A picture containing drawing, clock

Description automatically generated

**UFLP - Problema de localização de instalações sem capacidade: Estudo de algoritmos heurísticos**

César Castelo - 8220169

Hugo Guimarães - 8220337

João Santos - 8220256

Pedro Pinho - 8220307

Sónia Oliveira - 8220114

**Análise Algorítmica e Otimização**

# Índice

[Índice ii](#_Toc169689234)

[Índice de Figuras iv](#_Toc169689235)

[Índice de Tabelas v](#_Toc169689236)

[Lista de Siglas e Acrónimos vi](#_Toc169689237)

[1. Introdução 1](#_Toc169689238)

[1.1 Contextualização 1](#_Toc169689239)

[1.2 Apresentação do Caso de Estudo 1](#_Toc169689240)

[1.3 Motivação e Objetivos 1](#_Toc169689241)

[1.4 Estrutura do Relatório 2](#_Toc169689242)

[2. Pesquisa bibliográfica sobre o Problema de Localização de Instalações sem Restrições de Capacidade 3](#_Toc169689243)

[3. Implementação de algoritmos para a resolução do Problema de Localização de Instalações sem Restrições de Capacidade 5](#_Toc169689244)

[3.1 Algoritmo de Abertura Aleatória de Instalações 5](#_Toc169689245)

[3.2 Algoritmo de Greedy 2](#_Toc169689246)

[3.3 Algoritmo Swap | 2-opt 2](#_Toc169689247)

[3.4 Algoritmo Switch 2](#_Toc169689248)

[3.5 Algoritmo de Pesquisa Tabu 2](#_Toc169689249)

[3.5 Algoritmo F&F 3](#_Toc169689250)

[4. Análise do desempenho dos algoritmos implementados 4](#_Toc169689251)

[4.1 Heurísticos Construtivos 4](#_Toc169689252)

[Abertura Aleatória de Instalações 4](#_Toc169689253)

[Greedy 5](#_Toc169689254)

[4.2 Heurísticos de Pesquisa Local 6](#_Toc169689255)

[Swap 6](#_Toc169689256)

[Switch 7](#_Toc169689257)

[4.3 Meta Heurísticos 8](#_Toc169689258)

[Pesquisa Tabu 8](#_Toc169689259)

[Filter and Fan 9](#_Toc169689260)

[5. Complexidade dos algoritmos 11](#_Toc169689261)

[6. Características do projeto 15](#_Toc169689262)

[6.2 Linguagem Escolhida 15](#_Toc169689263)

[6.1 Aceleração da execução dos algoritmos 15](#_Toc169689264)

[7. Anexos 16](#_Toc169689265)

[8. Conclusões e Trabalho Futuro 27](#_Toc169689266)

[Referências Bibliográficas 28](#_Toc169689267)

[Referências WWW 29](#_Toc169689268)

# Índice de Figuras

[Figura 1 - Função que lê os dados das instâncias em txt 16](#_Toc169689269)

[Figura 2 - Função que calcula o custo total de uma solução 17](#_Toc169689270)

[Figura 3 – Algoritmo heurístico construtivo para abrir instalações aleatórias 18](#_Toc169689271)

[Figura 4 - Algoritmo heurístico construtivo de greedy 19](#_Toc169689272)

[Figura 5 - Algoritmo de pesquisa local Swap 20](#_Toc169689273)

[Figura 6 - Função que gera uma solução inicial e chama o Swap 20](#_Toc169689274)

[Figura 7 - Algoritmo de pesquisa local Switch 21](#_Toc169689275)

[Figura 8 - Função que constrói uma solução inicial e chama o Switch 21](#_Toc169689276)

[Figura 9 - função que verifica se uma determinada função está na lista tabu 22](#_Toc169689277)

[Figura 10 - Algoritmo meta heurístico de pesquisa tabu 23](#_Toc169689278)

[Figura 11 - Função que gera a solução inicial e chama o algoritmo de pesquisa tabu 24](#_Toc169689279)

[Figura 12 - Algoritmo de pesquisa local Swap usado pelo algoritmo meta heurístico Filter & Fan 24](#_Toc169689280)

[Figura 13 - Função que gera uma lista de candidatos segundo uma solução inicial 25](#_Toc169689281)

[Figura 14 - Algoritmo Filter & Fan 25](#_Toc169689282)

[Figura 15 - Função que gera uma solução inicial e chama o algoritmo meta heurístico F&F 26](#_Toc169689283)

# Índice de Tabelas

[Tabela 1 - Siglas e Acrónimos vi](#_Toc169689284)

[Tabela 2 - Resultados das iterações do algoritmo da abertura aleatória para o ficheiro Kcampo1.txt 5](#_Toc169689285)

[Tabela 3 - Resultados obtidos do algoritmo heurístico construtivo de abertura aleatória de instalações 4](#_Toc169689286)

[Tabela 4 - Análise dos resultados obtidos do algoritmo construtivo de greedy 5](#_Toc169689287)

[Tabela 5 - Resultados obtidos do algoritmo de pesquisa local swap 6](#_Toc169689288)

[Tabela 6 - Resultados obtidos do algoritmo de pesquisa local Switch 7](#_Toc169689289)

[Tabela 7 - Resultados obtidos do algoritmo meta heurístico Tabu Search 8](#_Toc169689290)

[Tabela 8 - Resultados obtidos do algoritmo meta heurístico Filter and Fan 9](#_Toc169689291)

# Lista de Siglas e Acrónimos

|  |  |
| --- | --- |
| **Sigla** | **Significado** |
| UFLP | Uncapacited Facility localtion Problem |
| F&F | Filter and Fan |
| CSV | Comma Separated Values |

Tabela 1 - Siglas e Acrónimos

# 1. Introdução

## 1.1 Contextualização

Este trabalho foi realizado para o âmbito da disciplina de Análise Algorítmica e Otimização, e consiste na utilização dos conhecimentos adquiridos no decorrer das aulas teóricas para o estudo e análise de algoritmos para um problema muito popular de localização de instalações sem restrições de capacidade (UFLP).

## 1.2 Apresentação do Caso de Estudo

O problema de localização de instalações (FLP), originalmente introduzido por Stollsteimer (1961) e Balinski (1966) tem objetivo descobrir o um número indeterminado de instalações de modo a minimizar os custos de abertura das instalações, e os custos de transportes para os clientes. Uma variante deste problema surgiu com o tempo: a localização de instalações sem restrições de capacidade (UFLP), aqui é assumido que cada instalação tem um limite virtualmente ilimitado, mesmo este problema sendo uma versão muito mais simplificada do problema original, ele provou ser um problema do tipo NP-Difícil.

A utilização de métodos exatos na resolução de problemas práticos de grande dimensão pode ser seriamente comprometida pelos tempos computacionais necessários para a obtenção da solução ótima. Várias tentativas de resolução exata têm sido propostas utilizando dualidade (Erlenkotter 1978) e relaxação Lagrangeana (Cornuéjols et al. 1977; Erlenkotter 1978; Guignard 1988; Korkel 1989; Beasley 1993). Para ultrapassar esta dificuldade, uns números significativos de algoritmos heurísticos de vários tipos têm sido propostos com o objetivo de encontrarem soluções de boa qualidade em tempos tão reduzidos quanto possível.

## 1.3 Motivação e Objetivos

Como o problema de localização de instalações sem restrições de capacidade é um problema do tipo NP-Difícil, o que significa que encontrar a solução ótima em tempo razoável é impraticável para grandes instâncias devido ao crescimento exponencial do tempo de computação necessário. Devido a essa complexidade têm surgido diversos algoritmos para tentar resolver este problema, no entanto como o uso de métodos exatos podem não oferecer uma solução ótima em tempo útil, diversas abordagens heurísticas surgiram de modo a conseguir uma solução muito perto do ideal em tempo útil com o mínimo de recursos computacionais possíveis.

Métodos exatos, embora possam garantir a obtenção da solução ótima, frequentemente não são viáveis em termos de tempo computacional, especialmente para instâncias de grande escala. Como resultado, diversas abordagens heurísticas surgiram como alternativas viáveis. Mais uma vez, essas heurísticas têm a vantagem de fornecer soluções muito próximas do ideal em um tempo consideravelmente menor e com uso reduzido de recursos computacionais.

O principal objetivo deste trabalho é implementar diversas abordagens heurísticas famosas para o problema de localização de instalações sem restrição de capacidade, e aplicar os conhecimentos de análise algorítmica adquiridos no decorrer das aulas teóricas da disciplina de Análise Algorítmica e Otimização para avaliar e comparar essas heurísticas.

## 1.4 Estrutura do Relatório

Este relatório está organizado em oito partes: a primeira parte é uma introdução ao problema que se vai estudar neste relatório; a segunda parte é uma pesquisa bibliográfica sobre o problema de localização de instalações sem restrições de capacidade; a terceira parte fala sobre os algoritmos implementados para a resolução do mesmo; na quarta parte irão ser montadas tabelas com os resultados de cada algoritmo e irá ser feita uma análise dos mesmos para saber qual abordagem deu melhor resultados, e porquê; na quinta parte os algoritmos implementados serão desmontados e analisar-se-á a complexidade dos mesmos segundo a escala Big O; na sexta parte são descritos alguns detalhes e características importantes sobre o trabalho desenvolvido; a sétima parte é uma zona de anexos que irá mostrar o código desenvolvido para os algoritmoss; por último, na oitava parte é feita uma reflexão sobre o trabalho realizado e o que se acha que ainda há para melhorar.

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

O problema de localização de instalações é um problema bastante popular que tem como principal objetivo decidir em que locais devem ser colocadas instalações de modo a suprir as necessidades dos clientes com o menor custo possível, sendo que cada cliente tem um custo fixo de transporte para cada instalação, e cada instalação tem um custo fixo de abertura.

