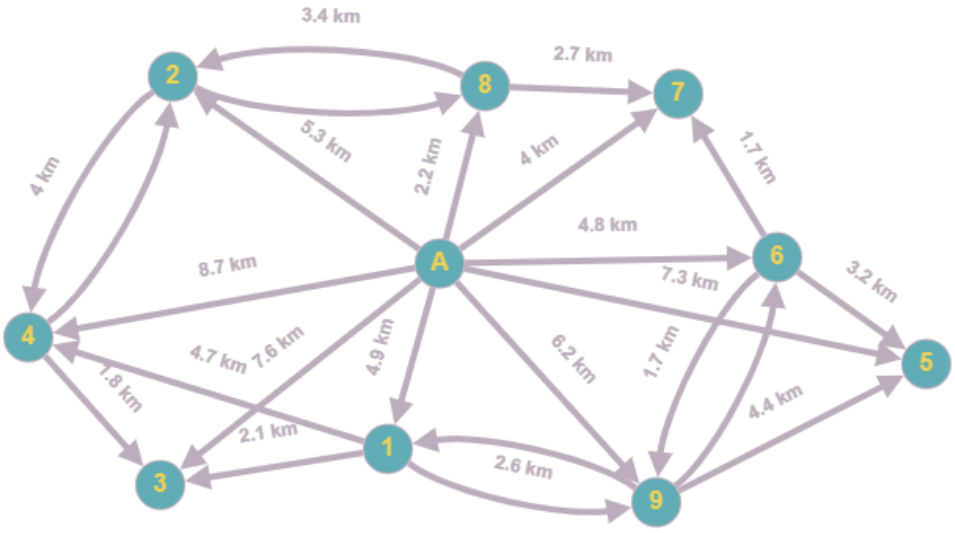


Otimização e Automatização de uma Ferramenta de Planeamento de Auditorias de Segurança Alimentar

Rui Alberto Ribeiro Pereira
Mestrado de Engenharia Matemática
Departamento de Matemática
2022

Orientador
Sílvio Marques de Almeida Gama
Professor associado, Faculdade de Ciências

Coorientador
Maria do Carmo Miranda Guedes
Professora aposentada, Faculdade de Ciências

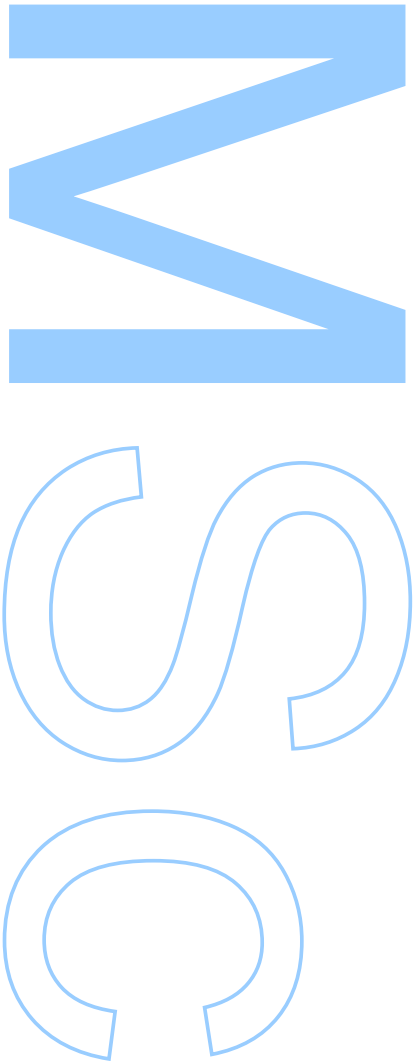
U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____



UNIVERSIDADE DO PORTO

RELATÓRIO DE ESTÁGIO

Otimização e Automatização de uma Ferramenta de Planeamento de Auditorias de Segurança Alimentar

Autor:

Rui Alberto Ribeiro Pereira

Orientador:

Sílvio Marques de Almeida Gama

Coorientador:

Maria do Carmo Miranda Guedes

Relatório de Estágio submetido em cumprimento dos requisitos para obtenção do grau de mestre em Engenharia Matemática

na

Faculdade de Ciências da Universidade do Porto

Departamento de Matemática

28 de novembro de 2022

Declaração de Honra

Eu, Rui Alberto Ribeiro Pereira, inscrito no Mestrado em Engenharia Matemática da Faculdade de Ciências da Universidade do Porto declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U.Porto, que o conteúdo da presente relatório de estágio reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta relatório de estágio, declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente relatório de estágio quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor.

Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

Rui Alberto Ribeiro Pereira

28 de Outubro de 2022

“ Where there’s a will there’s a way ”

George Herbert

Agradecimentos

Em primeiro lugar um obrigado ao meu ano por todos os momentos passados e por terem estado sempre lá presentes. Agradeço ao meu pai e aos meus restantes amigos por todo o apoio e por terem acreditado em mim. Gostaria também de dar um especial agradecimento ao Duarte Pinto pela incansável ajuda.

Por fim, e não menos importante, quero agradecer aos meus orientadores Prof. Sílvio Gama, Prof. Maria do Carmo, Drª. Mónica Caldeira e Eng. Catarina Furtado por todo o tempo disponibilizado e acompanhamento.

Resumo

Este Relatório de Estágio descreve o trabalho realizado na empresa MC * do Grupo Sonae ao longo do estágio curricular, ocorrido no ano letivo de 2021–2022, no âmbito do Mestrado de Engenharia Matemática da Faculdade de Ciências da Universidade do Porto.

O presente trabalho teve como objetivo desenvolver uma ferramenta de planeamento para as auditorias de segurança alimentar realizadas nas lojas Continente (cerca de 1200 auditorias por ano) pela direção de auditoria interna da MC. A ferramenta deve ser de fácil utilização, dinâmica e flexível, de modo a reduzir o tempo dedicado no planeamento e melhorando a eficiência de todo o seu processo. Neste sentido, foi desenvolvido um programa em *Python* que cria um ou mais planos de sete dias de auditoria para qualquer auditor que se pretenda, no qual atribui lojas de forma a obedecer todas as restrições estabelecidas. Inclui também a opção de alterar o plano manualmente conferindo flexibilidade à ferramenta e permitindo ajustar os planeamentos a necessidades especificas e imprevistas.

*MC - Modelo Continente

Abstract

This Report describes all the work carried out at MC of the Sonae Group during the curricular internship, which took place in the academic year 2021–2022, within the scope of the Mathematical Engineering course at the Sciences Faculty of Porto University.

The objective of the present work was to develop a planning tool for food safety audits performed by the internal audit department of Continente stores (about 1200 audits per year) (MC). The tool must be easy to use, dynamic and flexible in order to reduce the time needed for planning and improve the efficiency of the whole process. With this in mind, a program was developed in Python that creates one or more seven-day audit schedules for each designated auditor, with stores assigned to meet all established constraints. It also includes the possibility of manually modifying the plan, which gives flexibility to the tool and allows adapting the plans to specific and unforeseen needs.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A MC é líder do mercado nacional, no retalho alimentar, com um conjunto de negócios distintos que oferecem uma variada gama de produtos: Continente (hipermercados), Continente Modelo e Continente Bom Dia (supermercados de conveniência), Meu Super (lojas de proximidade e formato *franchising*), Bagga (cafetaria), Go Natural (supermercados e restaurantes saudáveis), Note! (livraria, papelaria e presentes), ZU (produtos e serviços para cães e gatos), Well's (saúde, bem-estar e ótica), Dr. Well's (clínicas de medicina dentária e medicina estética), entre outros. Inclui também gestão de imobiliário de ativos de retalho.

FIGURA 1.1: Portefólio de negócios da MC.

A MC por sua vez faz parte da Sonae, que é uma multinacional que gere um portefólio diversificado de negócios nas áreas de retalho, serviços financeiros, tecnologia, centros comerciais e telecomunicações. No seu portefólio tem a MC (75.01%), Zeitreel, Worten, ISRG (30%), Universo, BrightPixel (90%), Sierra (80%) e NOS (30.8%). Está também presente em sessenta e dois países.

A atividade da MC teve origem em 1985, dentro do Grupo Sonae, com a abertura do primeiro hipermercado em Portugal, situado em Matosinhos. Hoje em dia tem mais de 1300 lojas, um volume de negócios superior a 5100M€, mais de quatro milhões de famílias cliente e mais de trinta e seis mil colaboradores.

FIGURA 1.2: Torre de escritórios da MC, em Matosinhos.

A Direção de Auditoria Interna da MC realiza, por ano, cerca de 2000 auditorias de segurança alimentar às unidades alimentares. No início da realização do estágio curricular, o planeamento de todas essas auditorias era realizado de forma manual tendo um custo aproximado de quarenta dias de trabalho de um colaborador.

A Direção de Auditoria Interna tem como principais responsabilidades a execução de auditorias aos processos mais relevantes dos negócios, auditorias de segurança alimentar e auditorias aos sistemas de informação do Centro Corporativo e dos Negócios de Retalho.

A função de Auditoria Interna tem por missão identificar e avaliar a eficácia e eficiência da gestão e do controlo dos riscos dos processos de negócio.

Na MC existe um programa de auditorias de segurança alimentar às lojas, cafetarias, restaurantes e entrepostos, com o objetivo de verificar de forma sistemática o cumprimento das normas legais e das regras internas de segurança alimentar.

A auditoria de segurança alimentar é independente e sistemática, acompanha os planos de ações para mitigação dos riscos, apoia a empresa na definição de requisitos de controlo de segurança alimentar e na identificação de áreas de risco e legislação aplicável.

A auditoria interna apresenta um reporte trimestral das principais conclusões e planos de ações definidos pelos auditados: aos Pelouros dos negócios, Comissão de Liderança da MC, Comissão de Auditoria da Sonae e da MC, Comissão de Auditoria e Finanças da Sonae.

A realização de uma auditoria é um processo que envolve várias etapas. Primeiro inicia-se pela sua preparação na qual há análises de documentação com informação digital centralizada. Após a preparação há o trabalho de campo, onde existe avaliação de todos os pontos de controle de forma a verificar se os procedimentos estão implementados e se existe cumprimento de todos os padrões de qualidade e segurança alimentar e há seguimento da auditoria anteriormente realizada. Todas as auditorias são acompanhadas por chefias da unidade. Depois do trabalho de campo, existe uma reunião final de auditoria e emissão de um primeiro relatório, seguido de um relatório final e de planos de ação do auditado. Também é essencial para o processo que haja uma análise de risco, que é efetuada para cada tipologia de unidade. Nela são identificados os controlos existentes, tipificados os riscos *standard* e é associada a legislação e procedimentos internos ou códigos de boas práticas aplicáveis.

1.1 Condições iniciais e restrições

No trabalho deste estágio, apenas foram consideradas, para o desenvolvimento da ferramenta, o planeamento para as auditorias às lojas Continente (CNT), Continente Modelo (MH) e Continente Bom Dia (BD). Atualmente, para esses três tipos de lojas, existem trezentas e três lojas em funcionamento, número este que tem vindo, gradualmente, a aumentar. A equipa de auditoria interna de segurança alimentar é responsável por realizar as auditorias às referidas lojas e está organizada em dois locais - Matosinhos e Carnaxide. Considera-se que cada auditor dedica três dias por semana a trabalho de campo e os restantes dias à preparação das auditorias e à elaboração de relatórios.

No âmbito deste trabalho consideraram-se quatro tipos de auditorias de *checklists*: uma auditoria de segurança alimentar geral e três auditorias especificas a secções de frescos. Ao longo do Relatório de Estágio ir-se-á usar as designações Geral, Específica 1, Específica 2 e Específica 3 para fazer referência às auditorias referidas. O objetivo é, em cada ano, realizar os quatro tipos de auditorias a todas as lojas. Portanto, os auditores precisam de realizar mais de 1200 auditorias por ano. Por sua vez, as auditorias têm todas durações médias diferentes e algumas podem requisitar mais de um auditor para a

realizar. Na tabela 1.1 abaixo pode-se encontrar as horas e auditores necessários para cada *checklist*.

