Mestrado de Engenharia de Telecomunicações e Informática

Resolução de Problemas - Algoritmos de procura

Otimização de Redes de Telecomunicações Utilizando Algoritmos de Procura

Grupo 001:
Professora: Dalila Alves Durães

Avaliação pelo grupo

DELTAs referentes a participação de cada elemento do grupo:

Alunos	DELTA
João Pedro Costa Bastos	-0.2
Fernando João Santos Mendes	+0.1
Bruno Miguel Fernandes Araújo	+0.1

Informação referente aos membros do grupo:

Nome	Nº mecanográfico	Email institucional
João Pedro Costa Bastos	PG57564	pg57564@uminho.pt
Fernando João Santos Mendes	PG55807	pg55807@uminho.pt
Bruno Miguel Fernandes Araújo	PG55806	pg55806@uminho.pt

Índice

Introdução

O presente relatório documenta o desenvolvimento de um projeto integrado na unidade curricular de Inteligência Artificial para as Telecomunicações, do Mestrado Integrado em Telecomunicações e Informática da Universidade do Minho, no ano letivo de 2024/2025. Este projeto tem como principal objetivo explorar e implementar algoritmos de procura aplicados à otimização de redes de telecomunicações, com foco na melhoria da qualidade de serviço (QoS) e na eficiência energética.

No contexto apresentado, o problema aborda a alocação de utilizadores às estações base de uma rede de telecomunicações urbana. A complexidade reside em equilibrar requisitos como a distância entre utilizadores e antenas, capacidade de transmissão, e requerimentos específicos de largura de banda, garantindo ao mesmo tempo o menor impacto ambiental possível.

As atividades desenvolvidas neste projeto incluíram a formulação do problema como um modelo de procura, a concepção e implementação de diferentes algoritmos (informados e não informados), e a análise dos resultados obtidos através de simulações. Esta abordagem permitiu comparar a eficiência, o custo e o desempenho dos algoritmos na solução do problema em questão.

Este relatório detalha todo o processo de desenvolvimento, desde a descrição inicial do problema até a interpretação dos resultados e discussões finais, fornecendo uma visão abrangente sobre os desafios e as soluções propostas.

Descrição do Problema

A otimização de redes de telecomunicações é essencial para garantir a qualidade de serviço (QoS) aos utilizadores, especialmente em cenários urbanos onde a densidade populacional é alta e os recursos da rede são limitados. A empresa Conexia Telecom tem como objetivo melhorar a eficiência da sua rede de comunicações numa cidade, composta por várias antenas (estações base) distribuídas geograficamente.

Limitações específicas:

- Raio de cobertura: os utilizadores só podem ser alocados a uma antena dentro de uma distância máxima.
- Capacidade de transmissão: cada estação base tem um limite de capacidade, além do qual a qualidade de serviço começa a degradar.
- Demanda de largura de banda: atividades como videochamadas, streaming e navegação, têm requisitos específicos de largura de banda.

Objetivos ao alocar clientes:

1. Minimizar a distância entre os utilizadores e as estações base.
2. Evitar sobrecarga nas antenas, penalizando alocações que excedam 80% da capacidade de transmissão.
3. Garantir a largura de banda necessária para cada tipo de atividade, sem exceder a capacidade disponível em cada estação base.
4. Reduzir o custo energético da rede, promovendo um uso eficiente dos recursos.

Parâmetros do problema de procura:

- O estado inicial é a configuração inicial dos utilizadores e antenas.
- O estado objetivo é a configuração ótima que atende às restrições de capacidade e largura de banda, minimizando custos.
- As ações correspondem a realocações de utilizadores entre antenas, ajustando o uso da rede.
- O custo de uma solução está relacionado ao consumo energético e à qualidade de serviço alcançada.

Metodologia

Formulação do Problema

Para modelar o problema como um problema de procura, foi necessário definir:

- **Estado inicial**:
  A configuração inicial da rede, onde os utilizadores são alocados de maneira aleatória às antenas disponíveis, sem otimização.
- **Estado objetivo**:
  A configuração ideal da rede, na qual a distância entre utilizadores e antenas é minimizada, a utilização de largura de banda está dentro dos limites permitidos e a capacidade de transmissão das antenas não é excedida.
- **Ações (Operadores)**:
  A alocação ou realocação de utilizadores para diferentes antenas, considerando as restrições de distância, largura de banda e capacidade.
- **Função de custo**:
  O custo da solução foi definido como uma combinação ponderada de:
  - Consumo energético da rede.
  - Degradação da qualidade de serviço devido a sobrecarga.
  - Distância total entre utilizadores e antenas.