O problema de instalação de instalações está dividido em dois sub-problemas principais, sendo eles:

* **Com Restrições de Capacidade -** Neste cenário, cada instalação tem uma capacidade máxima de produção ou atendimento que não pode ser excedida. As decisões sobre a localização das instalações precisam considerar tanto o custo quanto a capacidade de cada instalação, assegurando que a demanda total dos clientes seja atendida sem ultrapassar as capacidades das instalações abertas. O desafio é encontrar uma solução que minimize os custos totais, respeitando as restrições de capacidade.
* **Sem Restrições de Capacidade -** Neste cenário, não há limites para a capacidade de atendimento das instalações. Isso significa que qualquer instalação pode, teoricamente, atender a toda a demanda dos clientes. O foco nesta variante está na minimização dos custos de abertura e transporte, sem a necessidade de se preocupar com a capacidade das instalações.

Este trabalho foca-se apenas no problema de localização de instalações sem restrições de capacidade, que é um problema do tipo NP-Difícil, onde não existem algoritmos conhecidos que consigam garantir a solução ótima para todas as instâncias do problema em um tempo que cresce de forma polinomial, isto quer dizer que é computacionalmente desafiador encontrar uma solução ótima em um período razoável à medida que o tamanho do problema aumenta. Como resultado, métodos heurísticos e meta heurísticos são frequentemente empregues para encontrar soluções quase ótimas com eficiência.

Várias abordagens de solução, como pesquisa local, meta heurísticas e métodos exatos baseados em dualidade e relaxações Lagrangeanas, foram propostas para lidar com a UFLP e encontrar soluções de alta qualidade para instâncias do mundo real com problemas de grandes tamanhos e estruturas complexas.

A primeira heurística apresentada para o problema de localização de instalações sem restrições de capacidade foi apresentada em 1963 por Kuehn e Hamburger, onde o problema é dividido em duas fases: na primeira fase é realizado um algoritmo de greedy, onde todas as instalações começam fechadas, e este tenta melhorar a solução ao abrir uma instalação em cada iteração de modo a diminuir o custo total, parando quando já não existir nenhuma instalação que irá contribuir para a melhora do custo total; na segunda fase é aplicado um algoritmo de pesquisa local Swap, onde pares de instalações trocam os seus estados, mais uma vez este procedimento é repetido iterativamente até que não exista mais nenhum movimento de troca que melhore a solução.

Como o problema de localização de instalações sem restrições de capacidade é um problema estudado há bastante tempo, surgiram diversas abordagens heurísticas, como: pesquisa tabu (Ghosh 2003; Michel e Hentenryck 2004; Resende e Werneck 2006; Sun 2006); redes neurais artificiais (Gen et al. 1996; Vaithyanathan et al. 1996); surgiram até alguns algoritmos inspirados na natureza, como: o algoritmo genético (Kratica et al. 2001); e o algoritmo de otimização de colónia de formigas (Marco Dorigo 1992).

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

## 3.1 Algoritmo de Abertura Aleatória de Instalações

Para este algoritmo foi implementado uma função em python que recebe como parâmetros: uma matriz com o custo de transporte de cada instalação para cada cliente ; um vetor com o custo de abertura de cada uma das instalações, e o número de instalações que o algoritmo deve abrir.

Este algoritmo é um algoritmo heurístico construtivo, logo ele não vai tentar encontrar uma solução ótima, apenas vai gerar uma solução inicial para ser usado num algoritmo heurístico de pesquisa local ou meta heurístico. No entanto, mesmo que não se procure uma solução ótima, deve-se tentar fazer com que as soluções dadas ofereçam alguma qualidade, para tentar garantir alguma qualidade fez-se um breve estudo, onde executou-se o algoritmo duzentas vezes para o ficheiro *Kcampmo1.txt* com diferentes números de instalações abertas e fazer uma média dos resultados, e com base nas médias obtidas selecionou-se aquela mais perto da solução ótima para esse problema.

Para o estudo, decidiu-se que o algoritmo deveria ser executado 200 vezes para cada número de instalações abertas, obtendo no final os resultados da Tabela 2, onde se encontra a média de parte dos resultados obtidos com o número de instalações que foram abertas, e como é possível visualizar, os resultados mais próximos da solução ótima para o ficheiro *Kcampmo1.txt* (1156,909) foi oito instalações.

Este estudo foi realizado apenas para um ficheiro, no entanto estes resultados não serão os mesmos para os outros ficheiros que serão estudados, visto que eles não contêm todos os mesmos números de clientes e instalações, portanto, para cada ficheiro, usar-se-á este como referência, e irá fazer-se uma regra de três simples arredondada para baixo para saber o número de instalações que serão abertas. Se para 100 instalações no ficheiro *Kcampmo1.txt* foram precisas ser abrir 8 instalações, então para o ficheiro o ficheiro , com instalações, serão abertas instalações*.*

Tabela 2 - Resultados das iterações do algoritmo da abertura aleatória para o ficheiro Kcampo1.txt

|  |  |
| --- | --- |
| **N.º Instalações Abertas** | **Média** |
| 1 | 270.357 |
| 2 | 247.414 |
| 3 | 420.330 |
| 4 | 530.470 |
| 5 | 690.752 |
| 6 | 860.619 |
| 7 | 1014.839 |
| 8 | 1163.076 |
| 9 | 1314.433 |
| 10 | 1460.739 |
| 11 | 1534.202 |
| 12 | 1724.791 |
| 13 | 1634.013 |
| 14 | 1976.545 |
| 15 | 2085.076 |
| 16 | 2121.375 |
| 17 | 2534.712 |
| 18 | 2647.821 |
| 19 | 2538.172 |
| 20 | 2771.791 |
| 21 | 2880.745 |
| 22 | 3079.520 |
| 23 | 3115.498 |
| 24 | 3470.604 |
| 25 | 3485.627 |
| 26 | 3628.425 |
| 27 | 3790.439 |
| 28 | 3911.212 |
| 29 | 4131.821 |
| 30 | 4263.770 |
| 31 | 4371.162 |
| 32 | 4616.565 |
| 33 | 4875.288 |
| 34 | 4779.695 |
| 35 | 4826.668 |
| 36 | 5126.682 |
| 37 | 5235.894 |
| 38 | 5294.075 |
| 39 | 5477.265 |
| 40 | 5596.558 |
| 41 | 5707.574 |
| 42 | 5927.574 |
| 43 | 6099.666 |
| 44 | 6288.558 |
| 45 | 6381.209 |
| 46 | 6520.738 |
| 47 | 6754.712 |
| 48 | 6939.628 |
| 49 | 7139.314 |
| 50 | 7259.460 |

**Nota:** a média é uma medida não muito precisa, visto que a mesma é sensível a valores extremos, mesmo assim escolheu-se esta medida pois foram realizadas diversas iterações por registo, como cada registo contém apenas a média para aquele número de instalações em específico, e somado que este é apenas um algoritmo heurístico construtivo, achou-se que não haveria problemas, mesmo que a média seja sensível a outliers, e que a média estudada seja apenas a média amostral.

## 3.2 Algoritmo de Greedy

O algoritmo de greedy é um algoritmo bastante popular que tenta gerar uma solução com base nas informações disponíveis no momento, sem considerar as consequências dessas decisões no futuro, a lógica principal é escolher a melhor opção possível a cada iteração, retornando uma solução que nem sempre seja a solução ótima, mas é boa que chegue para muitos problemas.

Para realizar este algoritmo, primeiramente é preciso identificar o problema, que no caso deste projeto é problema de localização de instalações sem restrições de capacidade (UFLP), depois, é preciso identificar a escolha que o algoritmo faz para poder prosseguir, um exemplo de escolha é o caminho de menor custo do algoritmo de Dijkstra, que é um algoritmo de greedy, portanto, para este problema será o menor custo de transporte de um cliente a uma determinada instalação, por fim, basta repetir o algoritmo até o critério de paragem ser alcançado e ser retornada uma solução.

Como este algoritmo ignora as consequências das suas escolhas, pode levar a soluções distantes das ótimas, por isso que este algoritmo é um heurístico construtivo.

## 3.3 Algoritmo Swap | 2-opt

O Swap é um algoritmo heurístico de pesquisa local, que vai tentar encontrar pares de instalações e alterar os estados (aberto / fechado) das mesmas, tendo como objetivo explorar eficientemente o espaço de soluções para encontrar uma solução de menor custo

Como o Swap é um algoritmo heurístico de pesquisa local, ele vai tentar encontrar a melhor solução ótima local, e como precisa de uma solução inicial para poder começar, podemos parar o algoritmo em qualquer iteração que continuaremos com uma solução válida, não irá ser a melhor, mas poderá servir para o problema em questão.

Para o algoritmo implementado, a solução inicial é sempre calculada pelo algoritmo de greedy, visto que o mesmo oferece soluções admissíveis, e não recorre a variáveis aleatórias, tendo um comportamento mais previsível, o que irá ajudar futuramente na análise dos resultados.

## 3.4 Algoritmo Switch

O Switch é um algoritmo heurístico de pesquisa local bastante parecido com o algoritmo Swap, a diferença é que ao contrário do Swap, que troca os estados entre pares de instalações à procura de um resultado que será menor que o custo total, já o Switch vai alterar individualmente os estados das instalações, com o objetivo de aperfeiçoar a solução de forma granular e eficiente.

Como o Switch é um algoritmo heurístico de pesquisa local, ele partilha todas as vantagens de um algoritmo desse tipo, vantagens essas já descritas na apresentação do algoritmo Swap. E tal como o Swap, o Switch também precisa de uma solução inicial para poder trabalhar sobre, portanto, como no swap, escolheu-se o algoritmo de greedy para formular a solução inicial para este algoritmo, pelos mesmos motivos descritos anteriormente no algoritmo Swap.