	Horas	Auditores Continente	Auditores Modelo/Bom Dia
Geral	8 (1 dia)	2	1
Específica 1	2	1	1
Específica 2	3	2	1
Específica 3	2.5	1	1

TABELA 1.1: Recursos necessários para cada tipo de auditoria

Quando uma das viagens para o local de auditoria envolve uma viagem de duas horas ou mais, chamada viagem longa, o auditor fica a dormir no local e realiza auditorias em lojas próximas no dia seguinte, de forma a reduzir tempo e custos de viagens. Quando isso acontece, o auditor não volta a realizar uma viagem tão longa pelo menos durante uma semana. Os custos de viagem estão positivamente correlacionados com o tempo gasto em viagens entre auditorias.

As auditorias não são avisadas e não devem ser previsíveis por parte das lojas. Dessa forma, as quatro auditorias por loja devem ser realizadas com um determinado intervalo de tempo entre elas. Para além disso, tem que existir uma distribuição nacional uniforme de auditorias por tipo de loja, ao longo do ano, tendo em conta que todas as lojas têm que ser visitadas. Ou seja, as percentagens de auditorias realizadas por concelhos devem ser relativamente as mesmas.

A ferramenta deve também ser flexível para acomodar possíveis imprevistos como, por exemplo, ausência dos auditores por doença ou necessidade de realocar recursos.

Esta tese está organizada do seguinte modo. No Capítulo 2 são expostos alguns potenciais métodos para resolução do problema, como modelos matemáticos de programação linear e de teoria de grafos e, de seguida, mostra-se um exemplo ilustrativo ou aplicação direta ao problema. O Capítulo 3 consiste no desenvolvimento da ferramenta de planeamento usando o software *Python*, na descrição das funções e métodos usados para tal e no exemplo de resultados obtidos da ferramenta desenvolvida. No último Capítulo apresentam-se alguns resultados, conclusões finais e perspetivas para trabalho futuro.

Capítulo 2

Modelos matemáticos para planeamento de auditorias

Ao longo dos anos, o problema do planeamento de auditorias tem sido cada vez mais relevante e cada vez mais investigado. Grande parte da literatura existente incide, principalmente, à volta de auditorias financeiras ou à volta de outro tipo de auditorias contendo apenas uma única auditoria, dividida em diversas tarefas, em que o objetivo é realizar um planeamento das tarefas da auditoria. Podemos ver exemplos disso no trabalho de Mohamed [2015] [1], que é um estado-de-arte sobre modelos de investigação operacional desenvolvidos para planeamento de auditorias internas, de Chang *et. al* [2002] [2] que é um artigo sobre como resolver o problema de auditorias em múltiplas e grandes tarefas, especialmente em empresas de contabilidade, em que o planeamento está focado na distribuição das várias tarefas da auditoria pelos auditores, de Rossi *et. al* [2010] [3] que está relacionado com auditorias que ocorrem durante longos períodos de tempo, envolvendo uma equipa de auditores e em que não se podem realizar auditorias simultâneas e no trabalho de Dodin *et. al* [1998] [4] que também trata de auditorias que são divididas em várias tarefas. O caso das auditorias de segurança alimentar realizadas na MC é um caso mais especial, no sentido em que o interesse é o planeamento das várias auditorias em si, considerando que cada auditoria é apenas constituída por uma tarefa, ao contrário dos típicos casos de planeamento de auditorias.

Pesquisas anteriores sobre o problema de planeamento de auditorias tentaram resolver o problema de diversas maneiras. Pode-se ver exemplos onde foi usado um modelo de programação linear [5], um modelo de programação linear inteira (modelo BZ) [6], um modelo estocástico [7] e onde foi usado um modelo PERT/CPM [8].

Nas secções seguintes irão ser analisados vários modelos usados ou investigados para a resolução do problema do planeamento de auditorias em diversos contextos. A ordem dos modelos corresponde à ordem em que foram analisados, tendo em conta a complexidade do problema. Inicialmente, considerou-se um problema com poucas restrições e pouca complexidade e, para tal, um problema de programação linear bastava. Mas ao longo do tempo em que foram sendo consideradas mais variáveis e restrições, foi necessário considerar novos modelos que fossem capazes de as suportar.

2.1 Programação linear

Os problemas de otimização, na sua forma mais genérica, são representáveis por uma minimização ou uma maximização de uma função escalar $F(x_1, ..., x_N)$ satisfazendo as restrições

$$\begin{aligned} g_1(x_1, \ldots, x_N) &= 0\,, \\ &\ldots \\ g_M(x_1, \ldots, x_N) &= 0\,, \end{aligned} \tag{2.1}$$

em que:

- $x_1, \ldots, x_N$ $(x_i \in \mathbb{R},\ \forall i \in \{1, \ldots, N\})$ são as variáveis que representam as incógnitas do problema e que se definem para domínios já preestabelecidos. São chamadas variáveis de decisão;
- $g_1, \ldots, g_M$ são as restrições a satisfazer para cada solução, de maneira a que a solução seja possível. Estas restrições definem o conjunto de soluções admissíveis ou viáveis;
- $F(x_1, ..., x_N)$ é a função objetivo, também designada por função custo.

Tanto as variáveis, como as restrições e a função objetivo dependem do tipo do problema a ser resolvido. Assim, os máximos ou os mínimos obtidos desta forma, dizem-se condicionados pelas restrições e pelos domínios estabelecidos.

Designa-se por programação linear um conjunto de modelos e de técnicas de resolução em que tanto a função objetivo como as restrições são lineares e que torne possível resolver o problema de otimização acima apresentado.

Para além das referências mencionadas anteriormente, também podemos encontrar exemplos de programação linear nos livros de L. Valadares Tavares [9] e Bazarra *et. al*[10].

2.1.1 Forma canónica do problema de programação linear

A forma canónica do problema de programação linear de um problema com M restrições e N variáveis de decisão pode ser representada da seguinte maneira:

$$\begin{aligned} \text{Min } F = {} & c_1x_1 + c_2x_2 + \cdots + c_Nx_N \\ & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N = b_1 \\ & a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N = b_2 \\ & \cdots \\ & a_{M1}x_1 + a_{M2}x_2 + \cdots + a_{MN}x_N = b_M \\ & x_i \geqslant 0 \quad (i = 1, 2, \ldots, N) \end{aligned} \tag{2.2}$$

Se o problema não estiver apresentado dessa forma, é simples transformá-lo num problema equivalente à forma canónica. Por exemplo, se o problema for um problema de maximização, basta multiplicar a função objetivo por -1 e dessa forma obtemos um problema de minimização, visto que $-\text{Max } (-F) = \text{Min } F$.

2.1.2 Problema da mochila (*knapsack problem*)

O problema da mochila pode ser, genericamente, definido do seguinte modo: suponha-se que um vendedor ambulante tem uma mochila onde deve guardar todo o seu material para vender, dentro de vários possíveis itens e considerando a capacidade máxima da mochila. Cada item tem o seu respetivo valor e o vendedor deve de escolher os itens para vender de forma a conseguir ter na sua mochila o maior valor possível para vender [11] [12]. Ele também pode ser visto como um problema de programação linear inteira [13] [14], também conhecido como programação inteira, que é um problema de programação linear em que algumas ou todas as variáveis estão restritas a serem valores inteiros [15].

A cada item i está associado:

- o seu peso (ou volume, etc.) $= p_i > 0$
- o seu valor (lucro, retorno, etc.) $= v_i > 0$

2.1.2.1 Problema da mochila inteiro

Também simplesmente conhecido como problema da mochila, o problema da mochila inteiro não possui qualquer limite na quantidade de itens selecionados. Desta forma, temos a seguinte modelação:

Variáveis de decisão:

- x_i - quantidades do tipo i selecionados, $i = 1, \ldots, m$

A função objetivo diz respeito ao valor total dos itens metidos na mochila:

$$z = \sum_{i=1}^{m} v_i x_i \,. \tag{2.3}$$

2.1.2.2 Problema da mochila 0-1

O problema da mochila 0-1, também conhecido como problema binário da mochila, é o problema da mochila mais estudado e, muito provavelmente, o mais importante. Alguns motivos para isso, são o facto de pertencer à classe NP-Difícil, como referido em [16], o facto de poder ser facilmente definido como um problema de programação linear inteira e por também aparecer como um subproblema em muitos outros problemas complexos.

O problema pode ser modelado da seguinte maneira:

Variáveis de decisão:

$$x_i = \begin{cases} 1 & \text{se o item } i \text{ é metido na mochila,} \\ 0 & \text{caso contrário.} \end{cases}$$

em que $i = 1, \ldots, m$.

Função objetivo:

$$z = \sum_{i=1}^{m} v_i x_i \,. \tag{2.4}$$

Restrições:

O peso total dos itens metidos na mochila não pode ultrapassar a capacidade limite P da mochila

$$z = \sum_{i=1}^{m} p_i x_i \leqslant P \,. \tag{2.5}$$

2.1.3 Modelo ilustrativo simples de auditoria

A abordagem inicial para lidar com o problema de auditorias foi encará-lo como um problema bastante simples e, sequencialmente, ir acrescentando mais condições e restrições.

Para tal, optou-se por considerar inicialmente um problema "ilustrativo", com dados não representativos dos casos reais de auditorias na empresa, com apenas:

- 1 auditor: A_1 ;
- 5 lojas: L_1 , $\cdots$, L_5 ;
- Tempos de trabalho em auditoria por cada loja: t_{L_1} , $\cdots$, t_{L_5} : escolhidos aleatoriamente de um intervalo entre 60 e 120 minutos;
- 1 dia de trabalho.

Para o exemplo em questão, encontram-se representados na seguinte matriz M os valores utilizados para os tempos de viagem (em minutos), em que $d_{i,j}$ é o tempo de viagem entre as lojas L_i e L_j :

$$M = (d_{i,j}) = \begin{array}{c} \\ A_1 \\ L_1 \\ L_2 \\ L_3 \\ L_4 \\ L_5 \end{array} \begin{array}{c} \begin{array}{cccccc} A_1 & L_1 & L_2 & L_3 & L_4 & L_5 \end{array} \\ \begin{pmatrix} 0 & 10 & 15 & 12 & 7 & 14 \\ 10 & 0 & 6 & 9 & 13 & 17 \\ 15 & 6 & 0 & 15 & 20 & 18 \\ 12 & 9 & 15 & 0 & 5 & 3 \\ 7 & 13 & 20 & 5 & 0 & 5 \\ 14 & 17 & 18 & 3 & 5 & 0 \end{pmatrix} \end{array}$$

Matriz dos tempos de viagem, em minutos, do problema ilustrativo

No contexto do problema, na matriz M , a entrada A_1 está a representar a casa do auditor. Ou seja, por exemplo, $d(A_1, L_1)$ é o tempo entre a casa do auditor e a loja L_1, ou, dito por outras palavras, o tempo de viagem de ida inicial do auditor para a loja L_1. Para este exemplo, considerou-se que a matriz é simétrica, ou seja, que o tempo entre a loja L_i e a loja L_j é o mesmo entre a loja L_j e a loja L_i . Na prática, o afirmado não é necessariamente verdade numa boa parte dos casos visto que, por vezes, existem diferenças devido a diferenças no trânsito ou no próprio caminho como, por exemplo, estradas de sentido único.

Para os tempos, em minutos, de realização de auditorias de cada loja, considerou-se $t_{L_1} = 97$, $t_{L_2} = 118$, $t_{L_3} = 65$, $t_{L_4} = 77$, e $t_{L_5} = 89$, isto é:

$$(t_{L_1}, t_{L_2}, t_{L_3}, t_{L_4}, t_{L_5}) = (97, 118, 65, 77, 89)\,. \tag{2.6}$$

A maneira mais simples e óbvia de enfrentar o problema é aplicar um modelo de programação linear na qual se procura maximizar ou minimizar alguma variável.