Heurística Utilizada

Para os algoritmos informados, foi desenvolvida uma heurística que se caracteriza pela soma das larguras de banda utilizada das ligações (arestas) até à antena destino.

Desenvolvimento de Algoritmos

Foram implementados diferentes algoritmos de procura em Python, divididos em duas categorias principais:

Algoritmos de procura não informada:

- **Busca em largura**: Explora todas as possibilidades de alocação em ordem de profundidade crescente.
- **Busca em profundidade**: Analisa as configurações possíveis de forma mais profunda, priorizando caminhos específicos antes de retroceder.

Algoritmos de procura informada:

- Utiliza uma heurística que considera a distância entre utilizadores e antenas, bem como a capacidade das antenas e os requisitos de largura de banda.
- **Greedy**: Baseia-se exclusivamente na heurística, priorizando o ganho imediato sem considerar o custo acumulado.

Ferramentas Utilizadas

O projeto foi inteiramente desenvolvido em Python, utilizando as seguintes bibliotecas:

- **Graph:** Para manipulação de dados e cálculos numéricos.
- **File:** Para utilização das funções “write” e “parse”.
- **matplotlib:**Representar graficamente os resultados das simulações
- **numpy:**
- **heapq:** Facilitar a implementação de algoritmos de procura informada, como A*
- **Math:** Para cálculo de distância euclidianas

Modelagem dos Objetos

Antena

Variáveis	Tipos de variáveis
id	Integer
name	String
raio_cobertura	Integer
largura_banda_max	Integer
largura_banda_utilizada	Integer
lista_antenas_vizinhas	List(String)
lista_clientes	List(object Cliente)
lista_clientes_chamada	List(object Cliente)
lista_clientes_streaming	List(object Cliente)
lista_clientes_jogos	List(object Cliente)
qos	Float
x	Float
y	Float

Tabela 1 - Referente às variáveis do objeto “Antena”

Cliente

Variáveis	Tipos de variáveis
id_antena	Integer
x	Float
y	Float

servico/atividade	String

Tabela 2 - Referente às variáveis do objeto “Cliente”

Graph

Variáveis	Tipos de variáveis
antenas	Key: Name Value: Object Antena
clientes	Key: id Cliente Value: Object Cliente
m_graph	Key: Name Antena Value: List((Name,Integer))
m_h	Key: Name Antena Value: List(Integer)

Tabela 3 - Referente às variáveis do objeto “Graph”

Funcionalidades

Funcionalidades gerais:

- Gerir clientes
- Gerir antenas
- Imprimir grafo
- Desenhar mapa
- Imprimir antenas de grafo
- Imprimir arestas de grafo
- DFS
- BFS
- A*
- Gulosa
- Gravar o estado da cidade

```
def menu():
    print("\nMenu Principal")
    print("1-Gerir Clientes")
    print("2-Gerir Antenas")
    print("3-Imprimir Grafo")
    print("4-Desenhar Mapa")
    print("5-Imprimir  antenas de Grafo")
    print("6-Imprimir arestas de Grafo")
    print("7-DFS")
    print("8-BFS")
    print("9-A*")
    print("10-Gulosa")
    print("11-Gravar o estado da cidade.")
    print("0-Saír")
```

Imagem 1 - Menu principal

As funcionalidades acima descritas descrevem os vários pontos do menu que o utilizador tem acesso ao correr o programa (como podemos conferir na imagem 1.