## 3.5 Algoritmo de Pesquisa Tabu

O algoritmo de pesquisa Tabu ao contrário de todos os algoritmos apresentados anteriormente é um algoritmo meta heurístico, isto quer dizer que conseguem explorar melhor o espaço de soluções, ao contrário dos algoritmos heurísticos de pesquisa local, que são limitados nesse aspeto. Essa característica permite aos algoritmos meta heurísticos encontrarem soluções próximas das ótimas num curto espaço de tempo e com recursos computacionais limitados.

A pesquisa Tabu também usa um algoritmo de pesquisa local, que iterativamente vai saltar de potencial solução em potencial solução, mas para evitar os defeitos dos algoritmos de pesquisa local que podem ficar presos em áreas de resultados fracos, ou áreas onde os resultados estabilizaram, a pesquisa Tabu tem como recurso uma estrutura de dados em lista, onde são armazenados Tabus, que são locais por onde o algoritmo já passou, permitindo que explore melhor o espaço de pesquisa até o critério de paragem ser verificado. Para este algoritmo, o critério de paragem é atingido quando o mesmo não consegue encontrar na vizinhança um vizinho que não esteja na lista Tabu.

## 3.5 Algoritmo F&F

O algoritmo Filter and Fan (F&F) foi projetado para criar movimentos compostos com base em um design eficaz de pesquisa em árvore, permitindo a exploração eficiente de espaços de soluções complexas.

Neste algoritmo, um movimento composto é um movimento que pode ser dividido em uma sequência de movimentos ou submovimentos componentes mais elementares. Estes movimentos podem estar ligados (preservando a viabilidade a cada nível) ou desligados (não necessariamente preservando a viabilidade), por o F&F seguir esta estrutura, logo a estrutura de dados em árvore é a estrutura de dados que melhor se encaixa para fazer a pesquisa.

O método F&F concentra-se em um subconjunto de estratégias de componentes básicos e utiliza uma forma implícita de memória de pesquisa tabu incorporada em condições de legitimidade para simplificar a abordagem. O F&F pode utilizar múltiplas threads de pesquisa tabu a partir do nó raiz da árvore, incorporando componentes de memória de curto prazo. Como um algoritmo meta-heurístico, ele visa fornecer soluções ótimas ou quase ótimas em um curto espaço de tempo.

# 4. Análise do desempenho dos algoritmos implementados

## 4.1 Heurísticos Construtivos

### Abertura Aleatória de Instalações

Tabela 3 - Resultados obtidos do algoritmo heurístico construtivo de abertura aleatória de instalações

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ficheiro** | **N.º Instalações** | **N.º Clientes** | **S. Ótima** | **S. Obtida** | **%** | **TC** |
| Kcapmo1.txt | 100 | 100 | 1156.909 | 903.527 | -21.90% | 0.003 |
| Kcapmo2.txt | 100 | 100 | 1227.667 | 1423.195 | 15.93% | 0.002 |
| Kcapmo3.txt | 100 | 100 | 1286.369 | 1580.279 | 22.85% | 0.003 |
| Kcapmo4.txt | 100 | 100 | 1177.88 | 1554.963 | 32.01% | 0.004 |
| Kcapmo5.txt | 100 | 100 | 1147.595 | 1332.161 | 16.08% | 0.004 |
| Kcapmp1.txt | 200 | 200 | 2460.101 | 4778.144 | 94.23% | 0.012 |
| Kcapmp2.txt | 200 | 200 | 2419.325 | 5485.559 | 126.74% | 0.011 |
| Kcapmp3.txt | 200 | 200 | 2498.151 | 6714.604 | 168.78% | 0.01 |
| Kcapmp4.txt | 200 | 200 | 2633.561 | 6012.691 | 128.31% | 0.011 |
| Kcapmp5.txt | 200 | 200 | 2290.164 | 5408.293 | 136.15% | 0.011 |
| Kcapmq1.txt | 300 | 300 | 3591.273 | 12427.147 | 246.04% | 0.028 |
| Kcapmq2.txt | 300 | 300 | 3543.662 | 12201.287 | 244.31% | 0.023 |
| Kcapmq3.txt | 300 | 300 | 3476.806 | 12017.51 | 245.65% | 0.024 |
| Kcapmq4.txt | 300 | 300 | 3742.474 | 12906.951 | 244.88% | 0.024 |
| Kcapmq5.txt | 300 | 300 | 3751.326 | 12767.074 | 240.33% | 0.023 |
| Kcapmr1.txt | 500 | 500 | 2349.856 | 14523.582 | 518.06% | 0.067 |
| Kcapmr2.txt | 500 | 500 | 2344.757 | 12922.955 | 451.14% | 0.065 |
| Kcapmr3.txt | 500 | 500 | 2183.235 | 12250.847 | 461.13% | 0.065 |
| Kcapmr4.txt | 500 | 500 | 2433.11 | 15280.88 | 528.04% | 0.067 |
| Kcapmr5.txt | 500 | 500 | 2344.353 | 13129.103 | 460.03% | 0.068 |
| cap101.txt | 25 | 50 | 796648.44 | 15000 | -98.12% | 0.001 |
| cap102.txt | 25 | 50 | 854704.2 | 25000 | -97.08% | 0.0 |
| cap103.txt | 25 | 50 | 893782.11 | 35000 | -96.08% | 0.001 |
| cap104.txt | 25 | 50 | 928941.75 | 50000 | -94.62% | 0.0 |
| cap131.txt | 50 | 50 | 793439.56 | 30000 | -96.22% | 0.001 |
| cap132.txt | 50 | 50 | 851495.33 | 50000 | -94.13% | 0.001 |
| cap133.txt | 50 | 50 | 893076.71 | 70000 | -92.16% | 0.001 |
| cap134.txt | 50 | 50 | 928941.75 | 100000 | -89.24% | 0.001 |
| cap71.txt | 16 | 50 | 932615.75 | 7500 | -99.20% | 0.0 |
| cap72.txt | 16 | 50 | 977799.4 | 0 | -100.00% | 0.001 |
| cap73.txt | 16 | 50 | 1010641.5 | 17500 | -98.27% | 0.0 |
| cap74.txt | 16 | 50 | 1034977 | 25000 | -97.58% | 0.001 |
| capa.txt | 100 | 1000 | 17156454 | 14623893 | -14.76% | 0.004 |
| capb.txt | 100 | 1000 | 12979072 | 5682985 | -56.21% | 0.004 |
| capc.txt | 100 | 1000 | 11505594 | 4366781 | -62.05% | 0.003 |

### Greedy

Tabela 4 - Análise dos resultados obtidos do algoritmo construtivo de greedy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ficheiro** | **N.º Instalações** | **N.º Clientes** | **S. Ótima** | **S. Obtida** | **%** | **TC** |
| Kcapmo1.txt | 100 | 100 | 1156.909 | 1556,997 | 34,58% | 0.445 |
| Kcapmo2.txt | 100 | 100 | 1227.667 | 1746,635 | 42,27% | 0.004 |
| Kcapmo3.txt | 100 | 100 | 1286.369 | 1669,543 | 29,79% | 0.002 |
| Kcapmo4.txt | 100 | 100 | 1177.88 | 1609,027 | 36,60% | 0.003 |
| Kcapmo5.txt | 100 | 100 | 1147.595 | 1527,96 | 33,14% | 0.004 |
| Kcapmp1.txt | 200 | 200 | 2460.101 | 3382,471 | 37,49% | 0.013 |
| Kcapmp2.txt | 200 | 200 | 2419.325 | 3270,906 | 35,20% | 0.017 |
| Kcapmp3.txt | 200 | 200 | 2498.151 | 3175,683 | 27,12% | 0.013 |
| Kcapmp4.txt | 200 | 200 | 2633.561 | 3359,466 | 27,56% | 0.015 |
| Kcapmp5.txt | 200 | 200 | 2290.164 | 3062,346 | 33,72% | 0.014 |
| Kcapmq1.txt | 300 | 300 | 3591.273 | 4832,17 | 34,55% | 0.029 |
| Kcapmq2.txt | 300 | 300 | 3543.662 | 4723,5 | 33,29% | 0.032 |
| Kcapmq3.txt | 300 | 300 | 3476.806 | 4911,787 | 41,27% | 0.029 |
| Kcapmq4.txt | 300 | 300 | 3742.474 | 5161,44 | 37,92% | 0.03 |
| Kcapmq5.txt | 300 | 300 | 3751.326 | 5086,521 | 35,59% | 0.029 |
| Kcapmr1.txt | 500 | 500 | 2349.856 | 3763,061 | 60,14% | 0.105 |
| Kcapmr2.txt | 500 | 500 | 2344.757 | 3787,648 | 61,54% | 0.093 |
| Kcapmr3.txt | 500 | 500 | 2183.235 | 3724,08 | 70,58% | 0.102 |
| Kcapmr4.txt | 500 | 500 | 2433.11 | 3788,229 | 55,69% | 0.108 |
| Kcapmr5.txt | 500 | 500 | 2344.353 | 3784,452 | 61,43% | 0.111 |
| cap101.txt | 25 | 50 | 796648.44 | 816969,575 | 2,55% | 0.022 |
| cap102.txt | 25 | 50 | 854704.2 | 882917,95 | 3,30% | 0.025 |
| cap103.txt | 25 | 50 | 893782.11 | 939570,638 | 5,12% | 0.025 |
| cap104.txt | 25 | 50 | 928941.75 | 970361,838 | 4,46% | 0.029 |
| cap131.txt | 50 | 50 | 793439.56 | 821387,675 | 3,52% | 0.034 |
| cap132.txt | 50 | 50 | 851495.33 | 885483,75 | 3,99% | 0.018 |
| cap133.txt | 50 | 50 | 893076.71 | 921916,475 | 3,23% | 0.024 |
| cap134.txt | 50 | 50 | 928941.75 | 970361,838 | 4,46% | 0.019 |
| cap71.txt | 16 | 50 | 932615.75 | 965952,075 | 3,57% | 0.021 |
| cap72.txt | 16 | 50 | 977799.4 | 1011722,85 | 3,47% | 0.023 |
| cap73.txt | 16 | 50 | 1010641.5 | 1035149,625 | 2,43% | 0.021 |
| cap74.txt | 16 | 50 | 1034977 | 1057649,625 | 2,19% | 0.023 |
| capa.txt | 100 | 1000 | 17156454 | 39750861,7 | 131,70% | 0.055 |
| capb.txt | 100 | 1000 | 12979072 | 33454991,48 | 157,76% | 0.057 |
| capc.txt | 100 | 1000 | 11505594 | 36863151,06 | 220,39% | 0.107 |