No contexto do problema na empresa, há duas hipóteses. Ou se procura maximizar o tempo gasto em realizar auditorias por dia ou se procura minimizar os tempos/custos de viagem, enquanto se cumpre um mínimo de horas de trabalho diário. Como a prioridade da empresa é realizar a maior quantidade de auditorias possíveis optou-se por procurar maximizar o tempo gasto na realização de auditorias por dia. Considera-se que os auditores trabalham, por dia, no mínimo seis horas e trinta minutos e no máximo nove horas.

O auditor também tem a possibilidade de escolher auditar uma loja ou não, semelhante ao problema da mochila em que o vendedor ambulante também tem a opção de decidir guardar ou não um certo item na sua mochila. Dessa forma, pode-se encarar o problema como um semelhante ao problema da mochila.

Seja:

$$x_{L_i} = \begin{cases} 1 & \text{a loja } L_i \text{ é auditada,} \\ 0 & \text{caso contrário.} \end{cases}$$

Função objetivo:

$$z = \sum_{i=1}^{5} t_{L_i} x_{L_i} - \sum_{(i,j)\in C} d_{i,j}\ , \tag{2.7}$$

onde C é o conjunto de pares ordenados (i, j) em que i é o local inicial e j é o local final da viagem, $i \neq j$, e em que a sequência de locais finais $S = j_1, \ldots, j_n$ não tem nenhum local repetido. Ou seja, na situação do exemplo em questão, caso o auditor A_1 tenha decidido visitar as lojas L_1 e L_4, só lhe resta poder visitar a sua casa A_1 e as lojas L_2, L_3 e L_5.

Restrições:

O total de tempo T gasto, tanto em auditorias como em viagens, tem de ser superior a seis horas e meia (390 minutos) e inferior a nove horas (540 minutos):

$$390 \leqslant T = \sum_{i=1}^{5} t_{L_i} x_{L_i} + \sum_{(i,j)\in C} d_{i,j} \leqslant 540\,. \tag{2.8}$$

Neste exemplo, consegue-se facilmente notar que é possível realizar todas as auditorias possíveis no dia em questão, visto a soma total do tempo de auditoria das lojas é

$97 + 118 + 65 + 77 + 89 = 446$ minutos e arranjamos, pelo menos, um caminho possível tal que o total de tempo não ultrapasse as nove horas, como, por exemplo, a sequência $S = L_1, L_2, L_3, L_4, L_5$ em que o caminho é percorrido em 58 minutos. Falta só encontrar o caminho tal que a função objetivo z tome o menor valor. Esse caminho é dado pela sequência $S = L_4, L_5, L_3, L_1, L_2$ em que a soma dos tempos é igual a 30 minutos.

No caso deste exemplo, o auditor era capaz de realizar todas as auditorias num dia, o que não é o caso no contexto da empresa. Então, de seguida, iremos ver um exemplo em que o auditor tem de optar por realizar apenas algumas auditorias. Para isso, considere-se uma nova matriz de tempos de viagem M_2 e novos tempos de auditoria, desta vez escolhidos aleatoriamente de um intervalo entre 90 e 210 minutos.

$$M_2 = \begin{array}{c} \\ A_1 \\ L_1 \\ L_2 \\ L_3 \\ L_4 \\ L_5 \end{array} \begin{array}{c} \begin{array}{cccccc} A_1 & L_1 & L_2 & L_3 & L_4 & L_5 \end{array} \\ \begin{pmatrix} 0 & 26 & 13 & 20 & 33 & 8 \\ 26 & 0 & 22 & 29 & 44 & 12 \\ 13 & 22 & 0 & 20 & 35 & 42 \\ 20 & 29 & 20 & 0 & 13 & 7 \\ 33 & 44 & 35 & 13 & 0 & 36 \\ 8 & 12 & 42 & 7 & 36 & 0 \end{pmatrix} \end{array}$$

Matriz dos tempos, em minutos, do segundo problema ilustrativo

Novos tempos de auditoria:

$$(t_{L_1}, t_{L_2}, t_{L_3}, t_{L_4}, t_{L_5}) = (174, 154, 178, 123, 95)\,. \tag{2.9}$$

Neste caso, de forma a maximizar o tempo gasto em auditorias e obedecendo às restrições estabelecidas, o melhor percurso de auditorias é dado pela sequência $S = L2, L3, L4$ na qual o auditor gasta 455 minutos em auditoria e 46 em viagem, sendo um total de 501 minutos de trabalho total.

O problema é que para já, ainda não estão a ser considerados mais do que um auditor, vários dias de trabalho, mais do que um tipo de auditoria para a mesma loja, ou seja, a possibilidade de dentro da mesma loja escolher auditorias com tempos de duração diferentes, e muitas outras restrições impostas pela empresa.

De forma a poder considerar tanto mais dias, como mais auditores e mais tipos de auditoria, optou-se por mudar a abordagem para um problema de teoria de grafos.

2.2 Teoria de grafos

Muitos problemas do mundo real podem ser representados como diagramas que consistem num conjunto de pontos, com linhas unindo alguns desses pontos. Um exemplo muito atual é na área de Análise de Redes, que está muito fortemente relacionada com Teoria de Grafos [17] em que os perfis de utilizadores de redes sociais e as suas conexões são representados por pontos e linhas, respetivamente. É de notar que a ligação entre os pontos é imaterial. A partir da abstração matemática de situações deste género, chegamos ao conceito de grafo [18].

2.2.1 Modelo

Um grafo G pode ser visto como um triplo ordenado (V(G), E(G), f_G) consistindo num conjunto não-vazio de vértices V(G), um conjunto E(G), disjunto de V(G), constituído pelas arestas (*edges*), e uma função de incidência f_G que associa a cada aresta de G um par não ordenado de vértices de G. Se $e_1 = \{x_1, x_2\}$ for uma aresta e x_1 e x_2 vértices tais que $f_G(e_1) = x_1x_2$, então diz-se que e_1 une x_1 e x_2. Também dizemos que a aresta e_1 incide em x_1 e x_2 e que x_1 e x_2 são as pontas da aresta. Caso x_ix_j seja uma aresta, dizemos que os vértices x_i e x_j são adjacentes.

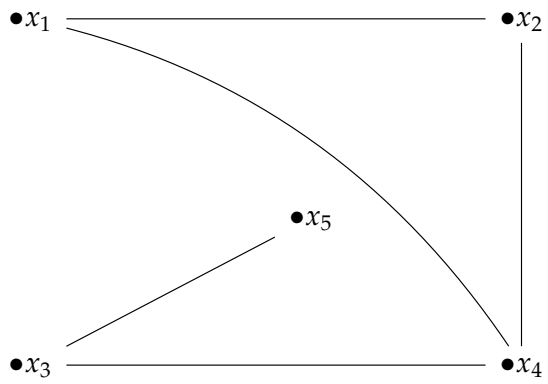


FIGURA 2.1: Exemplo de um grafo não direcionado.

A palavra "grafo"é derivada da palavra inglesa "*graph*", que por sua vez origina do facto de grafos poderem ser representados graficamente, como podemos observar na figura 2.1. Nesse exemplo, o grafo não possui pesos e é não direcionado. Na figura 2.2 está representado um exemplo de um grafo direcionado.

As arestas dos grafos também podem ter um peso associado, normalmente sendo um valor inteiro não-negativo, que usualmente está associado ao "custo"da aresta.

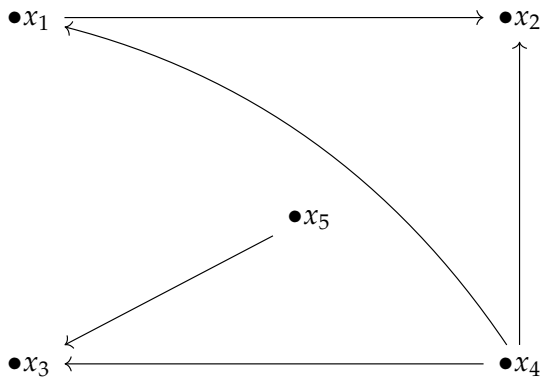


FIGURA 2.2: Exemplo de um grafo direcionado.

Outra maneira de representar um grafo, quer seja direcionado ou não, é através de uma matriz de adjacência. Nas figuras 2.1 e 2.2 encontram-se as representações de um grafo não direcionado e de um grafo direcionado, respetivamente. Notar que a matriz de adjacência de um grafo não direcionado é simétrica.

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Matriz de adjacência do grafo da figura 2.1

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Matriz de adjacência do grafo da figura 2.2

A representação pela matriz de adjacência de um grafo $G = (V, E)$, onde V representa o conjunto dos vértices e E o conjunto das arestas, consiste numa matriz $|V| \times |V|$, $A = (a_{ij})$, onde $|V|$ é o número de vértices de G, tal que:

$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E, \\ 0 & \text{caso contrário.} \end{cases}$$

Numa matriz não direcionada $x_i x_j$ e $x_j x_i$ representam a mesma aresta, enquanto que numa matriz direcionada, $x_i x_j$ representa a aresta cuja fonte é x_i e alvo é x_j, e $x_j x_i$ representa a aresta cuja fonte é x_j e alvo é x_i.

Para representar um grafo com pesos através de uma matriz de adjacência, basta substituir o valor 1, na respetiva entrada, pelo valor do peso da aresta.

2.2.1.1 Problema do caminho mais curto

O problema do caminho mais curto é um problema que pode ser definido como o problema de encontrar o caminho mais curto desde um ponto inicial até a um destino final, ou seja, o caminho cuja a soma dos pesos das arestas de um grafo seja mínima. É um problema de grande interesse, por exemplo, nas áreas de transporte [19], telecomunicação e análise de redes que, por sua vez, tem muitas outras aplicações no mundo real [20].

Existe uma grande quantidade algoritmos para o resolver mas o algoritmo mais importante e dos mais estudados, é o algoritmo de Dijsktra [21]. Apresenta-se abaixo a seguinte forma para o algoritmo de Dijsktra.

Algoritmo de Dijkstra [22]

Para cada vértice v de um grafo G o algoritmo procura atribuir um rótulo permanente que determina a distância mínima desde um ponto inicial i ao vértice v. Os rótulos são da forma $[d[v], a]$, onde $d[v]$ é a distância em km e a é o vértice antecessor, e cada rótulo pode ser de dois tipos: temporário ou permanente. Sequencialmente, o algoritmo tenta diminuir o valor das distâncias dos rótulos de cada vértice. Inicialmente é escolhido o ponto inicial i e é-lhe atribuído um rótulo permanente com valor da distância igual a zero ($d[i] = 0$) e aos restantes vértices são atribuídos rótulos temporários com distância de peso infinito ($d[v] = \infty$). O algoritmo de Dijkstra consiste em n iterações e termina quando todos os vértices forem permanentes. Caso ainda existam vértices temporários, o algoritmo escolhe entre eles o vértice v_k cujo rótulo tenha o menor valor $d[v_k]$ para tornar permanente. Depois, para cada vértice adjacente v_{a_i} de v_k, é considerado um novo rótulo temporário com distância igual à soma do valor do rótulo no vértice inicial, $d[v_k]$, com a distância D entre a aresta que une v_{a_i} e v_k. Se o valor da nova distância for inferior à distância do rótulo de v_{a_i}, então mudamos o valor do rótulo temporário para a nova distância:

$$d[\text{vértice adjacente}] = \min(d[\text{vértice adjacente}], d[v] + D)\,. \tag{2.10}$$

Quando todos os vértices estiverem permanentes, o algoritmo termina. Caso um vértice não esteja conectado com o vértice inicial, ele irá permanecer com um rótulo permanente de infinito.