Funcionalidades específicas de "Cliente":

- Criar Cliente
- Listar Cliente
- Consultar Cliente
- Remover Cliente

```
def menu_clientes():
    print("\nMenu de Clientes")
    print("1. Criar Cliente")
    print("2. Listar Clientes")
    print("3. Consultar Cliente")
    print("4. Remover Cliente")
    print("5. Voltar ao Menu Principal")
```

Imagem 2 - Menu de clientes

Através do menu dos clientes é possível recorrer a um conjunto de opções que nos permitem tornar o projeto mais perto de um cenário real. Tal é possível através da opção:

1. **Criar cliente**: Esta opção permite criar um cliente caracterizado pela sua localização num plano bidimensional (x,y) e pelo serviço em uso. Os serviços simulados são serviços de chamada que ocupam uma largura de banda de 1 MBps, serviços de jogos que ocupam uma largura de banda de 2MBps e serviços de streaming que ocupam uma largura de banda de 4 MBps.
   Em prole de um projeto dinâmico e autónomo, foi desenvolvida uma função que visa associar dinamicamente o cliente a uma antena. A associação rege-se sob 2 condições sendo a primeira a distância euclidiana entre 2 pontos. A lista que contém as antenas é percorrida no âmbito de se calcular a distância euclidiana entre o cliente e cada uma das antenas iteradas, sendo que, sobre a condição de que a distância calculada deverá ser menor que o raio de cobertura da antena comparada, são então guardadas todas aquelas que corresponderam positivamente à condição estabelecida. Para que houvesse uma penalização nas antenas com 20% da qualidade de serviço (ou inferior), todas as antenas cujo QoS final (valor representativo do cálculo do QoS da antena com o cliente adicionado) fosse superior a 0% e inferior a 20% eram guardadas numa lista diferente das opções de antenas cujas condições da distância e QoS final (superior a 20%) fossem respeitadas e guardadas numa lista de antenas viáveis cuja seria usada no seguimento do código. Foi tido em atenção que em caso de não haver antenas que respeitassem as condições das antenas viáveis, que a opção com melhor QoS final da lista das penalizadas fosse guardada na lista das viáveis, desta forma foi garantido que um cliente estabelecesse ligação de acordo com as prioridades definidas (tal podemos observar na Imagem 3).

```
cliente = self.clientes[cliente_id]
antenas_viaveis = []
antenas_penalizadas= []

for antena in self.antenas.values():
    distancia = self.calcular_distancia_euclidiana(cliente.get_x(),cliente.get_y(),antena.get_x(),antena.get_y())
    if distancia <= antena.get_raio_cobertura() :
        qos= self.calcular_qos(antena,largura_banda_necessaria)
        if antena.get_qos()>=0 and qos>=0 and qos<20:
            antenas_penalizadas.append((antena,distancia,qos))
        elif antena.get_qos()>=20 and qos>=20:
            antenas_viaveis.append((antena, distancia))

if len(antenas_viaveis) == 0:
    if len(antenas_penalizadas) != 0:
        melhor_antena = None
        melhor_qos = -1
        for antena, distancia, qos in antenas_penalizadas:
            if qos > melhor_qos:
                # Atualiza a melhor antena e seu QoS
                melhor_antena = antena
                melhor_distancia = distancia
                melhor_qos = qos

        # Adiciona a melhor antena à lista de antenas viáveis
        antenas_viaveis.append((melhor_antena, melhor_distancia))
    else:
        return print("Nenhuma antena disponível!")
```

Imagem 3 - Penalização para antenas com QoS inferior a 20%

Após conferir qual a ligação com menor distância entra em vigor a 2 condição para a associação, que se relaciona com a largura de banda disponível da antena face à largura de banda requisitada pelo serviço do cliente. Da mesma forma que no cálculo da distância as antenas são iteradas e são registradas aquelas que ,além de garantirem um raio de cobertura, tenham uma largura de banda suficiente para o cliente se associar. O cliente deverá então ser alocado na antena que obtiver uma distância menor e numa antena que tenha a melhor capacidade, ou seja, maior largura de banda disponível, para que obtenha um maior conforto ao estabelecer o seu serviço. Em caso de não conseguir encontrar nenhuma antena disponível o cliente é descartado. Ao associar o cliente a uma dada antena todas as características das mesmas são atualizadas, como é o caso da largura de banda utilizada cujo valor acresce consoante o calor do serviço do cliente. O QoS (Quality of service) é atualizado segundo a função representada na imagem 3, que traduz uma proporção direta entre a largura de banda utilizada e o decréscimo da qualidade de serviço da antena. É também atualizado o nome da antena no cliente à qual ficou associado.

```
def calcular_qos(self,melhor_antena, largura_banda_necessaria):
    qos_antena = melhor_antena.get_qos()
    diferenca_qos = (largura_banda_necessaria * 100)/melhor_antena.get_largura_banda_max()
    return qos_antena - diferenca_qos
```

