, o VNS é eficaz em superar soluções sub ótimas locais.
- **Flexibilidade:** Adapta-se bem a uma variedade de problemas devido à sua capacidade de ajustar o número e tipo de vizinhanças.

### 3.3.4 Desvantagens:

- **Complexidade na Configuração:** Requer um entendimento profundo das estruturas de vizinhança e uma implementação cuidadosa dos parâmetros.
- **Dependência dos Parâmetros:** O desempenho do algoritmo pode variar consideravelmente com a escolha e ajuste dos parâmetros, incluindo o número de vizinhanças exploradas.

### 3.3.5 Resultados

**Tabela 3 Resultados obtidos – Método: VNS**

| Nome Ficheiro | m | n | Solução ótima | Solução obtida | Desvio (%) | Tempo de execução (ms) |
|---|---|---|---|---|---|---|
| cap71.txt | 16 | 50 | 932615,750 | 932615,750 | 0,000 | 184 |
| cap72 | 16 | 50 | 977799,400 | 977799,400 | 0,000 | 200 |
| cap73 | 16 | 50 | 1010641,450 | 1010641,450 | 0,000 | 195 |
| cap74 | 16 | 50 | 1034976,975 | 1034976,975 | 0,000 | 191 |
| cap101 | 25 | 50 | 796648,437 | 796648,438 | 0,000 | 311 |
| cap102 | 25 | 50 | 854704,200 | 854704,200 | 0,000 | 343 |
| cap103 | 25 | 50 | 893782,112 | 893782,113 | 0,000 | 293 |
| cap104 | 25 | 50 | 928941,750 | 928941,750 | 0,000 | 275 |
| cap131 | 50 | 50 | 793439,562 | 793439,563 | 0,000 | 875 |
| cap132 | 50 | 50 | 851495,325 | 851495,325 | 0,000 | 919 |
| cap133 | 50 | 50 | 893076,712 | 893076,713 | 0,000 | 898 |
| cap134 | 50 | 50 | 928941,750 | 928941,750 | 0,000 | 722 |
| capa | 100 | 1000 | 17156454,478 | 17156454,478 | 0,000 | 45685 |
| capb | 100 | 1000 | 12979071,582 | 13087893,450 | 0,831 | 55673 |
| capc | 100 | 1000 | 11505594,329 | 11651459,534 | 1,252 | 52795 |
| KCAPMO1 | 100 | 100 | 1156,909 | 1156,909 | 0,000 | 6500 |
| KCAPMO2 | 100 | 100 | 1227,667 | 1227,667 | 0,000 | 7494 |

| | | | | | | |
|---|---|---|---|---|---|---|
| KCAPMO3 | 100 | 100 | 1286,369 | 1286,369 | 0,000 | 6474 |
| KCAPMO4 | 100 | 100 | 1177,880 | 1177,880 | 0,000 | 5565 |
| KCAPMO5 | 100 | 100 | 1147,595 | 1147,595 | 0,000 | 5760 |
| KCAPMP1 | 200 | 200 | 2460,101 | 2460,101 | 0,000 | 44641 |
| KCAPMP2 | 200 | 200 | 2419,325 | 2419,325 | 0,000 | 48374 |
| KCAPMP3 | 200 | 200 | 2498,151 | 2498,151 | 0,000 | 46477 |
| KCAPMP4 | 200 | 200 | 2633,561 | 2633,561 | 0,000 | 63718 |
| KCAPMP5 | 200 | 200 | 2290,164 | 2290,164 | 0,000 | 46722 |
| KCAPMQ1 | 300 | 300 | 3591,273 | 3591,273 | 0,000 | 182954 |
| KCAPMQ2 | 300 | 300 | 3543,662 | 3543,662 | 0,000 | 131088 |
| KCAPMQ3 | 300 | 300 | 3476,806 | 3476,806 | 0,000 | 193458 |
| KCAPMQ4 | 300 | 300 | 3742,474 | 3742,474 | 0,000 | 233283 |
| KCAPMQ5 | 300 | 300 | 3751,326 | 3751,326 | 0,000 | 200860 |
| KCAPMR1 | 500 | 500 | 2349,856 | 2349,856 | 0,000 | 772176 |
| KCAPMR2 | 500 | 500 | 2344,757 | 2344,757 | 0,000 | 892224 |
| KCAPMR3 | 500 | 500 | 2183,235 | 2183,235 | 0,000 | 912668 |
| KCAPMR4 | 500 | 500 | 2433,110 | 2433,110 | 0,000 | 798029 |
| KCAPMR5 | 500 | 500 | 2344,353 | 2344,353 | 0,000 | 854545 |