Para se saber o caminho mais curto entre o vértice inicial i e um certo vértice v, tal que $i \neq v$, temos de identificar o vetor p onde $p[v]$ é o penúltimo vértice do caminho mais curto até v, ou seja, o caminho mais curto de i a v pode ser escrito da seguinte forma:

$$P = (i, \ldots, p[p[p[v]]], p[p[v]], p[v], v)\,. \tag{2.11}$$

De seguida, iremos ver um pequeno exemplo para ilustrar o algoritmo de Dijkstra.

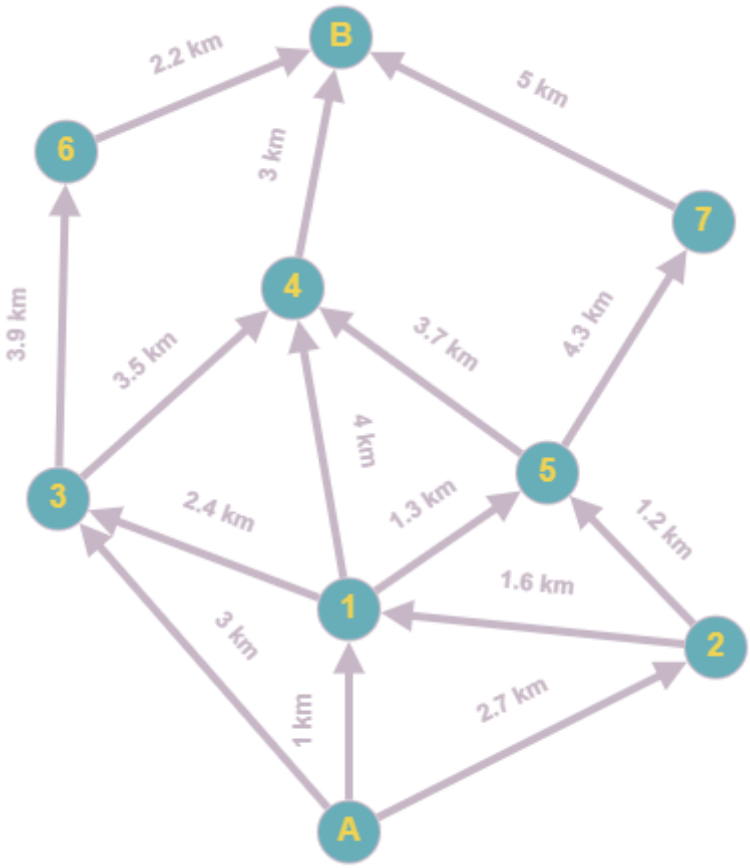


FIGURA 2.3: Exemplo de um caminho possível entre local A e B.

Suponhamos que o auditor gostava de fazer uma viagem até ao local B, partindo de A. Na figura 2.3 podemos observar as rotas possíveis entre os dois locais. Para calcular o caminho mais curto entre A e B usando o algoritmo de Dijsktra, primeiro começa-se por atribuir o rótulo $[0, -]$ ao vértice A visto ser o vértice inicial e não ter nenhum antecessor.

Vértice	Rótulo	Estado do Rótulo
A	$[0, -]$	Permanente

TABELA 2.1: Iteração 1

Como a partir do vértice A podemos chegar aos vértices 1, 2 e 3, estes ganham um rótulo temporário.

Vértice	Rótulo	Estado do Rótulo
A	$[0, -]$	Permanente
1	$[1, A]$	Temporário
2	$[2.7, A]$	Temporário
3	$[3, A]$	Temporário

TABELA 2.2: Iteração 2

Visto que o vértice 1 é o que possui o rótulo com a menor distância, o estado do seu rótulo transforma-se em permanente e ele passa a ser o novo vértice inicial. Os vértices adjacentes ao novo vértice inicial 1 são os vértices 3, 4 e 5 e estes ganham novos rótulos temporários.

Vértice	Rótulo	Estado do Rótulo
A	$[0, -]$	Permanente
1	$[1, A]$	Permanente
2	$[2.7, A]$	Temporário
3	$[3, A]$	Temporário
3	$[1 + 2.4, 1]$	Temporário
4	$[1 + 4, 1]$	Temporário
5	$[1 + 1.3, 1]$	Temporário

TABELA 2.3: Iteração 3.1

O vértice 3 passa a ter dois rótulos temporários, portanto remove-se o que tem a maior distância e passa-se a ter a seguinte tabela:

Vértice	Rótulo	Estado do Rótulo
A	$[0, -]$	Permanente
1	$[1, A]$	Permanente
2	$[2.7, A]$	Temporário
3	$[3, A]$	Temporário
4	$[5, 1]$	Temporário
5	$[2.3, 1]$	Temporário

TABELA 2.4: Iteração 3.2

Visto o vértice 5 ser o vértice temporário com a menor distância, o seu rótulo transforma-se em permanente e ele passa a ser o novo vértice inicial. Os seus vértices adjacentes são os vértices 4 e 7.

Vértice	Rótulo	Estado do Rótulo
A	$[0, -]$	Permanente
1	$[1, A]$	Permanente
2	$[2.7, A]$	Temporário
3	$[3, A]$	Temporário
4	$[5, 1]$	Temporário
5	$[2.3, 1]$	Permanente
4	$[2.3 + 3.7, 5]$	Temporário
7	$[2.3 + 4.3, 5]$	Temporário

TABELA 2.5: Iteração 4

O vértice temporário com rótulo com menor distância é o 2 e passa a ter rótulo permanente. Os seus vértices adjacentes são os vértices 1 e 5, que por sua vez já têm rótulos permanentes. Deste modo, visto eles não podem ser alterados, procura-se o seguinte vértice com rótulo com menor distância, altera-se o seu rótulo para permanente e considera-se como o novo vértice inicial. Esse vértice é o 3 e tem como vértices adjacentes os vértices 4 e 6. Como o vértice 4 possui dois rótulos temporários, o com maior distância é removido.

Vértice	Rótulo	Estado do Rótulo
A	$[0, -]$	Permanente
1	$[1, A]$	Permanente
2	$[2.7, A]$	Permanente
3	$[3, A]$	Permanente
4	$[5, 1]$	Temporário
5	$[2.3, 1]$	Permanente
7	$[6.6, 5]$	Temporário
4	$[6.5, 3]$	Temporário
6	$[6.9, 3]$	Temporário

TABELA 2.6: Iteração 5

Desta vez o vértice temporário com rótulo com menor distância é o vértice 4. Repetindo o mesmo processo, passamos a ter um novo vértice com rótulo temporário, o vértice B.

Vértice	Rótulo	Estado do Rótulo
A	$[0, -]$	Permanente
1	$[1, A]$	Permanente
2	$[2.7, A]$	Permanente
3	$[3, A]$	Permanente
4	$[5, 1]$	Permanente
5	$[2.3, 1]$	Permanente
7	$[6.6, 5]$	Temporário
6	$[6.9, 3]$	Temporário
B	$[8, 4]$	Temporário

TABELA 2.7: Iteração 6

Como os últimos vértices temporários que restam são os vértices 6, 7 e B, e tanto os vértices 6 e 7 apenas têm o vértice B como adjacente, que por sua vez é o último vértice, podemos avançar uma iteração e atribuir rótulos permanentes aos vértices 6 e 7.

Vértice	Rótulo	Estado do Rótulo
A	$[0, -]$	Permanente
1	$[1, A]$	Permanente
2	$[2.7, A]$	Permanente
3	$[3, A]$	Permanente
4	$[5, 1]$	Permanente
5	$[2.3, 1]$	Permanente
7	$[6.6, 5]$	Permanente
6	$[6.9, 3]$	Permanente
B	$[8, 4]$	Temporário
B	$[11.6, 7]$	Temporário
B	$[9.1, 6]$	Temporário

TABELA 2.8: Iteração 7 e 8

O rótulo de B com menor distância é o rótulo $[8, 4]$. Ou seja, o menor caminho P para ir de A até B é:

$$P = (A, 1, 4, B)\,. \tag{2.12}$$

2.2.2 Aplicação ao problema

Cada tipo de *checklist* teria um grafo associado, no qual os vértices representavam as lojas e locais de partida dos auditores, e os pesos das arestas corresponderiam ao tempo de viagem entre os vários locais. A ideia seria procurar um caminho que maximizasse o tempo de auditoria e que ao mesmo tempo tivesse o menor custo possível.

Visto que já só com essas condições o problema seria NP-Difícil [23], optou-se por utilizar uma heurística o que tornava o problema possível do ponto de vista computacional e bem mais prático de o fazer. Mas procurar o máximo do tempo de auditoria ou o mínimo do custo também teria outros constrangimentos. Primeiro, como a auditoria Geral ocupa um dia de trabalho completo, o que é o caso ideal visto termos sempre o dia preenchido com tempo de auditoria e sem desperdiçar tempo com viagens entre auditorias, ela seria sempre escolhida primeiro pelo nosso modelo, o que colocaria em causa a distribuição do tipo de auditoria e do tipo de loja. O segundo constrangimento seria o facto de, ao procurar sempre minimizar o custo das viagens, que por sua vez levava sempre a realizar as

viagens mais curtas possíveis, tornando o planeamento previsível, situação não adequada dado que as auditorias são não avisadas e o fator de imprevisibilidade da auditoria não pode ser negligenciado.

Considerando a quantidade de horas de cada *checklist*, representadas na tabela 1.1, e o *feedback* da empresa, pôde-se dividir as *checklists* em três grupos diferentes, tendo em conta quantas auditorias desse tipo, no máximo, se conseguiria realizar num dia de auditoria:

- Geral - 1 auditoria por dia,
- Específica 2/Específica 3 - 2 auditorias por dia,
- Específica 1 - 3 auditorias no mesmo dia.

O planeamento passa a ter três possíveis dias de auditoria. Caso se realize uma Geral, essa auditoria ocupa o dia todo e não se consegue realizar mais nenhuma. Como, por experiência da empresa, a única maneira de conseguirem realizar três auditorias num dia é realizando três auditorias Específica 1, tenta-se realizar sempre três auditorias Específica 1 agrupadas. Sobram as *checklists* Específica 2 e Específica 3, na qual se consegue realizar sempre uma combinação qualquer das duas *checklists* num dia de auditoria, ocupando o dia de trabalho todo.

Dada a necessidade de manter a percentagem de lojas auditadas pela distribuição geográfica relativamente uniforme, optou-se por procurar realizar uma percentagem semelhante a cada planeamento. Das 303 lojas totais, 41 são Continentes, o que implica a ida de dois auditores à mesma loja nas *checklists* Geral e Específica 2. Na tabela 2.9 encontram-se os dias necessários de auditoria para realizar cada *checklist*:

	Visitas Necessárias	Auditorias por Dia	Dias Necessários
Geral	303 + 41 = 344	1	344
Específica 1	303	3	101
Específica 2/Específica 3	(303 + 41) + 303 = 647	2	324

TABELA 2.9: Dias de auditoria necessários para cada *checklist*

A partir da tabela 2.9, podemos ver que, de forma a mantermos o número de auditorias de cada tipo equilibrado, para cada dia de auditorias Específica 1 é preciso realizar 3,41 ($\approx 344/101$) dias de auditorias Geral e 3,21 ($\approx 324/101$) dias de auditorias de Específica 2/Específica 3.