Imagem 4 - Cálculo do Quality of Service (QoS)

Considerando que as arestas detém como métrica (peso) a soma das larguras de banda utilizadas das duas antenas(vértices) e os clientes condicionam as mesmas, visto que alteram a largura de banda das antenas às quais são alocados, além da procura e associação na melhor antena é feita a atualização dos valores (pesos) das arestas que estão diretamente ligadas à antena que sofreu alterações. Em prole do mesmo, foi desenvolvida a função“atualizar_edges” que procura identificar todas as antenas vizinhas

da antena receptora do cliente e atualizar o peso das suas ligações redefinindo o mesmo na variável "m.graph". Visto que as ligações são bidirecionais, além de atualizar as arestas com os vizinhos da antena receptora é necessário atualizar o mesmo valor com os vértices invertidos, ou seja, sendo a ligação com origem nas antenas vizinhas para a antena receptora (Imagem 4)

```python
def atualizar_edges(self, cliente, largura_banda_nova, flag=True):
    nome_antena = cliente.get_name_antena()
    arestas_para_atualizar = []

    for adjacente, peso in self.m_graph[nome_antena]:
        if flag:
            novo_peso = peso + largura_banda_nova
        else:
            novo_peso = peso - largura_banda_nova

        arestas_para_atualizar.append((adjacente, novo_peso))

    self.m_graph[nome_antena] = arestas_para_atualizar

    for adjacente, novo_peso in arestas_para_atualizar:
        aux = 0
        for (adjacente_bidirecional, peso) in self.m_graph[adjacente]:
            if adjacente_bidirecional == nome_antena:
                self.m_graph[adjacente][aux] = (nome_antena, novo_peso)
                break
            aux += 1

    print(f"\n Atualizado: {self.m_graph[nome_antena]}")
```

Imagem 5 - Função para atualizar arestas

2. **Listar Clientes:** é o método responsável por exibir uma lista de todos os clientes que permite visualizar de forma clara e organizada as informações sobre os mesmos, como o seu ID, a sua localização e a antena associada.

3. **Consultar Cliente:** é o método que permite ao utilizador especificar um ID de um cliente para poder averiguar as informações específicas do mesmo (ID, localização e antena associada).

4. **Remover Cliente:** Através do ID do cliente que o utilizador pretender remover é realizado uma procura na lista de cliente em prol de o remover da mesma. No entanto, como a remoção de um cliente condiciona as ligações estabelecidas entre antenas, ao remover o cliente é registada a antena à qual estava associado e é feita uma pesquisa para atualizar o valor das ligações da mesma (os pesos das arestas). Este processo é semelhante ao criar um cliente com a diferença de que ao utilizar o método de "atualizar_edges" a largura de banda utilizada que é redefinida na antena associada tem o seu valor decrementado no valor do serviço que o cliente ocupava.

Funcionalidades específicas de "Antena":

- Criar Antena
- Listar Antenas
- Consultar Antena
- Remover Antena

```python
def menu_antenas():
    print("\nMenu de Antenas")
    print("1. Criar Antena")
    print("2. Listar Antenas")
    print("3. Consultar Antena")
    print("4. Remover Antena")
    print("5. Voltar ao Menu Principal")
```

Imagem 6 - Menu de antenas

O menu das antenas é um ponto fulcral do projeto dado que é através do mesmo que são criados os nós que são utilizados na utilização dos algoritmos. A opção:

1. **Criar Antena:** Este método permite criar antenas caracterizadas pelas suas coordenadas (x,y) num plano bidimensional, pelo seu nome e ID, pelo seu raio de cobertura, largura de banda máxima, largura de banda utilizada (largura de banda ocupada pelos clientes associados), a percentagem da qualidade de serviço (inicialmente a 100% por não haver clientes associados) e por diversas listas que registam os clientes associados à antena, e, listas para cada um dos serviços disponíveis para os clientes para saber exatamente qual a distribuição da largura de banda disponível.
   Neste método além de se instanciar a antena é necessário estabelecer ligação com outras possíveis antenas vizinhas, para tal, deve ser percorrida a lista de antenas sob a condição de que a distância euclidiana entre ambas as antenas seja inferior a ambos os raios de cobertura (método "procura_edges"), ou seja, que qualquer uma das atenas tem um alcance suficiente para se comunicarem. Após se afirmarem as condições necessárias para estabelecer a ligação utiliza-se a função "add_edge" que cria as arestas (as ligações) entre as antenas de forma bidirecional para que ao utilizar os algoritmos de procura seja possível iniciar a pesquisa num nó à escolha do utilizador sem ter um rota estática e unidirecional (Imagem 6).

```
def procura_edges(self,antena1):
    if len(list(self.antenas.values()))!=1:
        for antena2 in self.antenas.values():
            distancia= self.calcular_distancia_euclidiana(antena1.get_x(),antena1.get_y(),antena2.get_x(),antena2.get_y())
            if (antena1.getName()!=antena2.getName() and distancia<=antena1.get_raio_cobertura() and distancia<=antena2.get_raio_cobertura()):
                self.add_edge(antena1,antena2,self.peso(antena1,antena2))

def add_edge(self, antena1, antena2, peso):
    antena1.addVizinho(antena2.getName())
    antena2.addVizinho(antena1.getName())
    self.m_graph[antena1.getName()].append((antena2.getName(), peso))
    self.m_graph[antena2.getName()].append((antena1.getName(), peso))
```

Imagem 7 - Menu das funções de adicionar e procurar arestas

2. **Listar Antena:** é o método responsável por exibir uma lista de todas as antenas e que permite visualizar de forma clara e organizada as informações sobre as mesmas, informações estas que podemos averiguar na tabela 1.
3. **Consultar Antena:** é o método que permite ao utilizador especificar o nome de uma antena para poder averiguar as informações específicas da mesma, informações estas que estão descritas na tabela 1.
4. **Remover Antena:** Este método permite simular uma situação onde decorre a necessidade de remoção de uma antena, seja por avaria, seja por uma eventual escolha da empresa que , por ser uma antena que geograficamente não é favorável, não é uma solução viável ao negócio. Como tal, através do nome da antena que o utilizador referir, é realizada uma procura na lista que contém todas as antenas para que se possa remover. No entanto, como a remoção de uma antena condiciona as ligações estabelecidas entre antenas, e a cobertura de clientes, ao remover a antena é necessário remover na lista respetivas das ligações ("m_graph[]") todas as ligações com as antenas vizinhas e vice-versa. Para que os clientes não fiquem desprovidos de cobertura, é percorrida a lista de clientes que a antena a remover tem associada, para que os mesmos se possam dissociar e através do método "associar_cliente_a_antena()" possam voltar a procurar uma outra antena à qual consigam se associar e garantir cobertura para continuarem os seus serviços (Imagem 7).
5.

```python
def remover_antena(self, name):

    if name in self.antenas:
        lista_clientes = self.antenas[name].get_lista_clientes()
        del self.antenas[name]
        for antena_adjacente,antenas in self.m_graph.items():
            for nome_antena, peso in antenas:
                if name == nome_antena:
                    self.m_graph[antena_adjacente].remove((name,peso)) ## situações onde a mesma é adjacente
        del self.m_graph[name] ##remove a antena do gráfico
        for cliente in lista_clientes:
            cliente.set_name_antena(None)
            self.associar_cliente_a_antena(cliente.get_id(), cliente.get_servico())
        print(f"Antena {name} removida com sucesso.")
    else:
        print(f"Antena com ID {name} não encontrada.")
```

Imagem 8 - Função de remover antena

Opções de visualização :

- Imprimir grafo
- Desenhar mapa
- Imprimir antenas de grafo
- Imprimir arestas de grafo

```python
print("3-Imprimir Grafo")
print("4-Desenhar Mapa")
print("5-Imprimir  antenas de Grafo")
print("6-Imprimir arestas de Grafo")
```

Imagem 9 - Desenhos e gráficos

1. **Imprimir grafo:** Este método permite ao utilizador visualizar na consola as antenas e as suas respectivas ligações com antenas adjacentes (nome, peso de ligação) como é demonstrado na Imagem 8.

```
{'1': [('3', 6)], '2': [('4', 16)], '3': [('1', 6), ('4', 11), ('5', 7)]
```

Imagem 10- Resultado ao selecionar a opção “Imprimir grafo”

2. **Desenhar mapa:** Esta opção permite dar ao utilizador a possibilidade de desenhar um mapa que representa , numa interface gráfica, a colocação/posicionamento das antenas e dos seus respectivos clientes, sendo os pontos vermelhos as antenas/estações base e os azuis, os clientes conectados às mesmas.
   Percorrendo o dicionário das antenas, desenhamos os pontos destas, após isso, percorremos a lista dos clientes associados a esta antena e desenhamos os seus pontos , estabelecendo uma ligação entre estes. Por fim é percorrido a lista do m_graph cuja key é esta antena , ligamos-las às suas adjacentes e desenhamos o peso no centro da ligação.