## 3.4 Algoritmo *Branch and Bound*

O Algoritmo *Branch and Bound* é uma abordagem exata utilizada para resolver o Problema de Localização de Instalações sem Restrições de Capacidade (UFLP), distinguindo-se por sua capacidade de encontrar a solução ótima, explorando sistematicamente todas as combinações possíveis de instalações abertas e fechadas.

### 3.4.1 Descrição do Algoritmo

*Branch and Bound* começa com uma configuração inicial e procede de forma a examinar todas as possíveis configurações de abertura e fecho de instalações. Este método utiliza uma abordagem sistemática de divisão e limitação (*branching and bounding*), para eliminar rapidamente grandes conjuntos de soluções subótimas, concentrando-se nas áreas do espaço de solução que podem conter a solução ótima.

### 3.4.2 Implementação do Algoritmo

**Mecanismo de Atuação:** O algoritmo executa recursivamente, alternando o estado de cada instalação entre aberto e fechado, e propaga essa decisão para o próximo nível de decisão até que todas as instalações tenham sido consideradas.

```
public static Solution solveExact(ProblemInstance instance) {
    long startTime = System.currentTimeMillis();

    int numFacilities = instance.getFacilities().size();
    boolean[] openFacilities = new boolean[numFacilities];

    System.out.println("A iniciar o metodo Branch and Bound...");
    // Inicializa a procura no nivel 0
    branchAndBound(instance, openFacilities,  level: 0);

    long endTime = System.currentTimeMillis();
    long executionTime = endTime - startTime;

    System.out.println("Branch and Bound concluido.");
    return new Solution(new ArrayList<>(bestSolution), bestCost, executionTime);
}

private static void branchAndBound(ProblemInstance instance, boolean[] openFacilities, int level) {
    if (level == openFacilities.length) {
        double currentCost = calculateTotalCost(instance, openFacilities);
        if (currentCost < bestCost) {
            bestCost = currentCost;
            bestSolution = getOpenFacilities(instance, openFacilities);
            System.out.println("Nova melhor solução encontrada: Custo = " + bestCost);
        }
        return;
    }
```

**Figura 8 -** *Branch and Bound* - Excerto de código

**Critério de Paragem:** O algoritmo conclui quando todas as possíveis combinações de abertura e encerramento das instalações foram exploradas, garantindo que a melhor solução encontrada é a ótima.

### 3.4.3 Vantagens

- **Precisão:** Como uma técnica exata, o *Branch and Bound* é capaz de encontrar a solução ótima para o problema, desde que tenha tempo e recursos computacionais suficientes

### 3.4.4 Desvantagens

- **Escalabilidade:** Tende a ser computacionalmente intensivo e impraticável para problemas de grande escala devido ao crescimento exponencial do número de configurações possíveis.

- **Dependência do Tamanho da Instância**: A viabilidade de alcançar uma solução no tempo razoável diminui drasticamente à medida que o tamanho da instância aumenta.