É também importante referir que o planeamento se divide em dois tipos e em duas zonas. Os tipos são o planeamento de 'viagens curtas', que são as visitas a lojas que cuja maior distância ao ponto de partida de cada auditor não é superior a duas horas, e as 'viagens longas', que são as restantes visitas a lojas em que a distância supera duas horas de viagem. A diferença é que, quando acontece uma viagem longa, os auditores costumam ficar a dormir no local e no dia seguinte realizam auditorias perto do seu local de estadia, em vez de partirem do seu sítio de partida usual. As zonas são 'Norte' e 'Sul', em que no Norte há no total 154 lojas, das quais 129 fazem parte das 'viagens curtas' e 25 das 'viagens longas', e no Sul há 149 lojas, das quais 122 são de 'viagens curtas' e 27 de 'viagens longas'. A título de simplificação, no resto do trabalho irá apenas falar-se da zona Norte do planeamento, visto o seu processo ser idêntico ao Sul, mudando apenas a informação do local e distâncias das lojas.

De forma a também tentar manter o tamanho do planeamento de certa forma constante, optou-se por organizar cada plano gerado pelo planeamento da seguinte forma: dois dias de auditorias de 'viagens longas', e sete dias de 'viagens curtas', dos quais três dias são de Geral, um dia de Específica 1 e três dias de Específica 2/Específica 3.

Outro aspeto importante é que o planeamento não indica os dias específicos em que cada auditor tem de trabalhar, ou a ordem dos dias na qual o fazer. Apenas cria um conjunto de dias de auditorias, em que os auditores depois podem livremente escolher a sua ordem de realização. Isso tem a vantagem de dar aos auditores a liberdade de organizar a sua agenda de acordo com a sua vida pessoal (têm a opção de escolher quando querem realizar dias com auditorias ou viagens mais longas), e de tornar o planeamento mais dinâmico. Assim, quando acontece algum imprevisto e não é possível realizar uma auditoria, ou algum auditor fica indisponível para trabalhar, deixa de ser necessário voltar a fazer um novo plano, visto que o conjunto de lojas não é afetado, pois não indica em que dias específicos as auditorias precisam de acontecer.

Depois de considerados todos os modelos, decidiu-se usar a abordagem com teoria de grafos para o desenvolvimento do planeamento.

Capítulo 3

Desenvolvimento do planeamento

Para o desenvolvimento da ferramenta do planeamento foi usado o software *Python* [24]. Dentro do *Python* foi usada a biblioteca *'networkx'* que permite a fácil criação e manipulação de grafos. Para desenvolvimento da interface da ferramenta foi usada a biblioteca *'tkinter'*, também existente no *Python*.

O código responsável pela ferramenta divide-se em duas partes: o código que executa o planeamento e o código da interface da ferramenta. O código que executa o planeamento está dividido em cinco funções principais que correspondem aos três grupos diferentes de *checklists* referidos anteriormente (Geral, Específica 2/Específica 3 e Específica 1), a uma função responsável pelo planeamento de viagens longas e a uma função responsável por executar todas as funções anteriores.

Por cada planeamento que se decide criar, o programa executa cada uma das funções e obtém uma combinação de lojas e as suas respetivas *checklists* para cada auditor a que se decidiu atribuir um plano de auditorias. Caso alguma *checklist* já não tenha nenhuma loja disponível para visitar, ou caso nenhuma das lojas disponíveis obedeça às restrições para ser visitada, nenhuma loja é atribuída para essa *checklist* e o auditor é notificado de tal. Nesse caso, os auditores têm a opção de fazer auditorias a outros tipos de loja para compensar os dias de auditoria que não lhes foram atribuídos.

Depois das lojas serem atribuídas, elas são retiradas de um dicionário * que contém, por *checklist*, todas as lojas que ainda falta serem auditadas. Para além disso, acrescenta

*No contexto do *Python*, dicionários são a implementação de uma estrutura de dados, mais conhecida como uma matriz associativa. Um dicionário consiste numa coleção de pares chave-valor. Cada par de chave-valor mapeia a chave para seu valor associado.

todas as lojas a um dicionário chamado 'ListaEspera' que armazena as lojas visitadas durante quinze dias úteis de trabalho de auditoria. Enquanto uma loja estiver na ListaEspera, ela encontra-se indisponível para ser escolhida para planeamentos. Ao final de quinze dias de trabalho de campo, as lojas são removidas e voltam a estar disponíveis. Isto serve para prevenir que uma loja seja escolhida sem ter realizado o tempo de espera mínimo de um mês visto que, como os auditores auditam três dias por semana, quinze dias de auditoria corresponde a um pouco mais de um mês.

A interface é a parte da ferramenta que fornece uma visualização do planeamento e torna fácil e prática a sua utilização, como por exemplo para fazer alterações manuais aos planos de auditoria, verificar que lojas estão disponíveis para serem auditadas e verificar planeamentos anteriores.

As seguintes secções vão estar divididas tanto pelas diversas funções que foram usadas para executar o programa, em que em cada secção vai estar explicado o seu desenvolvimento e evolução tendo em conta o *feedback* obtido da empresa, como também pelo código responsável pela interface da ferramenta, seguido de uma explicação sobre como funciona a sua utilização. Junto de cada função encontra-se uma pequena descrição dos parâmetros e outputs da função juntamente com o seu tipo de variável no *Python*. De forma a evitar repetição, como as funções partilham parâmetros iguais, eles apenas apenas se encontram descritos uma vez, na primeira função em que são referidos.

Na interface da ferramenta há uma opção de escolher fazer um planeamento para auditores específicos, mas no resto do relatório assume-se que foi escolhido realizar-se um planeamento para quatro auditores.

3.1 Função - Auditoria Geral

A função 'Auditoria_Geral' é a função responsável por atribuir a cada auditor as lojas escolhidas para realizar auditorias do tipo Geral. O seu código pode ser encontrado no apêndice, juntamente com um pequeno resumo da função e novamente enunciado os seguintes parâmetros e *outputs*.

```
1 def Auditoria_Geral(time_matrix, plano, LojG, todas, Auds, tipo_loja, Loj_total,
                      num_auds, dias=3, Pos_Estadia=False, Loj_Dormida=None):
```

Parâmetros:

- time_matrix : *numpy.ndarray*

  DataFrame contendo a matriz de distâncias entre todas as lojas.
- plano : *int*

  Número do plano a ser realizado.
- LojG : *list*

  Lista de lojas Geral disponíveis para o plano.
- todas : *list*

  Lista contendo as lojas disponíveis de todos os tipos de auditoria.
- Auds : *list*

  Lista de auditores para qual fazer o plano.
- tipo_loja : *dict*

  Dicionário contendo todas as lojas e o seu respetivo tipo de loja, CNT, MH ou BD.
- Loj_total : *list*

  Lista de todas as lojas no *DataFrame*.
- num_auds : *int*

  Quantidade de auditores existentes no *DataFrame*.
- dias : *int*, opcional

  Quantidade de dias para o qual se realizar o plano. O *default* é 3.
- Pos_Estadia : *bool*, opcional

  Indica se o auditor ficou a dormir no local (pós estadia) ou não. O *default* é *False*.

- Loj_Dormida : *dict*, opcional

  Dicionário contendo as lojas perto do local onde cada auditor ficou a dormir. O *default* é *None*.

Output:

- Total_Visitadas : *dict*

  Dicionário contendo como *key* os vários auditores como dicionários. Cada dicionário auditor, tem por sua vez um dicionário por cada tipo de auditoria. Além disso, cada *key* "tipo de auditoria"tem uma lista contendo as lojas visitadas para o respetivo auditor e tipo de auditoria.
- Lojas_Finais : *dict*

  Contém a loja perto do local onde cada auditor ficou a dormir.

3.1.1 Viagem curta

Começa por escolher quantas lojas vão ser escolhidas para serem auditadas. Para tal, existe um ciclo em que primeiro se define um número n de lojas a serem aleatoriamente escolhidas das lojas disponíveis. Esse valor é o mínimo entre o número de lojas disponíveis para auditoria Geral e doze, visto que no máximo visita-se $1 \times 3 \times 4$ lojas, que é no caso em que cada um dos quatro auditores visita uma loja nos três dias. Depois de serem escolhidas aleatoriamente as n lojas, é necessário verificar quantas delas são lojas CNT pois a escolha de uma loja CNT implica a sua atribuição a um segundo auditor. Portanto, por cada loja CNT escolhida, é removida uma loja MH ou BD. Também, por opção da empresa, no máximo só pode ser escolhida uma loja CNT no total em cada plano. O ciclo corre até se encontrar um conjunto de lojas que obedeça às restrições, ou, até ter corrido 100 vezes. Caso isso aconteça, reduz-se o número n em uma unidade, e volta-se a correr o ciclo. O ciclo acaba quando se escolhe um conjunto de lojas ou quando $n = 0$. Este procedimento existe para evitar que o ciclo corra infinitamente.

Após as lojas terem sido escolhidas, é criado um grafo com o apoio da biblioteca *'networkx'* em que os vértices são as lojas escolhidas e os sítios de partida de cada auditor, e os pesos das arestas são as distâncias entre os locais. De seguida, atribui cada loja ao auditor disponível mais próximo até cada auditor ter no máximo três lojas atribuídas e de forma que a soma total das distâncias dos locais de partida dos auditores às lojas seja a

menor possível. Se for uma loja CNT, ela é atribuída aos dois auditores disponíveis mais próximos.

3.1.2 Viagem longa

Caso seja o primeiro dia de viagem longa o processo de escolha é exatamente igual. Caso seja uma pós-dormida, ou seja, no segundo dia de uma viagem longa, o processo é bastante semelhante ao da viagem curta, apenas mudando em alguns detalhes. Primeiro, o local de partida passa de ser o habitual de cada auditor para ser considerado como a última loja auditada para cada auditor. Segundo, em vez de se escolher as lojas do conjunto de todas as lojas disponíveis, cria-se um ciclo para cada auditor no qual ele irá tentar atribuir uma loja diferente a cada um que esteja no máximo a 40 minutos do seu ponto de partida. Caso não encontre, acrescenta cinco minutos à distância máxima de procura e repete o ciclo. O ciclo acaba quando se encontra uma loja diferente para cada auditor ou quando se atinge os 75 minutos de distância máxima.

Outro pormenor importante de viagens longas é que lojas CNT apenas podem ser auditadas no primeiro dia, por opção da empresa. Este pormenor existe para evitar casos onde fossem necessários dois auditores para realizar uma auditoria numa loja CNT e só houvesse um auditor nas proximidades dessa loja. Desta forma evita-se que o segundo auditor tenha de fazer uma viagem excessivamente longa para realizar a auditoria à loja.

3.2 Função - Específica 1

Função responsável por atribuir a cada auditor as lojas escolhidas para realizar auditorias Específica 1.

```
def Especifica_1(Grafo, time_matrix, plano, Loj_E1, todas, Auds, tipo_loja,
2            linha_ponte, Loj_total, num_auds, Pos_Estadia=False,
             Loj_Dormida=None, ListaEspera=None):
```

Parâmetros:

- Grafo : *networkx.classes.digraph.DiGraph*

  Grafo contendo todos os vértice e ligações da matriz de distâncias exceto as lojas da ListaEspera e as lojas de Específica 1 já visitadas.

- Loj_E1 : *list*

Lista de lojas Específica 1 disponíveis para o plano.

- linha_ponte : *dict*

  Dicionário indicando se a loja pode visitar apenas lojas acima da ponte da Arrábida, apenas abaixo, ou ambas (acima e abaixo).

- ListaEspera : *dict*, opcional

  Dicionário com todas as lojas visitadas recentemente e os respetivos dias desde a última vez que foram auditadas. O *default* é *None*.