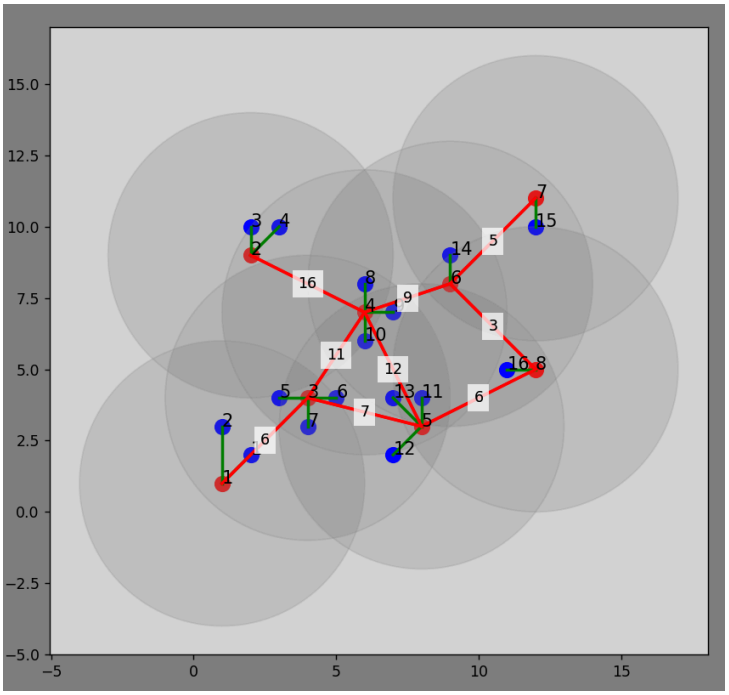

Imagem 11- Representação gráfica do mapa com o exemplo predefinido num ficheiro de texto chamado save.txt.

3. **Imprimir antenas de grafo:** Esta funcionalidade permite ao utilizador averiguar os nomes de todas as antenas que foram criadas até ao momento.

4. **Imprimir arestas de grafo:** Este método permite ao utilizador visualizar as ligações entre as antenas de forma mais simples como podemos averiguar na imagem 10.

```
1 ->3 custo:6
2 ->4 custo:16
3 ->1 custo:6
3 ->4 custo:11
3 ->5 custo:7
4 ->2 custo:16
4 ->3 custo:11
```

Imagem 12- Exemplo de output ao "imprimir arestas de grafo"

Algoritmos :

- DFS
- BFS

```
print("7-DFS")
print("8-BFS")
print("9-A*")
print("10-Gulosa")
```

- A*
- Gulosa

Imagem 13 - Algoritmos

No âmbito deste projeto foi necessário implementar algoritmos de procura informada e não informada para otimizar os custos das ligações entre antenas e para tal foram usados algoritmos como:

1. **DFS (Depth-First Search)** é um algoritmo de pesquisa que explora o caminho mais profundo de um grafo ou árvore antes de voltar ao início e explorar outros caminhos. DFS não leva em consideração o custo ou peso das arestas, o que o torna eficiente em termos de uso de memória, mas não garante a solução mais curta.

2. **BFS (Breadth-First Search)**: é um algoritmo de pesquisa que explora todos os nós de um nível antes de passar para o próximo nível. Ele utiliza uma fila para armazenar os nós a serem visitados, garantindo que a primeira solução encontrada seja a mais curta em termos de número de arestas. No entanto, o BFS pode ser mais lento e consumir mais memória em grafos de grande escala.

3. **A*:** É um algoritmo de pesquisa informada que utiliza tanto o custo acumulado até o nó atual quanto uma estimativa heurística do custo até o objetivo para encontrar o caminho mais curto de forma eficiente. A combinação desses dois fatores permite ao A* garantir a solução ótima, desde que a heurística utilizada seja admissível (não superestime o custo restante).