**Interpretação dos resultados:**

**Conforme esperado, o algoritmo de abertura aleatória de instalações dá resultados muito instáveis, com taxas de desvio que variaram entre -100% e 500%**. Se o algoritmo for executado repetidamente, continuará a produzir resultados diferentes, podendo variar entre soluções distantes da ótima até à solução ótima em si. Essa inconsistência é natural para algoritmos que usam variáveis aleatórias**.**

**No entanto, o algoritmo greedy já não apresenta essa variação de resultados, visto que o algoritmo irá gerar os mesmos resultados sempre que o mesmo for executado, se, como é obvio, os problemas forem os mesmos, isto acontece devido à inexistência de variáveis aleatórias que desviam o algoritmo da imprevisibilidade**. **Em termos de resultados, na maioria dos casos, o algoritmo greedy conseguiu resultados mais próximos da solução ótima do que o algoritmo de abertura aleatória de instalações. No entanto, essa interpretação poderia ter sido completamente diferente caso o algoritmo de abertura aleatória de instalações tivesse sido executado noutro momento.**

**Quanto ao tempo de execução, o algoritmo de abertura aleatória de instalações como tem complexidade de tempo , acaba por ser mais rápida do que o algoritmo de greedy, que tem uma complexidade , devido à necessidade de um ciclo dentro de outro ciclo que vai percorrer todas as instalações para cada cliente.**

## 4.2 Heurísticos de Pesquisa Local

### Swap

Tabela 5 - Resultados obtidos do algoritmo de pesquisa local swap

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ficheiro** | **N.º Instalações** | **N.º Clientes** | **S. Ótima** | **S. Obtida** | **%** | **TC** |
| Kcapmo1.txt | 100 | 100 | 1156.909 | 1160,229 | 0,29% | 1.027 |
| Kcapmo2.txt | 100 | 100 | 1227.667 | 1227,667 | 0,00% | 0.075 |
| Kcapmo3.txt | 100 | 100 | 1286.369 | 1286,369 | 0,00% | 0.047 |
| Kcapmo4.txt | 100 | 100 | 1177.88 | 1198,678 | 1,77% | 0.076 |
| Kcapmo5.txt | 100 | 100 | 1147.595 | 1157,082 | 0,83% | 0.103 |
| Kcapmp1.txt | 200 | 200 | 2460.101 | 2460,101 | 0,00% | 1.269 |
| Kcapmp2.txt | 200 | 200 | 2419.325 | 2442,656 | 0,96% | 1.315 |
| Kcapmp3.txt | 200 | 200 | 2498.151 | 2526,639 | 1,14% | 0.628 |
| Kcapmp4.txt | 200 | 200 | 2633.561 | 2661,987 | 1,08% | 1.593 |
| Kcapmp5.txt | 200 | 200 | 2290.164 | 2290,164 | 0,00% | 2.892 |
| Kcapmq1.txt | 300 | 300 | 3591.273 | 3591,273 | 0,00% | 12.266 |
| Kcapmq2.txt | 300 | 300 | 3543.662 | 3564,405 | 0,59% | 5.139 |
| Kcapmq3.txt | 300 | 300 | 3476.806 | 3512,448 | 1,03% | 7.433 |
| Kcapmq4.txt | 300 | 300 | 3742.474 | 3742,474 | 0,00% | 25.036 |
| Kcapmq5.txt | 300 | 300 | 3751.326 | 3765,64 | 0,38% | 15.906 |
| Kcapmr1.txt | 500 | 500 | 2349.856 | 2349,856 | 0,00% | 100.421 |
| Kcapmr2.txt | 500 | 500 | 2344.757 | 2344,757 | 0,00% | 80.429 |
| Kcapmr3.txt | 500 | 500 | 2183.235 | 2195,486 | 0,56% | 78.954 |
| Kcapmr4.txt | 500 | 500 | 2433.11 | 2433,11 | 0,00% | 96.081 |
| Kcapmr5.txt | 500 | 500 | 2344.353 | 2344,353 | 0,00% | 56.649 |
| cap101.txt | 25 | 50 | 796648.44 | 797508,725 | 0,11% | 0.005 |
| cap102.txt | 25 | 50 | 854704.2 | 855466,85 | 0,09% | 0.002 |
| cap103.txt | 25 | 50 | 893782.11 | 894008,137 | 0,03% | 0.004 |
| cap104.txt | 25 | 50 | 928941.75 | 928941,75 | 0,00% | 0.003 |
| cap131.txt | 50 | 50 | 793439.56 | 793439,563 | 0,00% | 0.03 |
| cap132.txt | 50 | 50 | 851495.33 | 852257,975 | 0,09% | 0.015 |
| cap133.txt | 50 | 50 | 893076.71 | 893076,713 | 0,00% | 0.01 |
| cap134.txt | 50 | 50 | 928941.75 | 928941,75 | 0,00% | 0.01 |
| cap71.txt | 16 | 50 | 932615.75 | 932615,75 | 0,00% | 0.001 |
| cap72.txt | 16 | 50 | 977799.4 | 977799,4 | 0,00% | 0.001 |
| cap73.txt | 16 | 50 | 1010641.5 | 1012476,975 | 0,18% | 0.001 |
| cap74.txt | 16 | 50 | 1034977 | 1034976,975 | 0,00% | 0.001 |
| capa.txt | 100 | 1000 | 17156454 | 17781125,3 | 3,64% | 1.071 |
| capb.txt | 100 | 1000 | 12979072 | 13081049,25 | 0,79% | 1.479 |
| capc.txt | 100 | 1000 | 11505594 | 11505594,33 | 0,00% | 1.82 |

### Switch

Tabela 6 - Resultados obtidos do algoritmo de pesquisa local Switch

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ficheiro** | **N.º Instalações** | **N.º Clientes** | **S. Ótima** | **S. Obtida** | **%** | **TC** |
| Kcapmo1.txt | 100 | 100 | 1156.909 | 1173,864 | 1,47% | 0.839 |
| Kcapmo2.txt | 100 | 100 | 1227.667 | 1277,6 | 4,07% | 0.004 |
| Kcapmo3.txt | 100 | 100 | 1286.369 | 1286,369 | 0,00% | 0.004 |
| Kcapmo4.txt | 100 | 100 | 1177.88 | 1229,452 | 4,38% | 0.005 |
| Kcapmo5.txt | 100 | 100 | 1147.595 | 1161,167 | 1,18% | 0.004 |
| Kcapmp1.txt | 200 | 200 | 2460.101 | 2469,935 | 0,40% | 0.046 |
| Kcapmp2.txt | 200 | 200 | 2419.325 | 2439,564 | 0,84% | 0.071 |
| Kcapmp3.txt | 200 | 200 | 2498.151 | 2510,04 | 0,48% | 0.053 |
| Kcapmp4.txt | 200 | 200 | 2633.561 | 2693,316 | 2,27% | 0.028 |
| Kcapmp5.txt | 200 | 200 | 2290.164 | 2290,164 | 0,00% | 0.033 |
| Kcapmq1.txt | 300 | 300 | 3591.273 | 3613,191 | 0,61% | 0.067 |
| Kcapmq2.txt | 300 | 300 | 3543.662 | 3543,662 | 0,00% | 0.052 |
| Kcapmq3.txt | 300 | 300 | 3476.806 | 3498,521 | 0,62% | 0.081 |
| Kcapmq4.txt | 300 | 300 | 3742.474 | 3751,56 | 0,24% | 0.069 |
| Kcapmq5.txt | 300 | 300 | 3751.326 | 3751,326 | 0,00% | 0.063 |
| Kcapmr1.txt | 500 | 500 | 2349.856 | 2378,954 | 1,24% | 0.245 |
| Kcapmr2.txt | 500 | 500 | 2344.757 | 2384,981 | 1,72% | 0.231 |
| Kcapmr3.txt | 500 | 500 | 2183.235 | 2218,67 | 1,62% | 0.277 |
| Kcapmr4.txt | 500 | 500 | 2433.11 | 2457,078 | 0,99% | 0.231 |
| Kcapmr5.txt | 500 | 500 | 2344.353 | 2344,353 | 0,00% | 0.251 |
| cap101.txt | 25 | 50 | 796648.44 | 802191,275 | 0,70% | 0.001 |
| cap102.txt | 25 | 50 | 854704.2 | 855971,75 | 0,15% | 0.002 |
| cap103.txt | 25 | 50 | 893782.11 | 894008,137 | 0,03% | 0.003 |
| cap104.txt | 25 | 50 | 928941.75 | 934586,975 | 0,61% | 0.001 |
| cap131.txt | 50 | 50 | 793439.56 | 802797,588 | 1,18% | 0.003 |
| cap132.txt | 50 | 50 | 851495.33 | 855328,675 | 0,45% | 0.002 |
| cap133.txt | 50 | 50 | 893076.71 | 895642,512 | 0,29% | 0.002 |
| cap134.txt | 50 | 50 | 928941.75 | 934586,975 | 0,61% | 0.002 |
| cap71.txt | 16 | 50 | 932615.75 | 934199,137 | 0,17% | 0.001 |
| cap72.txt | 16 | 50 | 977799.4 | 977799,4 | 0,00% | 0.0 |
| cap73.txt | 16 | 50 | 1010641.5 | 1012476,975 | 0,18% | 0.001 |
| cap74.txt | 16 | 50 | 1034977 | 1034976,975 | 0,00% | 0.0 |
| capa.txt | 100 | 1000 | 17156454 | 17876593,32 | 4,20% | 0.058 |
| capb.txt | 100 | 1000 | 12979072 | 13070745,09 | 0,71% | 0.071 |
| capc.txt | 100 | 1000 | 11505594 | 11718205,39 | 1,85% | 0.079 |

**Interpretação dos resultados:**

A partir das tabelas com os resultados obtidos, é possível ver a partir da percentagem de desvio que o algoritmo swap conseguiu resultados muito mais próximos da solução ótima que o algoritmo switch, e em muitos casos conseguiu descobrir a solução ótima, no entanto, o algoritmo switch demora menos a executar que o swap, mas nada de muito significativo para as instâncias estudadas, no entanto, se se estivesse a executar instâncias de dimensões extremamente grandes, a diferença entre o tempo de execução entre os dois algoritmos podia ser muito maior.