Output:

- Total_Visitadas : *dict*
- Lojas_Finais : *dict*

3.2.1 Viagem curta

Desta vez, já são escolhidas mais do que uma loja por dia para cada auditor. Então, o processo divide-se em duas partes. A primeira é a escolha da primeira loja do dia de cada auditor e a segunda parte trata da escolha da segunda e terceira loja do dia. A primeira parte é idêntica ao processo de escolha da loja da função Auditoria Geral, excluindo a atribuição de um segundo auditor a uma loja caso ela seja CNT. A segunda parte é um ciclo na qual se atribui primeiro a segunda loja a cada auditor, e depois, se possível, a terceira loja. A escolha é feita da seguinte forma: depois da primeira loja do dia ter sido selecionada, procura-se o conjunto de todas as lojas disponíveis que estão a uma distância menor de $tmax = 30$ minutos da loja selecionada e escolhe-se uma aleatoriamente desse conjunto. Caso não haja nenhuma loja a menos de $tmax$ minutos, não se atribui nenhuma loja. A escolha da terceira loja é idêntica, mas dessa vez temos $tmax = 20$ minutos. Realiza-se a segunda e terceira escolha desta forma em vez de procurar a loja imediatamente mais próxima da selecionada, pois dessa maneira o planeamento podia-se tornar previsível, o que por consequência reduzia o impacto da auditoria.

Embora a única maneira de se realizar três auditorias num único dia seja com três Específica 1, nem sempre é possível realizar três num dia. Um dos motivos para tal é, em geral, auditorias em CNT serem de maior duração do que para os restantes tipos de loja. Portanto, foram acrescentadas novas restrições. Caso seja escolhida uma loja CNT, só se podem realizar duas auditorias nesse dia, e a outra auditoria tem de ser a um MH ou a um BD. E caso já se tenha escolhido uma combinação de duas lojas MH e BD, não se pode escolher uma terceira que seja CNT.

Outro aspeto que foi necessário considerar foi a posição das lojas relativamente à ponte da Arrábida. Se as lojas estão bastante acima da ponte categorizam-se como 'acima', se estão bastante abaixo categorizam-se como 'abaixo', e, se estão relativamente perto da ponte, são categorizadas como 'ambas'. Depois foi acrescentada uma nova condição: lojas que pertencem à categoria 'acima' só podem visitar de seguida lojas pertencentes às categorias 'acima' ou 'ambas', lojas da categoria 'abaixo' só podem visitar de seguida lojas das categorias 'abaixo' ou 'ambas', e lojas que pertencem à categoria 'ambas' podem visitar de seguida qualquer loja. Este detalhe surgiu ao notar que a ferramenta recomendava emparelhamentos de auditorias que não eram realistas, embora obedecessem a todas as restrições até à altura estabelecidas, inclusive estarem a menos da distância máxima considerada. Isso acontecia porque por vezes podem existir grandes disparidades entre o tempo que é considerado necessário para realizar uma viagem e o tempo que verdadeiramente demora a realizá-la porque existem certas alturas do dia em que o trânsito atrasa bastante a travessia da ponte e os tempos considerados para as viagens eram de viagens com pouco ou nenhum trânsito.

3.2.2 Viagem longa

As diferenças entre viagem curta e viagem longa são idênticas às diferenças na função Auditoria Geral.

3.3 Função - Específica 2/Específica 3

Função responsável por atribuir a cada auditor as lojas escolhidas para realizar ou auditorias Específica 2, ou auditorias Específica 3 ou uma combinação das duas.

```
1 def Especifica_23(Grafo, time_matrix, plano, Loj_E2, Loj_E3 todas, Auds, tipo_loja,
            linha_ponte, Loj_total, num_auds, dia=3, Longe=False, Pos_Estadia=False,
                                              Loj_Dormida = None, ListaEspera = None):
```

Parâmetros:

- Loj_E2 : *list*

  Lista de lojas Específica 2 disponíveis para o plano.

- Loj_E3 : *list*

  Lista de lojas Específica 3 disponíveis para o plano.

- Longe : *bool*, opcional

  Indica se o auditor ficou a dormir no local (pós estadia) ou não. O *default* é *False*.

Output:

- comb_tot : *list*

  Lista das combinações de auditorias para cada dia.
- Total_Visitadas : *dict*
- Total_Visitadas_E2 : *list*

  Lista de todas as lojas visitadas para Específica 2.
- Total_Visitadas_E3 : *list*

  Lista de todas as lojas visitadas para Específica 3.
- Lojas_Finais : *dict*

3.3.1 Viagem curta

Esta função também se divide em duas partes, a escolha da primeira loja e a escolha da segunda loja do dia. Por sua vez, cada parte divide-se em duas hipóteses: ou atribui (i) uma auditoria de Específica 2, ou (ii) uma de Específica 3.

Inicialmente, para cada dia, começa-se por escolher a combinação de auditorias em que as possibilidades são $\{E_2E_2, E_2E_3, E_3E_2, E_3E_3\}$, onde E_2 = Específica 2 e E_3 = Específica 3. De seguida descreve-se o processo de decisão da escolha da combinação de auditorias para cada dia. Caso não haja nem lojas Específica 2 nem lojas Específica 3 disponíveis, não se atribui nenhuma e a função termina. Caso apenas não haja Específica 2, escolhe-se E_3E_3. Caso apenas não haja Específica 3, escolhe-se E_2E_2. Caso não haja lojas de Específica 2 suficientes para realizar duas auditorias de Específica 2 para cada auditor, retira-se E_2E_2 das opções e caso não haja lojas suficientes de Específica 3 para realizar duas auditorias de Específica 3 para cada auditor, retira-se E_3E_3 das opções. Depois escolhe-se uma das combinações aleatoriamente das opções disponíveis.

O processo de escolha da primeira loja para auditoria Específica 2 é bastante semelhante ao processo para auditoria Específica 3, com a diferença que, para auditoria Específica 2, a escolha de uma loja CNT implica a atribuição da loja a um segundo auditor, tal como na função de Geral. Mas, para o planeamento de auditorias Específica 2/Específica 3, existe uma condição extra. Como é preciso dois auditores para a mesma auditoria Específica 2 a uma loja CNT, eles costumam viajar até à loja num carro da empresa

partilhado, de forma a poupar gastos. Portanto, precisam de voltar ambos no mesmo carro. Então, caso a viagem para a primeira loja seja superior a 30 minutos, a segunda auditoria de cada um dos auditores tem de ser relativamente perto do local da primeira auditoria para permitir que um dos auditores consiga viajar sem o carro para a segunda loja e que a boleia no final do dia seja possível. Para um dos auditores, que vai ter acesso ao carro, a segunda viagem pode ser, no máximo, de vinte minutos. Para o outro, a viagem tem no máximo cinco minutos.

O processo de escolha da segunda loja é idêntico ao processo de escolha da segunda loja na função Específica 1, tirando que agora temos que $tmax = 20$, que não se pode escolher para a segunda loja uma loja CNT em auditoria Específica 2 e que se tem a condição extra do carro partilhado.

3.3.2 Viagem longa

As diferenças são idênticas às diferenças na função Auditoria Geral.

3.4 Função - Viagem Longa

Função responsável pelo planeamento de todas as auditorias consideradas de "viagens longas", ou seja, a uma distância maior de duas horas de qualquer auditor. O código da função pode ser encontrado no apêndice.

```
def viagem_longa(LstGrafos, time_matrix, plano, Loj_Longe, todas_longe,Aud,
2                tipo_loja, linha_ponte, Loj_total, num_aud, ListaEspera):
```

Parâmetros:

- LstGrafos : *list*

  Lista com três grafos, um para Específica 1, outro para Específica 2 e outro para Específica 3 que são usados após a escolha da primeira loja.

- Loj_Longe : *dict*

  Contém todos os tipos de auditoria como *key*, e associado a cada *key* todas as lojas disponíveis para serem visitadas a mais de duas horas.

- todas_longe : *list*

  Lista contendo as lojas a mais de duas horas disponíveis de todos os tipos de auditoria.

Output:

- planeamento : *dict*

  Dicionário contendo duas *key*s representando o 'Dia 1' e o 'Dia 2' da viagem. Cada *key* tem uma lista contendo uma lista de lojas visitadas, o tipo de auditoria, e caso o tipo seja 'Específca 23', tem uma *string* que indica a combinação das auditorias desse dia.

- ListaEspera : *dict*

  Dicionário 'Lista de Espera' atualizado.

Esta função serve para realizar os planeamentos de viagens superiores a duas horas. Os planeamentos são de dois dias, em que no segundo dia considera-se que o auditor parte da última loja auditada no dia anterior. Para tal, primeiro decide-se qual a combinação de dias de auditoria para o planeamento, e depois corre-se as devidas funções com os respetivos *inputs*.

A escolha da combinação é feita da seguinte maneira: primeiro, procura-se idealmente uma combinação na qual existam lojas disponíveis suficientes para realizar os dias completos. Depois, caso isso não seja possível, procura-se a combinação na qual se consiga realizar mais auditorias tendo em conta as lojas disponíveis.

Para procurar uma combinação ideal, começa-se com todas as combinações possíveis, $\{G/G, G/E_{23}, G/E_1, E_{23}/G, E_{23}/E_{23}, E_{23}/E_1, E_1/G, E_1/E_{23}, E_1/E_1\}$, onde E_1 = Específica 1, E_{23} = Específica 2/Específica 3, e G = Geral.

Depois, para cada tipo de dia de auditoria, verifica-se se o número de lojas disponíveis é inferior a

$$2 \times \text{nº auditorias p/ dia} \times \text{nº de auditores} \tag{3.1}$$

Se for, significa que não tem o mínimo de lojas para realizar dois dias de auditorias desse tipo. Portanto, caso se verifique, remove-se essa possibilidade. Também, caso se verifique essa condição, confirma-se se também se verifica que o número de lojas é inferior a

$$\text{nº auditorias p/ dia} \times \text{nº de auditores} \tag{3.2}$$

Caso seja, remove-se todas as possibilidades cujo primeiro dia é desse tipo de auditoria.

Depois de escolhida a combinação, resta correr as respetivas funções.

3.5 Função - Correr_Planeamento

Função responsável por executar todas as anteriores funções e definir o planeamento para cada auditor.

```
def Correr_Planeamento(num_planeamentos, Auditores=None, num_plano=0,
                       ListaEspera=None, LojAT_tot=None, LojFL_tot=None,
                       Loj_E2_tot=None, Loj_E3_tot=None):
```

Parâmetros:

- num_planeamentos : *int*

  Número de vezes que corre a função (número de planeamentos).

- Auditores : *list*, opcional

  Lista de auditores para qual se cria plano. Caso se use o valor *default* usa todos os auditores disponíveis. O *default* é *None*.

- num_plano : *int*, opcional

  Número do último plano anteriormente realizado. O *default* é 0.

- LojAT_tot : *list*, opcional

  Conjunto de lojas de Geral disponíveis. O *default* é 0.

- LojFL_tot : *list*, opcional

  Conjunto de lojas de Específica 1 disponíveis. O *default* é 0.

- Loj_E2_tot : *list*, opcional

  Conjunto de lojas de Específica 2 disponíveis. O *default* é 0.

- Loj_E3_tot : *list*, opcional

  Conjunto de lojas de Específica 3 disponíveis. O *default* é 0.

Output:

- ultimo_plano : *int*

  Número do último plano realizado no planeamento.

- ListaEspera : *dict*
- LojAT_tot : list
- LojFL_tot : list
- Loj_E2_tot : list
- Loj_E3_tot : list

- Total_Visitadas : dict

  Dicionário contendo como *key* os vários auditores como dicionários. Cada dicionário "auditor"tem respetivamente um dicionário por cada tipo de auditoria. Por sua vez, cada *key* "tipo de auditoria"tem uma lista contendo as lojas visitadas para o respetivo auditor e tipo de auditoria.

Por último, esta função serve para correr todas as funções anteriores, de forma a gerar um planeamento para o número de dias desejados.

3.6 Interface

A interface, tal como referido anteriormente, é o que possibilita a visualização do planeamento e facilita a sua criação e manuseamento. É constituída por uma janela principal (raíz) e sete janelas secundárias, às quais conseguimos aceder através de botões na janela principal.

As seguintes secções contêm uma descrição da janela principal e das janelas secundárias.

3.6.1 Janela principal

A janela principal é a que contém todas as opções essenciais para a criação e manuseamento do planeamento.