4. **Greedy** é um algoritmo de pesquisa que se baseia apenas numa heurística (h(n)), escolhendo o caminho que parece ser previsivelmente melhor. O mesmo não considera o custo acumulado, o que pode levar a decisões rápidas, mas nem sempre as mais eficientes ou ótimas. Greedy é útil para problemas onde a rapidez é mais importante que a precisão da solução.

Opção de gravar estado:

Imagem 14 - Funcionalidade Gravar estado da cidade

Na classe File existem duas função utilizadas para a leitura (parse) e para a escrita (write) num ficheiro que tem a gravação do estado da cidade naquele momento. Fizemos isto para, na execução do código, podermos já trabalhar com uma cidade em vez de dedicarmos tempo a construir esta comando a comando, criando todas as antenas e todos os clientes.

Com esta opção podemos então após a criação de novas componentes, gravar as alterações feitas na cidade , permitindo que continuemos o progresso numa futura execução do código.
Este ficheiro de texto tem as componentes antena e cliente no seguinte formato.
antena_nome_raio de cobertura_largura de banda máxima_coordenada x_coordenada y
cliente_id do cliente_coordenada x_coordenada y_servico
Exemplo do ficheiro com uma antena e um cliente antena (a linha de espaçamento entre eles é importante):
antena_1_5.0_10_1.0_1.0
cliente_1_2.0_2.0_jogos

Executando o código com o ficheiro exemplo e escolhendo a opção 4 (desenhar) obtemos:

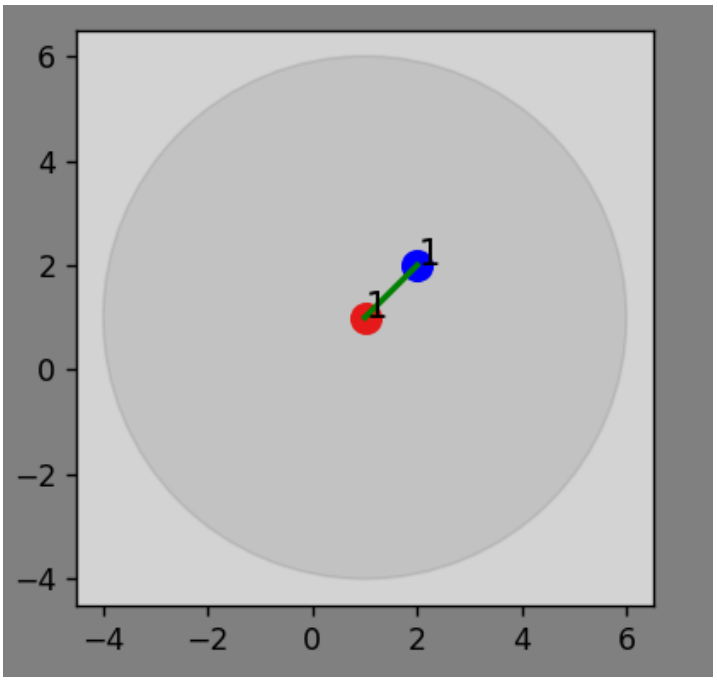

Imagem 15 - Desenho do mapa exemplo.

Simulação

Os resultados obtidos foram analisados com base na aplicação dos algoritmos de pesquisa DFS, BFS, A* e Gulosa, utilizando como entrada um arquivo que define a configuração inicial da rede. Neste arquivo, foram introduzidas:

- **Antenas**: Incluindo 8 antenas identificadas por IDs de 1 a 8, além de uma antena denominada "Lisboa".
- **Clientes**: Um total de 16 clientes identificados por IDs, associados a diversos serviços como streaming, chamadas e jogos.

A partir dessas informações, foi gerado um mapa gráfico que representa graficamente a distribuição geográfica das antenas e clientes, destacando suas conexões e alocações finais.