## 4.3 Meta Heurísticos

### Pesquisa Tabu

Tabela 7 - Resultados obtidos do algoritmo meta heurístico Tabu Search

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ficheiro** | **N.º Instalações** | **N.º Clientes** | **S. Ótima** | **S. Obtida** | **%** | **TC** |
| Kcapmo1.txt | 100 | 100 | 1156.909 | 1173,864 | 1,47% | 4.239 |
| Kcapmo2.txt | 100 | 100 | 1227.667 | 1277,6 | 4,07% | 0.102 |
| Kcapmo3.txt | 100 | 100 | 1286.369 | 1286,369 | 0,00% | 0.099 |
| Kcapmo4.txt | 100 | 100 | 1177.88 | 1229,452 | 4,38% | 0.103 |
| Kcapmo5.txt | 100 | 100 | 1147.595 | 1161,167 | 1,18% | 0.096 |
| Kcapmp1.txt | 200 | 200 | 2460.101 | 2469,935 | 0,40% | 0.386 |
| Kcapmp2.txt | 200 | 200 | 2419.325 | 2439,564 | 0,84% | 0.39 |
| Kcapmp3.txt | 200 | 200 | 2498.151 | 2510,04 | 0,48% | 0.377 |
| Kcapmp4.txt | 200 | 200 | 2633.561 | 2693,316 | 2,27% | 0.357 |
| Kcapmp5.txt | 200 | 200 | 2290.164 | 2290,164 | 0,00% | 0.369 |
| Kcapmq1.txt | 300 | 300 | 3591.273 | 3613,191 | 0,61% | 1.015 |
| Kcapmq2.txt | 300 | 300 | 3543.662 | 3543,662 | 0,00% | 0.923 |
| Kcapmq3.txt | 300 | 300 | 3476.806 | 3498,521 | 0,62% | 0.976 |
| Kcapmq4.txt | 300 | 300 | 3742.474 | 3751,56 | 0,24% | 1.024 |
| Kcapmq5.txt | 300 | 300 | 3751.326 | 3751,326 | 0,00% | 0.993 |
| Kcapmr1.txt | 500 | 500 | 2349.856 | 2378,954 | 1,24% | 4.077 |
| Kcapmr2.txt | 500 | 500 | 2344.757 | 2384,981 | 1,72% | 3.711 |
| Kcapmr3.txt | 500 | 500 | 2183.235 | 2218,67 | 1,62% | 3.979 |
| Kcapmr4.txt | 500 | 500 | 2433.11 | 2457,078 | 0,99% | 3.789 |
| Kcapmr5.txt | 500 | 500 | 2344.353 | 2344,353 | 0,00% | 3.905 |
| cap101.txt | 25 | 50 | 796648.44 | 802191,275 | 0,70% | 0.009 |
| cap102.txt | 25 | 50 | 854704.2 | 855971,75 | 0,15% | 0.011 |
| cap103.txt | 25 | 50 | 893782.11 | 894008,137 | 0,03% | 0.009 |
| cap104.txt | 25 | 50 | 928941.75 | 934586,975 | 0,61% | 0.007 |
| cap131.txt | 50 | 50 | 793439.56 | 802797,588 | 1,18% | 0.025 |
| cap132.txt | 50 | 50 | 851495.33 | 855328,675 | 0,45% | 0.024 |
| cap133.txt | 50 | 50 | 893076.71 | 895642,512 | 0,29% | 0.024 |
| cap134.txt | 50 | 50 | 928941.75 | 934586,975 | 0,61% | 0.021 |
| cap71.txt | 16 | 50 | 932615.75 | 934199,137 | 0,17% | 0.005 |
| cap72.txt | 16 | 50 | 977799.4 | 977799,4 | 0,00% | 0.007 |
| cap73.txt | 16 | 50 | 1010641.5 | 1012476,975 | 0,18% | 0.006 |
| cap74.txt | 16 | 50 | 1034977 | 1034976,975 | 0,00% | 0.006 |
| capa.txt | 100 | 1000 | 17156454 | 17876593,32 | 4,20% | 0.493 |
| capb.txt | 100 | 1000 | 12979072 | 13070745,09 | 0,71% | 0.512 |
| capc.txt | 100 | 1000 | 11505594 | 11718205,39 | 1,85% | 0.57 |

### Filter and Fan

Tabela 8 - Resultados obtidos do algoritmo meta heurístico Filter and Fan

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ficheiro** | **N.º Instalações** | **N.º Clientes** | **S. Ótima** | **S. Obtida** | **%** | **TC** |
| Kcapmo1.txt | 100 | 100 | 1156.909 | 1556,997 | 34,58% | 1.742 |
| Kcapmo2.txt | 100 | 100 | 1227.667 | 1746,635 | 42,27% | 0.071 |
| Kcapmo3.txt | 100 | 100 | 1286.369 | 1669,543 | 29,79% | 0.069 |
| Kcapmo4.txt | 100 | 100 | 1177.88 | 1609,027 | 36,60% | 0.072 |
| Kcapmo5.txt | 100 | 100 | 1147.595 | 1527,96 | 33,14% | 0.074 |
| Kcapmp1.txt | 200 | 200 | 2460.101 | 3382,471 | 37,49% | 0.763 |
| Kcapmp2.txt | 200 | 200 | 2419.325 | 3270,906 | 35,20% | 0.736 |
| Kcapmp3.txt | 200 | 200 | 2498.151 | 3175,683 | 27,12% | 0.715 |
| Kcapmp4.txt | 200 | 200 | 2633.561 | 3359,466 | 27,56% | 0.719 |
| Kcapmp5.txt | 200 | 200 | 2290.164 | 3062,346 | 33,72% | 0.737 |
| Kcapmq1.txt | 300 | 300 | 3591.273 | 4832,17 | 34,55% | 3.014 |
| Kcapmq2.txt | 300 | 300 | 3543.662 | 4723,5 | 33,29% | 3.04 |
| Kcapmq3.txt | 300 | 300 | 3476.806 | 4911,787 | 41,27% | 3.103 |
| Kcapmq4.txt | 300 | 300 | 3742.474 | 5161,44 | 37,92% | 3.429 |
| Kcapmq5.txt | 300 | 300 | 3751.326 | 5086,521 | 35,59% | 3.031 |
| Kcapmr1.txt | 500 | 500 | 2349.856 | 3763,061 | 60,14% | 19.506 |
| Kcapmr2.txt | 500 | 500 | 2344.757 | 3787,648 | 61,54% | 22.065 |
| Kcapmr3.txt | 500 | 500 | 2183.235 | 3724,08 | 70,58% | 26.627 |
| Kcapmr4.txt | 500 | 500 | 2433.11 | 3788,229 | 55,69% | 19.552 |
| Kcapmr5.txt | 500 | 500 | 2344.353 | 3784,452 | 61,43% | 21.047 |
| cap101.txt | 25 | 50 | 796648.44 | 811014,3 | 1,80% | 0.004 |
| cap102.txt | 25 | 50 | 854704.2 | 860461,738 | 0,67% | 0.013 |
| cap103.txt | 25 | 50 | 893782.11 | 914381,95 | 2,30% | 0.006 |
| cap104.txt | 25 | 50 | 928941.75 | 967069,475 | 4,10% | 0.005 |
| cap131.txt | 50 | 50 | 793439.56 | 815432,4 | 2,77% | 0.011 |
| cap132.txt | 50 | 50 | 851495.33 | 877377,288 | 3,04% | 0.014 |
| cap133.txt | 50 | 50 | 893076.71 | 918624,112 | 2,86% | 0.013 |
| cap134.txt | 50 | 50 | 928941.75 | 967069,475 | 4,10% | 0.014 |
| cap71.txt | 16 | 50 | 932615.75 | 953309,812 | 2,22% | 0.002 |
| cap72.txt | 16 | 50 | 977799.4 | 990695,163 | 1,32% | 0.004 |
| cap73.txt | 16 | 50 | 1010641.5 | 1013856,45 | 0,32% | 0.005 |
| cap74.txt | 16 | 50 | 1034977 | 1049206,625 | 1,37% | 0.002 |
| capa.txt | 100 | 1000 | 17156454 | 35083092,99 | 104,49% | 1.266 |
| capb.txt | 100 | 1000 | 12979072 | 18446232,48 | 42,12% | 1.202 |
| capc.txt | 100 | 1000 | 11505594 | 15676531,89 | 36,25% | 1.3 |

**Interpretação dos resultados:**

A partir da análise das tabelas dos algoritmos meta heurísticos, é possível visualizar a partir da taxa de desvio que a pesquisa tabu conseguiu soluções muito mais próximas das soluções ótimas que o Filter and Fan, e em alguns deles conseguiu até encontrar a solução ótima, no entanto o Filter and Fan demora, na maior parte dos registos, menos tempo a executar que a pesquisa tabu, mas nada de muito significativo, o que é normal, visto que a pesquisa taby tem uma complexidade de O(n3) e o Filter and Fan tem uma complexidade de O(n2).