### 3.4.5 Resultados

**Tabela 4 -** Resultados obtidos – Método: *Branch and Bound*

| Nome do Ficheiro | m | n | Solução Ótima | Solução Obtida | Desvio (%) | Tempo de execuçao (ms) |
|---|---|---|---|---|---|---|
| cap71.txt | 16 | 50 | 932615,750 | 932615,750 | 0,000 | 58 |
| cap72 | 16 | 50 | 977799,400 | 977799,400 | 0,000 | 58 |
| cap73 | 16 | 50 | 1010641,450 | 1010641,450 | 0,000 | 58 |
| cap74 | 16 | 50 | 1034976,975 | 1034976,975 | 0,000 | 58 |
| cap101 | 25 | 50 | 796648,437 | 796648,438 | 0,000 | 30619 |
| cap102 | 25 | 50 | 854704,200 | 854704,200 | 0,000 | 30520 |
| cap103 | 25 | 50 | 893782,112 | 893782,113 | 0,000 | 30689 |
| cap104 | 25 | 50 | 928941,750 | 928941,750 | 0,000 | 30443 |
| cap131 | 50 | 50 | 793439,562 | 801723,213 | 0,000 | 72574093 (2h>) |
| cap132 | 50 | 50 | 851495,325 | 858821,213 | 0,000 | 77011442 (2h>) |
| cap133 | 50 | 50 | 893076,712 | | | |
| cap134 | 50 | 50 | 928941,750 | | | |
| capa | 100 | 1000 | 17156454,478 | | | |
| capb | 100 | 1000 | 12979071,582 | | | |
| capc | 100 | 1000 | 11505594,329 | | | |
| KCAPMO1 | 100 | 100 | 1156,909 | | | |
| KCAPMO2 | 100 | 100 | 1227,667 | | | |
| KCAPMO3 | 100 | 100 | 1286,369 | | | |
| KCAPMO4 | 100 | 100 | 1177,880 | | | |
| KCAPMO5 | 100 | 100 | 1147,595 | | | |
| KCAPMP1 | 200 | 200 | 2460,101 | | | |
| KCAPMP2 | 200 | 200 | 2419,325 | | | |
| KCAPMP3 | 200 | 200 | 2498,151 | | | |
| KCAPMP4 | 200 | 200 | 2633,561 | | | |
| KCAPMP5 | 200 | 200 | 2290,164 | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| KCAPMQ1 | 300 | 300 | 3591,273 | | | |
| KCAPMQ2 | 300 | 300 | 3543,662 | | | |
| KCAPMQ3 | 300 | 300 | 3476,806 | | | |
| KCAPMQ4 | 300 | 300 | 3742,474 | | | |
| KCAPMQ5 | 300 | 300 | 3751,326 | | | |
| KCAPMR1 | 500 | 500 | 2349,856 | | | |
| KCAPMR2 | 500 | 500 | 2344,757 | | | |
| KCAPMR3 | 500 | 500 | 2183,235 | | | |
| KCAPMR4 | 500 | 500 | 2433,110 | | | |
| KCAPMR5 | 500 | 500 | 2344,353 | | | |

# 4. Análise do desempenho dos algoritmos implementados

Esta seção compara e discute o desempenho dos quatro algoritmos implementados para resolver o Problema de Localização de Instalações sem Restrições de Capacidade (UFLP): SWAP, SWITCH, VNS e *Branch and Bound*. A análise baseia-se nos dados recolhidos para diversos tamanhos de problema e configurações, considerando a solução ótima conhecida, a solução obtida, o desvio percentual da solução ótima e o tempo de execução.

## 4.1 Analise do Algoritmo SWAP

O algoritmo SWAP mostrou eficiência para instâncias menores e médias, com tempos de execução consistentemente baixos e desvios mínimos em relação à solução ótima. Para instâncias maiores, a performance em precisão decai ligeiramente, mas ainda mantém resultados aceitáveis. Este algoritmo é recomendado para aplicações onde o tempo de resposta é crítico e as soluções não precisam ser perfeitamente ótimas, mas sim suficientemente próximas.

## 4.2 Analise do Algoritmo SWITCH

Similar ao SWAP em termos de tempo de execução, o SWITCH tende a mostrar desvios maiores em instâncias de maior tamanho, com desempenho variando mais significativamente. Embora eficaz para problemas menores, sua performance piora em escala, tornando-o menos ideal para instâncias grandes. Este método pode ser preferido em cenários onde alterações mínimas na configuração inicial são necessárias e aceitáveis.