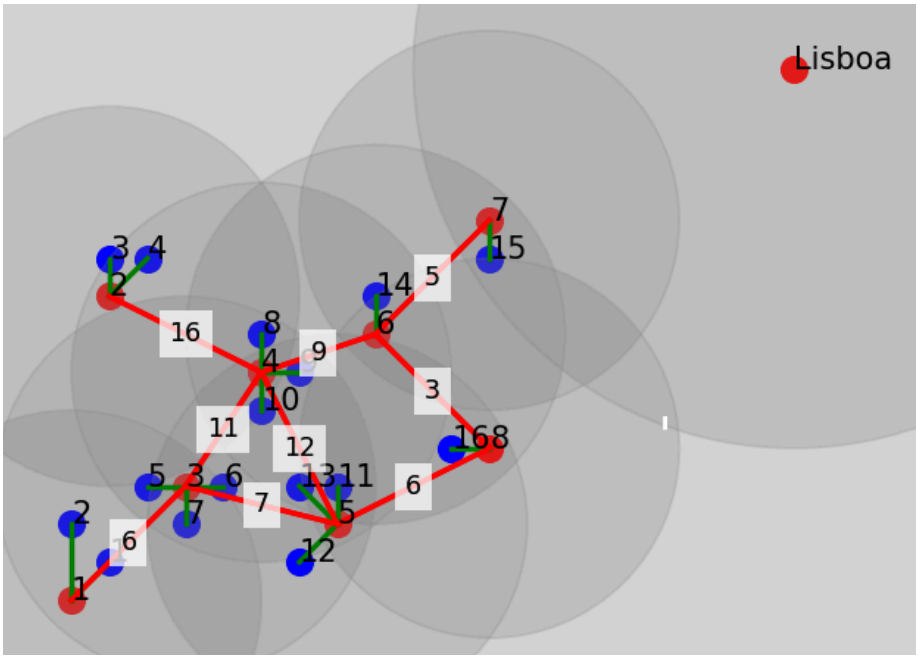

Imagem 16 : Mapa com uma antena adicional, “Lisboa”.

Os nós representados no mapa correspondem a pontos de acesso e conexão que compõem a rede. Estes pontos foram modelados considerando a distribuição espacial das antenas e os requisitos de cobertura entre o ponto inicial e o ponto final. A rota entre as antenas foi analisada utilizando diferentes algoritmos de busca, cada um com características específicas que afetam o caminho escolhido e o custo total associado. Além disso está representado também, no mapa, o custo associado às rotas entre as antenas.

Algoritmo	Caminho Encontrado	Custo Total	Notas
DFS	`['1', '3', '4', '5', '8', '6']`	`44`	Caminho mais longo e com maior custo.
BFS	`['1', '3', '4', '6']`	`31`	Caminho curto (em nós) e com custo moderado.
A*	`['1', '3', '5', '8', '6']`	`18`	Caminho mais eficiente (menor custo).
Gulosa	`['1', '3', '5', '8', '6']`	`18`	Mesmo resultado que o A* (boa heurística).

Conclusões Gerais

---

1. **A* destaca-se como o melhor algoritmo:**
    - Ele combina o custo real e a heurística para garantir o menor custo total. No problema, encontrou o caminho mais eficiente com custo 18.
2. **A Gulosa foi eficiente neste caso específico**:

    - Apesar de ignorar o custo acumulado, a heurística foi eficaz, resultando no mesmo caminho do A*. Contudo, essa eficiência não é garantida em todos os problemas.
3. **BFS é ótimo para caminhos curtos em termos de nós**:
    - Ele não considera custos, mas ainda encontrou um caminho com custo aceitável (31), que é menor que o DFS.
4. **DFS é o menos eficiente**:
    - Ele ignorou completamente o custo e encontrou um caminho mais longo e caro (44). O algoritmo é menos apropriado para problemas em que custos são importantes.

Conclusão

Este trabalho abordou a otimização de redes de telecomunicações através da aplicação de algoritmos de procura, com o objetivo de alocar utilizadores às estações base de forma eficiente. Utilizando técnicas de inteligência artificial e implementações em Python, foi possível minimizar distâncias, respeitar restrições de largura de banda e evitar sobrecargas, garantindo uma melhor qualidade de serviço.

Os resultados demonstraram que os algoritmos informados foram mais eficazes ao combinar eficiência computacional e qualidade das soluções, enquanto os algoritmos não informados, embora completos, apresentaram maior custo computacional. As simulações destacaram a importância de planejar redes que considerem fatores como capacidade, distância e consumo energético, contribuindo para a sustentabilidade e desempenho operacional.

Este projeto permitiu aplicar conceitos teóricos a problemas reais, consolidando conhecimentos em inteligência artificial e telecomunicações, e demonstrando o impacto positivo de soluções bem estruturadas para desafios práticos.