Mais uma vez, as diferenças para os tempos de execução não são significativas, mas o contexto poderia ser bastante diferente se se estivesse a utilizar instâncias com números extremamente elevados de instalações e clientes.

## 5. Complexidade dos algoritmos

Todos os algoritmos executam bastante rápido quando são executados com poucas instâncias, no entanto, consoante o volume dos dados vai aumentando a velocidade de execução dos mesmos tende a diminuir, podendo às vezes não nos dar a solução em tempo útil. Para avaliar o desempenho dos algoritmos, foi inventada uma escala denominada de Big O.

O Big O é uma notação matemática que descreve o comportamento limitante de uma função quando o argumento tende a um valor específico ou ao infinito. Ela pertence a uma família de notações inventadas por Paul Bachmann, Edmund Landau e outros, coletivamente chamadas de notação Bachmann–Landau ou de notação assintótica.

Nesta secção será feita uma análise dos algoritmos implementados, e será medida a complexidade de cada algoritmo segundo a notação Big O.

Algoritmo de Abertura Aleatória de Instalações:

Neste algoritmo, a parte do código que influencia significativamente para a complexidade do algoritmo, é a criação de um array booleano para as instalações, tendo já uma complexidade de , sendo o número de instalações, e um ciclo while que vai iterar até que sejam abertas o número de instalações pedidas, que acrescenta mais uma complexidade de . Como abrir uma instalação é feita a partir de uma variável aleatória, pode ser que precise de mais iterações do que o número que se pediu para abrir as instalações, pois pode ser sorteado duas vezes seguidas o mesmo número, e para resolver isso o algoritmo vai precisar iterar mais vezes do que era suposto.

Mesmo com a inconveniência mostrada anteriormente, a complexidade do algoritmo vai ser do tipo , sendo o número de instalações, e o número de instalações a serem abertas.

Algoritmo de Greedy:

Como para este algoritmo também é preciso criar um array booleano para as instalações, temos logo uma complexidade de no começo do algoritmo, sendo o número de instalações. Como este algoritmo tem um ciclo dentro de um ciclo, sendo o primeiro ciclo para iterar pelos clientes, e o segundo para iterar por todas instalações por cliente, logo vai-se ter uma complexidade de

Ao juntar tudo vai-se ficar com uma complexidade de , sendo o número de instalações, e n o número de clientes. Esta complexidade pode ser arredondada para

Algoritmo Swap | 2-opt:

Visto que os algoritmos de pesquisa local são mais extensos que os anteriores, então a análise vai ser efetuada parte a parte.

Para a função *calculate\_cost*, que vai calcular o custo total conforme a solução que lhe passamos, vai ter uma complexidade de , visto que contém um ciclo dentro de um ciclo, onde o primeiro itera por todos os clientes, e o segundo vai iterar por todas as instalações por cada cliente.

Para a função *swap\_heuristic\_local\_search* que vai executar o algoritmo Swap, as funções que impactam significativamente para a complexidade são as seguintes:

* **Inicialização e Cópia da Solução Inicial:**
* **Cálculo do Custo Inicial:**
* **Ciclo *while improved*:** Depende do número de iterações até que não haja melhoria, portanto vai-se assumir que é o número máximo de iterações.
  + **Ciclo dentro de ciclo para as instalações:** para todas as combinações de instalações
  + **Cópia da solução:**
  + **Cálculo do custo da vizinhança:**

A cada iteração do ciclo while a complexidade vai ser de , portanto a complexidade final desta função vai ser .

Para a função *swap\_heuristic\_uflp* que serve para preparar os dados, e retornar a solução final, vai ter as seguintes complexidades:

* **Conversão dos custos das instalações:**
* **Solução inicial com o algoritmo de greedy:**
* **Local Search:**

Com a análise, pode-se afirmar que a complexidade total da função é de , onde é o número de instalações, o número de clientes, e o número máximo de iterações do ciclo while.

Algoritmo Switch:

Neste algoritmo também existe uma função *calculate\_cost*, portanto a complexidade vai ser a mesma do algoritmo Swap: .

O algoritmo swap vai trocar o estado de pares de instalações simultaneamente, já o algoritmo switch vai alterar o estado de cada instalação individualmente e avalia o impacto no custo total, como o comportamento dos dois algoritmos é bastante similar, e como as diferenças que existem entre os dois algoritmos exibem a mesma complexidade, então a complexidade do algoritmo Switch será a mesma do algoritmo Swap: , onde é o número de instalações, o número de clientes, e o número máximo de iterações do ciclo while.

Pesquisa Tabu:

Tal como os algoritmos de pesquisa local, a pesquisa tabu também utiliza um método para calcular custos, e o algoritmo de greedy para obter uma solução inicial, logo já vai ter complexidades de e .

A principal diferença da pesquisa tabu para um algoritmo de pesquisa local é a lista tabu, que vai guardar os locais por onde o algoritmo já passou. Para verificar se algum local está presente na lista tabu, foi criado um método chamado *is\_solution\_in\_tabu\_list*, esse método apresenta uma complexidade de , onde é o tamanho da lista tabu, e é o número de instalações.

Já na função *tabu\_search\_core*, a função responsável por executar o algoritmo de pesquisa tabu, apresenta as seguintes complexidades:

* **Inicialização e cópia da solução inicial:**
* **Cálculo do custo inicial:**
* **Ciclo:** , onde é o número máximo de iterações
  + **Gerar vizinhança para cada instalação:**
  + **Verificação da lista tabu:**
  + **Cálculo do custo do vizinho:**
  + **Ordenação da vizinhança:**
  + **Atualização da solução corrente e a lita tabu:**

Ao juntar todas as complexidades, vai-se obter uma complexidade de .

Filter & Fan:

Tal como o algoritmo de pesquisa tabu, e os algoritmos de pesquisa local, o F&F utiliza também o método *calculate\_cost* e uma solução inicial dada pelo algoritmo de greedy, logo vai começar já com complexidade de: e .

O F&F para poder operar, para além de precisar da função *calculate\_cost*, precisa também das funções: *local\_search* e *generate\_candidate\_solutions*, que contém as seguintes complexidades:

**Generate\_candidate\_solution:**

* **Cópia da solução atual:**
* **Ciclo para mudar 20% das instalações:**

Como complexidade total este algoritmo tem

**Local\_search:**

* **Cópia da solução inicial:**
* **Cálculo do custo inicial:**
* **Ciclo principal:**
  + **Ciclo externo das instalações:**
  + **Ciclo interno das instalações:**
  + **Cópia e modificação da solução:**
  + **Cálculo do custo do vizinho:**

Ao juntar as complexidades do ciclo principal vai-se ficar com uma complexidade de:

Por fim, a função *filter\_and\_fan*, apresenta as seguintes complexidades:

* **Chamada inicial do *local\_search:***onde é o número de iterações
* **Ciclo das iterações:** , onde é o número máximo de iterações definido como parâmetro
  + **Geração de candidatos:** ,onde é o número máximo de candidatos definido como parâmetro
    - **Pesquisa local para cada candidato:**
* **Seleção do melhor candidato:**

Ao juntar as complexidades, vai resultar numa complexidade de

# 6. Características do projeto

## 6.2 Linguagem Escolhida

Como o principal objetivo deste trabalho é implementar algoritmos heurísticos já conhecidos e observar os resultados obtidos, faz com que o python seja a melhor linguagem para se usar para o desenvolvimento dos mesmos, visto que ele é muito fácil de se usar e sobretudo de se ler, contém uma gama enorme de bibliotecas para se usar, e muito rapidamente se desenvolve algoritmos para o mesmo, levando a uma maior flexibilidade, que por sua vez motiva a experimentação de novas abordagens.

## 6.1 Aceleração da execução dos algoritmos

O python por ser uma linguagem interpretada linha a linha em vez se ser compilada leva a surpreendentes impactos negativos na performance da aplicação, sobretudo quando se quer executar algoritmos complexos, que processam muita informação, e contém muitos ciclos dentro de ciclos, infelizmente quase todos os algoritmos desenvolvidos cumprem esses três pontos que fazem com que o python demore mais do que o tempo que não se tem para poder executar os algoritmos. Para resolver esse problema decidiu-se colocar um compilador no python, inicialmente tentou-se usar o compilador *pypy*, no entanto como este trabalha muito mal com bibliotecas externas como o *numpy*, que também acelera a execução de algoritmos, então descartou-se a ideia da utilização do mesmo e foi decido usar a função *njit* da biblioteca *numba*, que também faz a compilação do código, e é muito mais simples de se usar, basta inserir a tag *@njit* por cima da função que se quer compilar. O *njit* também tem algumas limitações com algumas funções, mas como como são poucas, conseguiu-se sempre uma alternativa para as mesmas.

Para acelerar mais, nos algoritmos meta heurísticos usou-se a tag *@njit(paralel = True)* que vai fazer com que os ciclos dentro da função sejam executados em paralelo a partir de threads, acelerando ainda mais o processo de pesquisa pela solução.