Na figura seguinte, podemos encontrar representada a janela principal da interface.

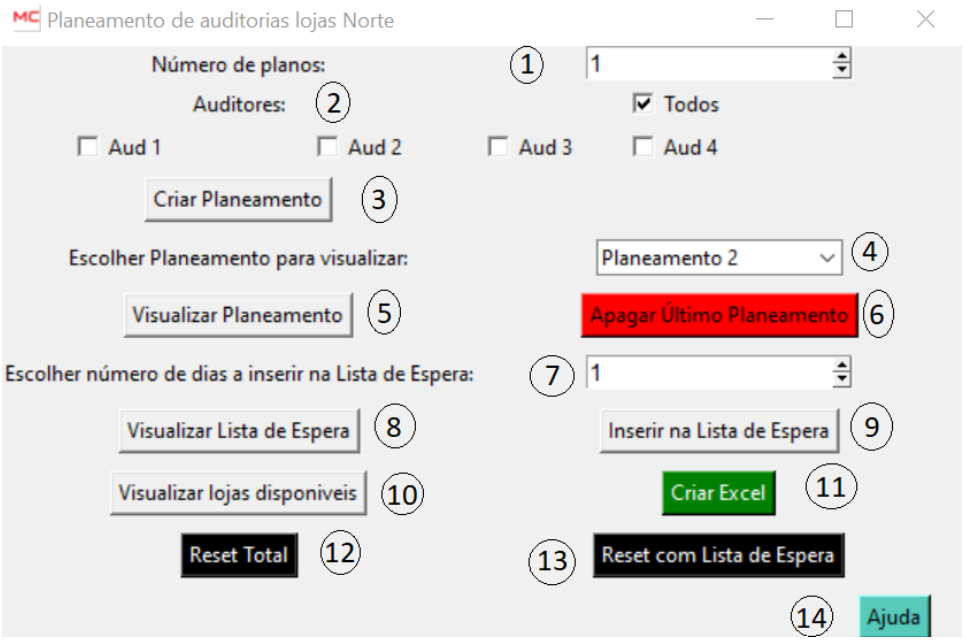


FIGURA 3.1: Janela principal da interface.

De seguida, encontra-se uma listagem das *widgets*[*] presentes na janela principal e a sua respetiva função.

- 1 - *Spinbox* onde se pode selecionar o número de planos a serem criados no planeamento.
- 2 - *Check button* para escolha do número de auditores para o qual se irá realizar o planeamento. Se o *check button* "Todos"estiver selecionado, o planeamento será realizado para todos os auditores. Os restantes *check button* "Aud 1", "Aud 2", "Aud 3"e "Aud 4"representam quatro auditores que, por questões privacidade de dados, não têm os seus respetivos nomes representados.
- 3 - Botão que, quando selecionado, executa a função Correr_Planeamento e depois representa o planeamento criado numa nova janela, como se poderá observar na figura 3.2.
- 4 - *Combo Box* onde se tem a opção de selecionar qualquer planeamento criado e guardado.
- 5 - Botão que permite visualizar o planeamento selecionado na *widget* 4.
- 6 - Botão que permite apagar o planeamento mais recentemente criado.
- 7 - *Spinbox* onde se pode selecionar o número de dias a serem acrescentados aos dias já esperados de todas as lojas na lista ListaEspera. Isto dá a opção à empresa de controlar mais facilmente essa lista de espera para caso haja a necessidade de reduzir a quantidade de dias que uma loja tem de esperar até voltar a estar disponível para ser auditada.
- 8 - Botão que permite visualizar todas as lojas presentes na ListaEspera, e o seu respetivo número de dias esperados.
- 9 - Botão que acrescenta o número de dias selecionado na *widget* 7 nas lojas de espera presentes na ListaEspera.
- 10 - Botão que permite visualizar, para cada *checklist*, todas as lojas disponíveis para serem visitadas no momento em que o planeamento irá ser realizado.
- 11 - Botão que cria um ficheiro Excel com a lista de lojas visitadas em todos os planeamentos já realizados.
- 12 - Botão que apaga toda a informação relativamente a planeamentos já realizados.

[*]Uma widget, numa interface gráfica, é um elemento de interação - tal como janelas, botões, menus, ícones, etc..

- 13 - Botão semelhante ao botão 13, mas que mantém a informação relativa à ListaEspera. Desta forma a empresa pode começar um novo ano de auditorias, com todas as lojas por fazer auditoria, e manter uma lista com todas as lojas recentemente visitadas.
- 14 - Botão que abre uma nova janela contendo um texto com informação relativamente ao manuseamento das *widgets* da janela principal.

3.6.2 Janelas secundárias

No total existem sete janelas secundárias. Uma serve para a visualização do novo planeamento criado, outra para a visualização de planeamentos antigos, duas contêm textos 'Ajuda' que servem para ajudar a esclarecer possíveis dúvidas quanto ao manuseamento do programa , outra para selecionar lojas para acrescentar manualmente de planeamentos e as restantes duas servem para visualizar as lojas e respetivos dias na lista ListaEspera e visualizar as lojas disponíveis para auditoria.

Nas figuras seguintes pode-se observar uma janela secundária contendo um exemplo de um planeamento novo, seguido de uma descrições das *widgets* presentes.

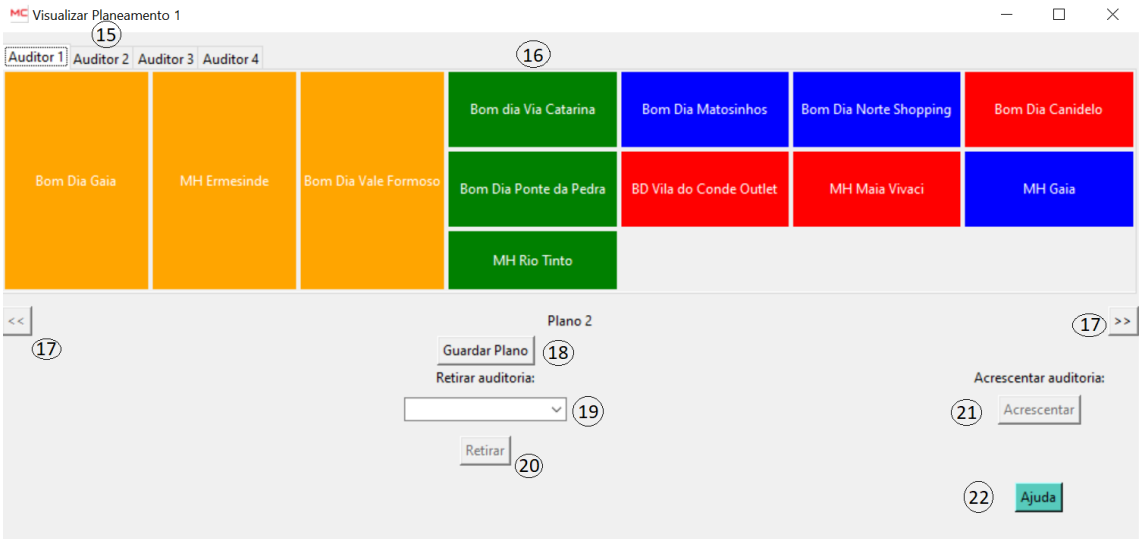


FIGURA 3.2: Janela secundária com planeamento novo - Viagem curta.

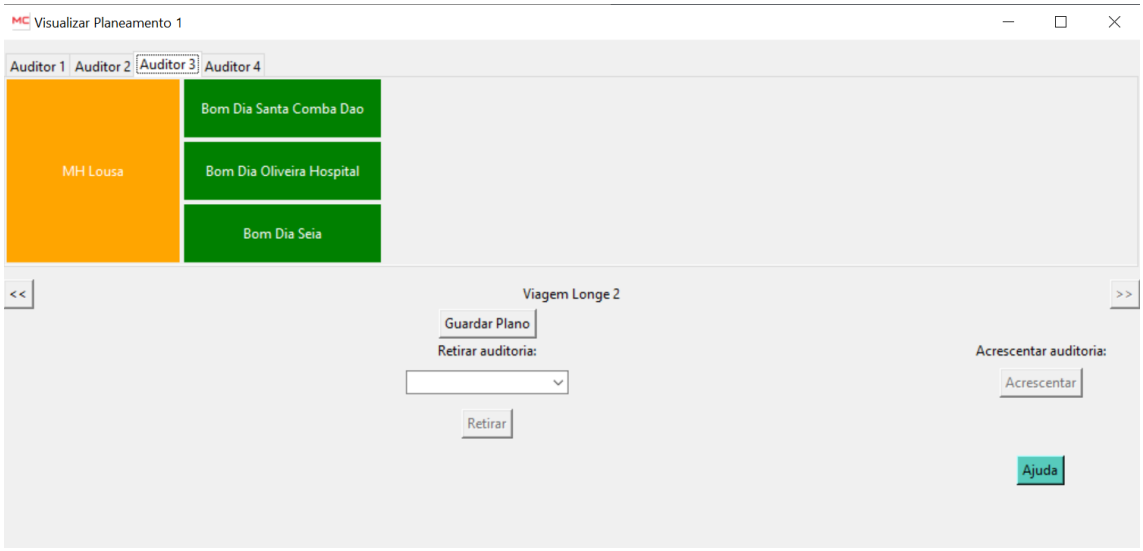


FIGURA 3.3: Janela secundária com planeamento novo - Viagem longa.

- 15 - Separadores onde se pode escolher para qual auditor queremos visualizar o planeamento.
- 16 - Blocos de texto representando cada auditoria do planeamento. No centro do bloco encontra-se a respetiva loja a ser auditada, e cada bloco tem uma cor que corresponde ao tipo de *checklist*. A cor amarela corresponde à *checklist* Geral, a verde à *checklist* Específica 1, azul à *checklist* Específica 2 e por último, a cor vermelha corresponde à *checklist* Específica 3.
- 17 - Botões que permitem trocar entre visualizar o planeamento de viagens curtas e viagens longas.
- 18 - Botão que permite guardar a informação relativa ao planeamento num ficheiro para futuras consultas e mudanças.
- 19 - *Combo Box* que permite selecionar uma auditoria do presente planeamento para retirar manualmente.
- 20 - Botão que, quando pressionado, remove a auditoria selecionada na *Combo Box* 19.
- 21 - Botão que abre uma nova janela secundária onde é possível selecionar manualmente auditorias extras a serem acrescentadas ao planeamento.
- 22 - Botão que abre uma nova janela contendo um texto com informação relativamente ao manuseamento das *widgets* da respetiva janela secundária.

3.7 Resultados

Os resultados da ferramenta são, por exemplo, o planeamento que pode ser observado nas figuras 3.2 e 3.3 através de vários blocos de cor, em que cada bloco representa uma auditoria à loja que se encontra identificada no bloco.

No caso deste exemplo, a ferramenta sugeriu, nos próximos sete dias de trabalho de campo, fazer:

- Quatro dias com uma auditoria Geral por dia às lojas Bom Dia Gaia, Modelo Ermesinde, Bom Dia Vale Formoso e Modelo Lousã.
- Dois dias com três auditorias Específica 2 por dia às lojas Bom Dia Via Catarina, Bom Dia Ponte da Pedra, Modelo Rio Tinto, Bom Dia Santa Comba Dão, Bom Dia Oliveira do Hospital e Bom Dia Seia.
- Três dias onde se realiza por dia uma auditoria Específica 2 e uma auditoria Específica 3. As lojas para auditoria Específica 2 são a Bom Dia Matosinhos, Bom Dia Norte Shopping e Modelo Gaia, e as lojas para auditoria Específica 3 são Bom Dia Vila do Conde Outlet, Modelo Maia Vivaci e Bom Dia Canidelo.