## 4.3 Analise do Algoritmo VNS

O VNS destaca-se pela alta precisão das soluções, atingindo resultados próximos ou iguais à solução ótima em todas as instâncias testadas. No entanto, isso vem com um aumento no tempo de execução, especialmente visível em instâncias maiores. O VNS é ideal para situações onde a precisão da solução é mais crítica do que a velocidade de execução, como em decisões estratégicas de longo prazo que beneficiam de uma análise mais detalhada e cuidadosa.

### 4.4 Analise do Algoritmo *Branch and Bound*

O *Branch and Bound* provou a sua eficácia ao alcançar soluções ótimas para todas as instâncias que conseguiu completar dentro do limite de tempo prático, que foi de até duas horas para as maiores instâncias testadas. Embora seja o único método que garante a otimalidade, o tempo de execução pode-se tornar um limitador severo para instâncias ainda maiores, onde não se testou devido à falta de viabilidade prática. Este método é mais adequado para problemas onde a obtenção da solução exata é imperativa e o tempo de execução é viável dentro do contexto operacional.

**Conclusão Comparativa Final**

Ao escolher entre estes algoritmos, a decisão deve considerar principalmente o tamanho da instância e a necessidade de precisão versus o tempo de execução:

- **Para instâncias pequenas a médias** onde a velocidade é crítica, **SWAP** e **SWITCH** são recomendados.
- **Para situações onde a precisão é crítica** e o tempo menos relevante, o **VNS** oferece uma excelente escolha.
- **Para casos onde a solução ótima é necessária** e o tempo de execução é viável, o ***Branch and Bound*** é o método de escolha, especialmente em cenários menos escaláveis.

Cada método tem seu nicho dependendo dos requisitos específicos de precisão, tempo e escala, permitindo escolher a ferramenta mais adequada conforme a necessidade.

# 5. Conclusões e Trabalho Futuro

**Conclusões**

Neste trabalho, abordámos o Problema de Localização de Instalações sem Restrições de Capacidade (UFLP), implementando e comparando diferentes algoritmos de resolução, incluindo métodos exatos *(Branch and Bound*) e heurísticos (*Swap*, *Switch*, *VNS*).

**Pontos Fortes:**

- **Diversidade de Algoritmos**: A análise de diversos algoritmos proporcionou uma compreensão abrangente das suas capacidades e limitações.
- **Análise Comparativa**: Avaliação detalhada do desempenho dos algoritmos em termos de precisão e tempo de execução.
- **Relevância Prática**: A utilização de instâncias de teste realistas reforçou a aplicabilidade prática dos resultados.

**Pontos Fracos:**

- **Limitações Computacionais**: O algoritmo *Branch and Bound* mostrou-se ineficiente para grandes instâncias.
- **Dependência da Solução Inicial**: Algoritmos como *Swap* e *Switch* dependem fortemente da solução inicial.
- **Escalabilidade**: Algoritmos metaheurísticos apresentaram tempos de execução elevados para instâncias maiores.

**Trabalho Futuro**

Sugere-se a continuidade da investigação nas seguintes direções:

- **Melhoria das Heurísticas**: Desenvolver heurísticas híbridas para melhorar a qualidade das soluções.
- **Paralelização**: Implementar versões paralelas dos algoritmos para aumentar a eficiência.
- **Análise de Sensibilidade**: Avaliar o impacto das variações de custos nas soluções.
- **Novas Metaheurísticas**: Explorar outras metaheurísticas, como Algoritmos de Colónia de Formigas.
- **Casos Práticos**: Aplicar os algoritmos em contextos empresariais reais para validar as soluções.

Em suma, este trabalho contribuiu significativamente para a compreensão e resolução do UFLP, oferecendo uma base sólida para futuras investigações e aplicações práticas

## 6. Referências Bibliográficas

[1] G. L. N. a. L. A. W. Gerard Cornuejols, “The Uncapacitated Facility Location Problem,” Pittsburgh, Pennsylvania, 1983.

[2] T. Matos, “RAMP para o Problema de Localização de Instalações com Restrições de Capacidade,” Felgueiras, Portugal, 2011.

# 7. Apêndice

**Código Fonte**

Zip Folder: TP_AAO.zip