# 7. Anexos

Funções Comuns:

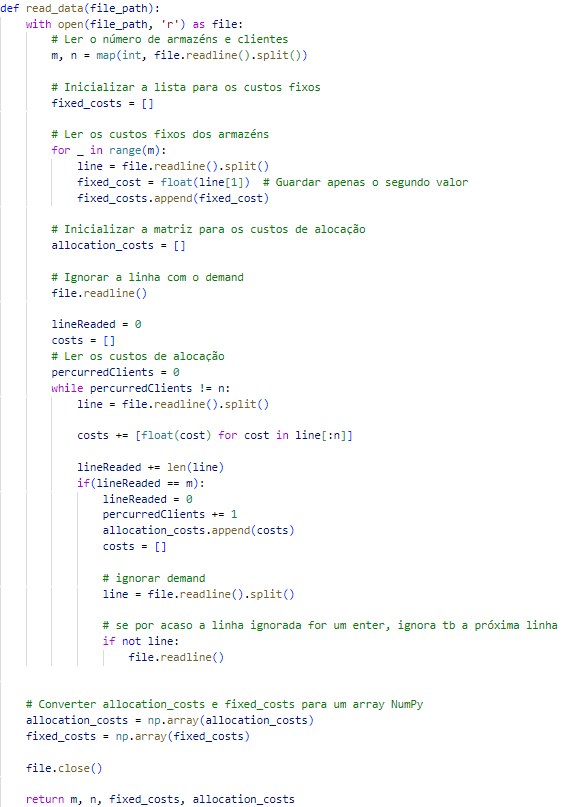


Figura 1 - Função que lê os dados das instâncias em txt

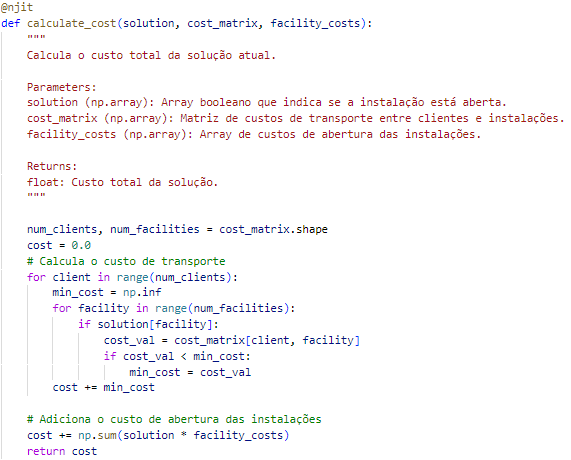


Figura 2 - Função que calcula o custo total de uma solução

Abertura aleatória de instalações:

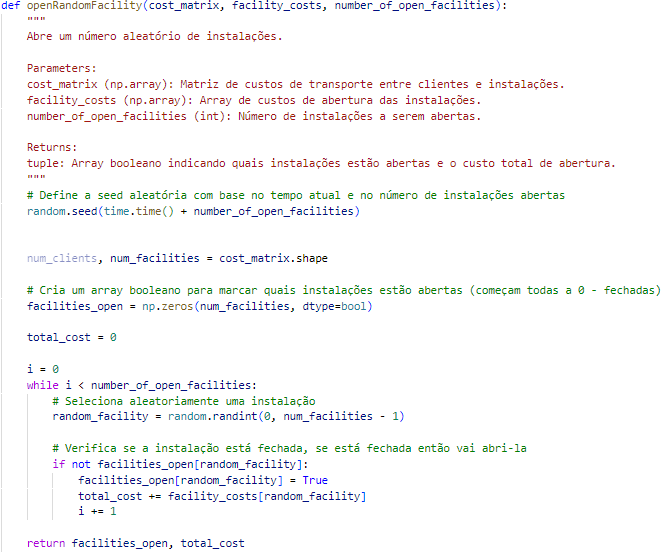


Figura 3 – Algoritmo heurístico construtivo para abrir instalações aleatórias

Algoritmo de Greedy:

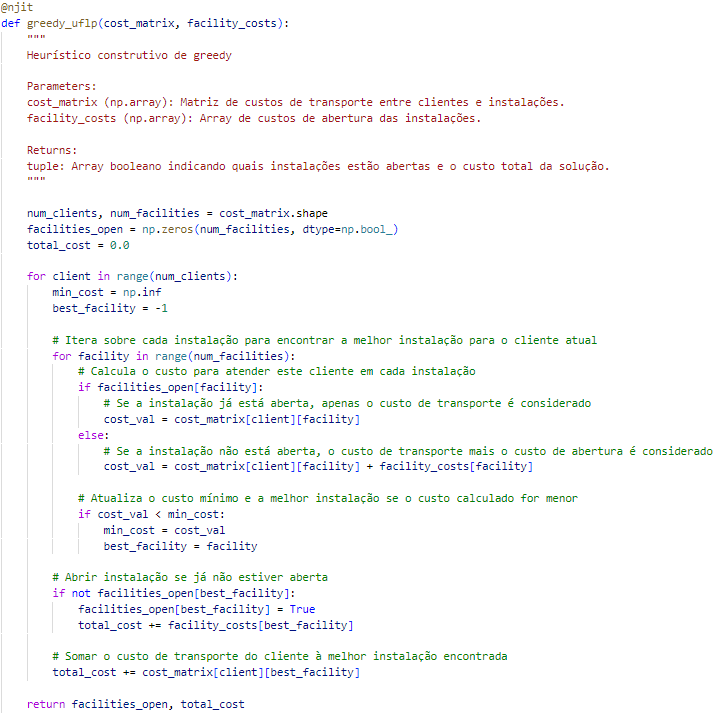


Figura 4 - Algoritmo heurístico construtivo de greedy

Pesquisa Local Swap:

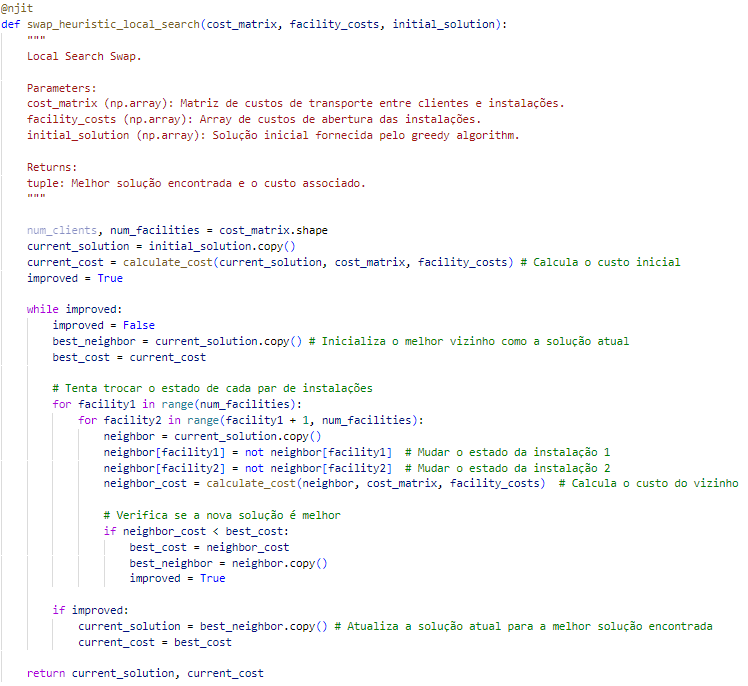


Figura 5 - Algoritmo de pesquisa local Swap

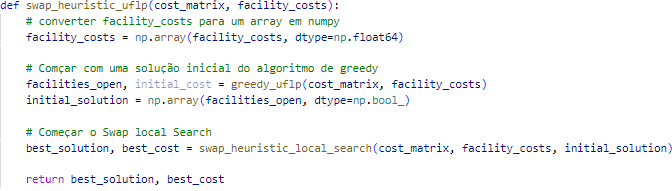


Figura 6 - Função que gera uma solução inicial e chama o Swap

Pesquisa Local Switch:

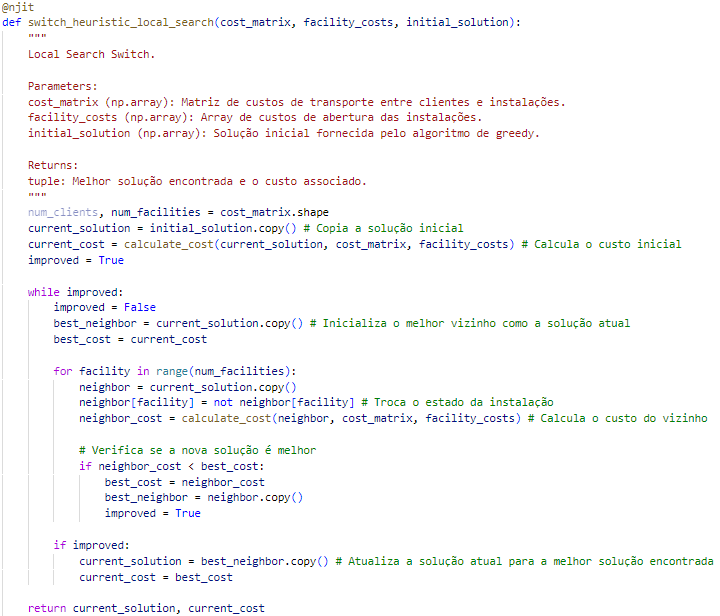


Figura 7 - Algoritmo de pesquisa local Switch

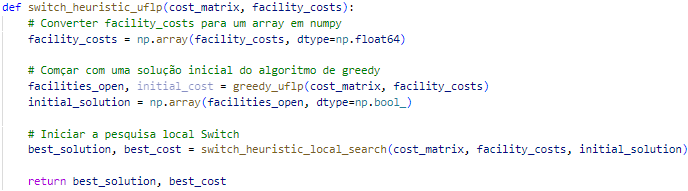


Figura 8 - Função que constrói uma solução inicial e chama o Switch

Pesquisa Tabu:

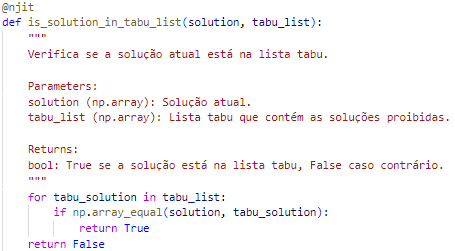


Figura 9 - função que verifica se uma determinada função está na lista tabu

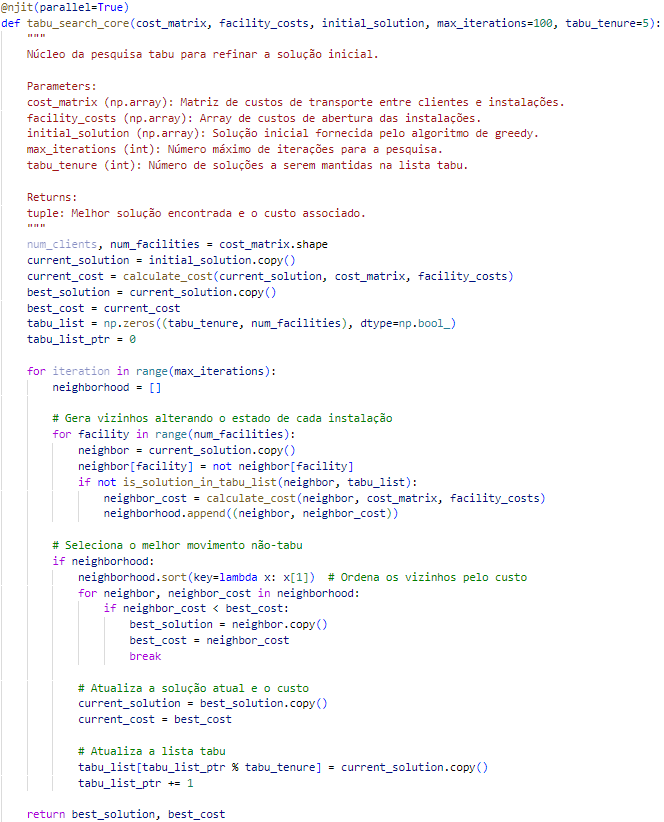


Figura 10 - Algoritmo meta heurístico de pesquisa tabu

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 11 - Função que gera a solução inicial e chama o algoritmo de pesquisa tabu

Filter & Fan:

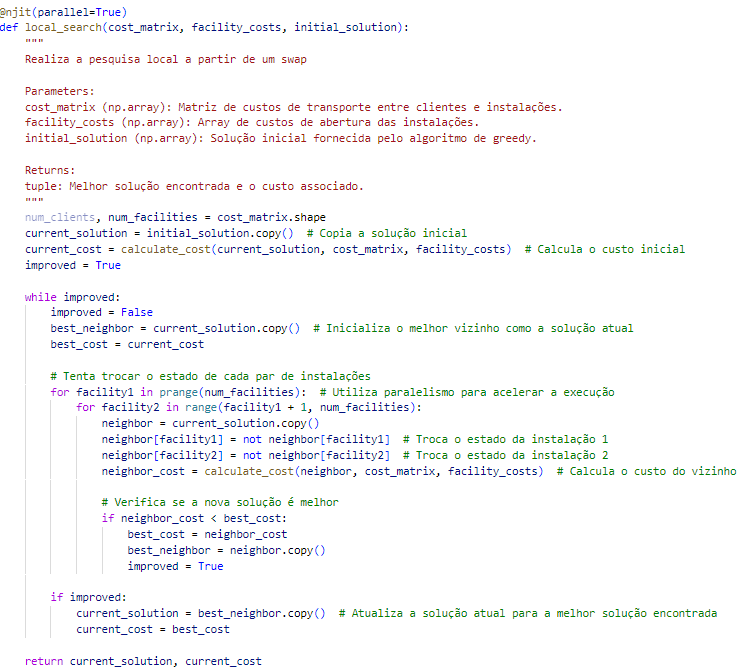


Figura 12 - Algoritmo de pesquisa local Swap usado pelo algoritmo meta heurístico Filter & Fan

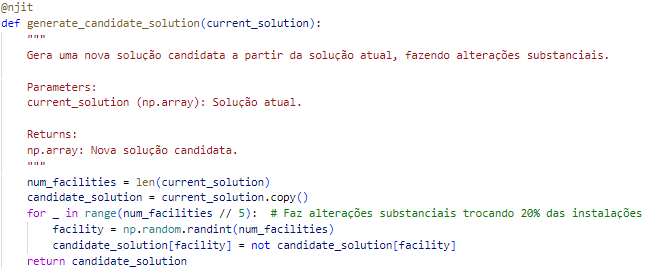


Figura 13 - Função que gera uma lista de candidatos segundo uma solução inicial

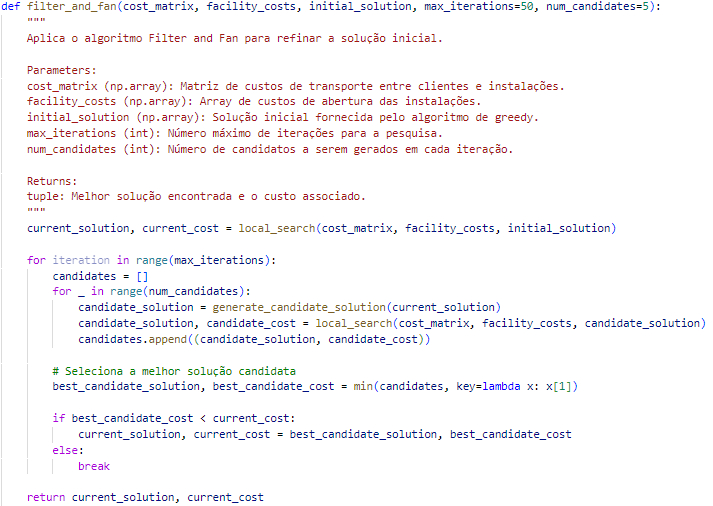


Figura 14 - Algoritmo Filter & Fan

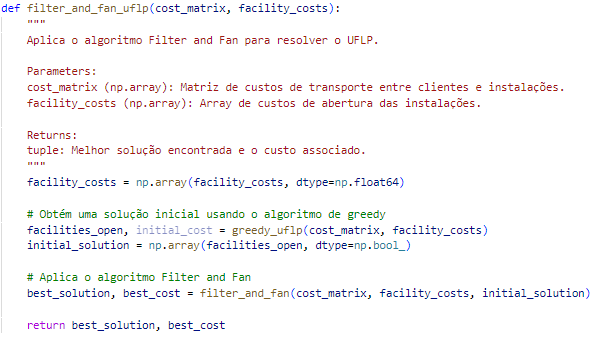


Figura 15 - Função que gera uma solução inicial e chama o algoritmo meta heurístico F&F

# 8. Conclusões e Trabalho Futuro

Neste trabalho foi abordado o problema de localização de instalações sem restrições de capacidade, que é uma variante do problema de localização de instalações classificada como NP-Difícil. Considerando a complexidade deste tema, foram implementadas e estudadas diversas abordagens heurísticas famosas de modo a conseguir alternativas aos métodos exatos, que podem não conseguir as soluções em tempo útil, e necessitam de diversos recursos computacionais quando lidam com instâncias de grande escala.

Olhando em retrospetiva podemos afirmar que não só atendemos a todos os objetivos propostos inicialmente, como conseguimos acrescentar mais aquilo que nos foi pedido, sendo o principal: a implementação de algoritmos meta heurísticas.

Para além de algumas questões que deveriam ter sido melhoradas, este trabalho ainda é suscetível a trabalho futuro, como: desenvolvimento de novas heurísticas e meta heurísticas que possam oferecer melhor desempenho; ampliação do conjunto de testes, assim não se fica tão preso a uma amostra tão pequena, podendo fazer-se uma análise mais precisa do comportamento dos algoritmos; e a análise de outros aspetos de desempenho, como a adaptabilidade dos algoritmos a diferentes contextos práticos.

Concluindo, este trabalho contribuiu significativamente para a nossa compreensão sobre abordagens heurísticas para a resolução de problemas, a análise de algoritmos, e sobre a área de conhecimento de algoritmos para a localização de instalações sem restrições de capacidade, podendo auxiliar no desenvolvimento de trabalhos futuros.

# Referências Bibliográficas

D. G. (2001). *Neighborhood search heuristics for the uncapacitated facility location problem.*

J. d., A. A., J. M., & J. P. (2016). *Solving the deterministic and stochastic uncapacitated.*

L. M., & P. V. (2004). *A simple tabu search for warehouse location.*

P. G., & C. R. (2006). *A simple filter-and-fan approach to the facility location problem.*

# Referências WWW

[01] **https://www.geeksforgeeks.org/greedy-algorithms/**

Página do geeksforgeeks que explica a lógica do algoritmo de greedy

[02] **https://en.wikipedia.org/wiki/Local\_search\_(optimization)**

Página da wikipédia que explica o que são algoritmos de otimização em pesquisa local

[03] **https://www.geeksforgeeks.org/what-is-tabu-search/**

Página do geekforgeeks que explica a lógica do algoritmo Tabu Search

[04] **https://en.wikipedia.org/wiki/Tabu\_search**

Página da Wikipédia que explica a lógica do algoritmo Tabu Search

[04] **https://en.wikipedia.org/wiki/Big\_O\_notation**

Página da Wikipédia que explica o que é a notação Big O

[05] **https://en.wikipedia.org/wiki/Local\_search\_(optimization)**

Página da Wikipédia que explica o que é otimização em pesquisa local

[06] **https://en.wikipedia.org/wiki/2-opt**

Página da Wikipédia que explica o que é o Swao | 2-opt