Capítulo 4

Conclusões e trabalho futuro

O modelo e ferramentas aqui apresentados foram validados e implementados pela equipa encarregada de realizar o planeamento e estão hoje em dia a ser usados. Foram principalmente testados pela equipa Sul, onde se sentiu uma redução aproximada de 50% do tempo gasto no planeamento de auditorias e onde houve um *feedback* bastante positivo relativamente à flexibilidade do planeamento.

A ferramenta pode ainda ser melhorada e complementada. Podendo ser melhorada a Lista de Espera criando um automatismo que realiza a interseção entre o plano proposto e as auditorias realizadas; ter em consideração o risco de cada loja para a priorização no planeamento e a avaliação da necessidade de auditorias extra por loja.

Apêndice A

Código - Funções Auxiliares

Neste apêndice encontram-se algumas das funções descritas no Capítulo 3. Por motivos de confidencialidade e de propriedade intelectual, nem todas as funções encontram-se no apêndice e algumas informações foram excluídas, assim como também não é revelado detalhes sobre os ficheiros de apoio que fornecem a informação usada no código sobre os auditores e as lojas. O código completo do software contém aproximadamente 5250 linhas.

As funções encontram-se documentadas visto ser uma boa prática de realizar, de forma a tornar mais fácil a sua leitura e compreensão por outros leitores. A sua documentação contém uma breve descrição da função, seguida de uma listagem dos seus parâmetros e *outputs*. Tanto o texto da documentação como os comentários encontram-se escritos sem acentos ou cedilhas por motivos de *encoding*.

A.1 Função Auditoria Geral

Código da função responsável pelo planeamento da Auditoria Geral. O código é relativamente semelhante ao código das funções de planeamento das restantes auditorias específicas. Inicialmente importa-se todas as bibliotecas necessárias e define-se todas as variáveis e dicionários a ser usados ao longo da função, tais como por exemplo dicionários que registam que lojas foram visitadas por cada auditor e dicionários que registam o tipo de cada loja.

Depois escolhe-se quantas lojas vão ser selecionadas aleatoriamente para serem auditadas do conjunto total de lojas disponíveis, tendo em conta se é um dia de auditoria pós-estadia ou não. De seguida, determina-se aleatoriamente as lojas a serem auditadas

e criam-se vários grafos, um contendo todas as lojas escolhidas, outro com todas as lojas CNT escolhidas e outro contendo as lojas MH e BD escolhidas, de forma a averiguar quais atribuir a cada auditor tendo em conta à distância entre o auditor e a loja. Depois realiza a sua atribuição tendo em conta as restrições estipuladas.

```
import random
import networkx as nx
from itertools import permutations
from funcoes.func_extra import print_output

def Geral(time_matrix, plano, LojG, todas, Auds, tipo_loja, Loj_total,
              num_auds, dias=3, Pos_Estadia=False, Loj_Dormida=None):
    """ Cria um plano para auditoria Geral

    Caso nao seja uma pos-estadia:
        Escolhe um numero n de lojas para visitar, dependendo do numero de
        auditores
    Caso seja estadia:
        V todas as lojas num raio de no maximo 75 minutos das lojas iniciais
        onde ficaram a dormir. Caso o auditor nao tenha 1 loja a menos de
        75 minutos, nao lhe e atribuido loja.
    So pode escolher no maximo 1 loja Continente(CNT) entre todas as lojas por
    todos os auditores.
    Caso seja uma loja CNT, atribuir a loja a 2 auditores
    Distribuir as n lojas pelos auditores mais perto, de forma a cada um ter
    no maximo 3 lojas

    Parameters
    ----------
    time_matrix : numpy.ndarray
        Dataframe contendo a matriz de distncias entre todas as lojas.
    plano : int
        Numero do plano a ser realizado.
    LojG : list
        Lista de lojas Geral disponiveis para o plano.
    todas : list
        Lista contendo as lojas disponiveis de todos os tipos de auditoria.
```

```
    Auds : list
        Lista de auditores para qual fazer o plano.
    tipo_loja : dict
        Dicionario contendo todas as lojas e o seu respetivo tipo de loja,
        CNT, MH ou BD.
    Loj_total : list
        Lista de todas as lojas no dataframe.
    num_auds : int
        Quantidade de auditores existentes no dataframe.
    dias : int, optional
        Quantidade de dias para o qual se realizar o plano. The default is 3.
    Pos_Estadia : bool, optional
        Indica se o auditor ficou a dormir no local(pos estadia) ou nao.
        The default is False.
    Loj_Dormida : dict, optional
        Dicionononario contendo as lojas perto do local onde cada auditor ficou
        a dormir. The default is None.

    Returns
    -------
    Total_Visitadas : dict
        Dicionario contendo como key os varios auditores como dicionarios.
        Para cada dicionario auditor, tem um dicionario por cada tipo de
        auditoria. Por sua vez, cada key tipo de auditoria tem uma lista
        contendo as lojas visitadas para o respetivo auditor e tipo de
        auditoria
    Lojas_Finais : dict
        Contem a loja perto do local onde cada auditor ficou
        a dormir

    """

    from funcoes.func_extra import saber_aud, index

    g = time_matrix

    # Criar dicionario para guardar as lojas totais visitadas para cada auditor
    Total_Visitadas = {}
```

```
    for aud in Auds:
        Total_Visitadas[aud] = []

    Lojas_Finais = {}

    # Conjunto de lojas Continente
    tot_loj_CNT = {}
    for x in tipo_loja:
        if tipo_loja[x] == 'CNT':
            tot_loj_CNT[x] = 0

    if Pos_Estadia:
        Lojas_Iniciais = list(Loj_Dormida.values())
        # Cria uma lista com os indices de cada loja, na eventualidade de
        # haver lojas iniciais repetidas, e desta forma temos o indice de
        # cada loja
        ind = index(Lojas_Iniciais)

    # Remover da lista 'todas' as lojas nao presentes em Geral
    todas_presentes = [x for x in todas if x in LojG]

    if not Pos_Estadia:
        Aud_disp = list(Auds)
    else:
        Aud_disp = list(Loj_Dormida)

    plano = plano + 1
    if Pos_Estadia:
        Loj_ini_75m = {}
        escolhida = {}
        LojG_copy = [ele for ele in todas_presentes if ele not in tot_loj_CNT]
        tot_ini_75m = set()
        tmax = 40
        check = False
        # Ver todas as lojas a menos de tmax das iniciais
        while not check:
            ok = 0
            for x in set(Lojas_Iniciais):
```

```
109                 Loj_ini_75m[x] = []
                    for loj in LojG_copy:
111                     if g[Loj_total.index(x) + num_auds][Loj_total.index(loj) +
    num_auds] < tmax:
                            Loj_ini_75m[x].append(loj)
113                         tot_ini_75m.add(loj)
                    # Remove as ja escolhidas
115                 if len(set(Loj_ini_75m[x]) - set(escolhida)) >= 1:
                        # Caso seja unica, atribui logo a loja
117                     if len(Loj_ini_75m[x]) == 1:
                            escolhida[x] = Loj_ini_75m[x]
119                     if tipo_loja[x] == 'CNT':
                            if len(Loj_ini_75m[x]) >= 2:
121                             ok += 2
                        else:
123                         ok += 1
            if ok == len(Lojas_Iniciais) and len(tot_ini_75m) >= len(
    Lojas_Iniciais):
125                 check = True
            elif tmax == 75:
127                 check = True
            else:
129                 tmax += 5


131     # Numero lojas iniciais a considerar
        n = min(dias * len(Aud_disp), len(tot_ini_75m), ok)
133     # Depois, calcular todas as distncias possiveis e escolher a menor soma
    de distncias
        if n == 1:
135         perm = []
            for x in tot_ini_75m:
137             perm.append([x])
        else:
139         # Lista das varias hipoteses de lojas para cada auditor
            perm = list(permutations(tot_ini_75m, n))
141

        dist_max = 10000
143     # Para cada permutacao, ver qual e a que resulta na menor distancia
```

```
        # Ou seja, qual a melhor atribuicao de lojas por auditor
145     for p in perm:
            dist_tot = 0
147         for lojs in range(len(p)):
                dist_tot += g[Loj_total.index(Lojas_Iniciais[lojs]) + num_auds][
    Loj_total.index(p[lojs]) + num_auds]
149         if dist_tot < dist_max:
                dist_max = dist_tot
151             melhor_perm = p
        loj_visit = set(melhor_perm)
153
    else:
155     n = min(dias * len(Aud_disp), len(LojG))  #numero lojas iniciais

157 # Criar conjunto de lojas para escolher que d mais peso a lojas menos
    escolhidas
    # Remover da lista 'todas' as lojas nao presentes em Geral
159 todas_presentes = [x for x in todas if x in LojG]

161 check = False
    contar_ciclos = 0
163 while check is False:
        contar_ciclos += 1
165     # Caso nao consiga em 100 tentativas, provavelmente nao tem comb possivel
        # E assim evita correr infinitamente o ciclo
167     # Sempre que chegar a 100 tentativas, reduz 1 auditor ao qual atribuir
        # lojas
169     if contar_ciclos == 100:
            contar_ciclos = 0
171         n -= 1
            if n == 0:
173             return([],[])
        # Ciclo para escolher lojas unicas para visitar do conjunto '
    todas_presentes'
175     if not Pos_Estadia:
            loj_visit = set()
177         while len(loj_visit) < n:
                selection = random.choice(todas_presentes)
```

```
179                 if selection not in loj_visit:
                        loj_visit.add(selection)
181         qnt_CNT = 0
            loj_MHBD = []
183         loj_CNT = []
            #Contar lojas Continente:
185         for x in loj_visit:
                if tipo_loja[x] == 'CNT':
187                 qnt_CNT += 1
                    loj_CNT.append(x)
189             elif tipo_loja[x] == 'MH' or tipo_loja[x] == 'BD':
                    loj_MHBD.append(x)
191
            retirar_qnt = 0
193
            # Caso so seja escolhido 1 auditor, nao pode escolher lojas cnt
195         if len(Aud_disp) <= 1:
                if qnt_CNT == 0:
197                 check = True
            # So pode ter 1 loja CNT por planeamento
199         elif qnt_CNT <= 2:
                check = True
201             if qnt_CNT > dias * len(Aud_disp) - n:
                    retirar_qnt = qnt_CNT - (dias * len(Aud_disp) - n)
203                 loj_visit = loj_visit - set(loj_MHBD[:retirar_qnt])
                #caso contrario, contar se ha pelo menos metade de len(Auds) em
205             #lojas MH/BD, para garantir que o ciclo consegue acabar
                #Acabar caso nao haja
207             else:
                    contar_MHBD = 0
209                 for x in LojG:
                        if tipo_loja[x] == 'MH' or tipo_loja[x] == 'BD':
211                         contar_MHBD += 1
                    if contar_MHBD < dias * len(Aud_disp)/2:
213                     continue
                if Pos_Estadia:
215                 for x in loj_CNT:
                        contar_perto = 0
```

```
                for l in Lojas_Iniciais:
                    if g[Loj_total.index(l) + num_auds][Loj_total.index(x) +
num_auds] < 30:
                        contar_perto += 1
                if contar_perto < 2:
                    check = False

# Nao queremos escolher a sorte a loja inicial do 2 dia, mas sim uma perto

#Criar um grafo com todos os vertices escolhidos
G_apoio = nx.DiGraph()
if not Pos_Estadia:
    G_apoio.add_nodes_from(Aud_disp) #adicionar os vertices/casas dos
auditores
else:
    G_apoio.add_nodes_from(Lojas_Iniciais)
G_apoio.add_nodes_from(list(loj_visit)) #adicionar as lojas escolhidas
nx.set_node_attributes(G_apoio, tipo_loja, name='loja')

edges = []
if Pos_Estadia:
    vertices_iniciais = Lojas_Iniciais
else:
    vertices_iniciais = Auds

for e_i in vertices_iniciais:
    for e_f in list(loj_visit):
        edges.append(tuple((e_i,e_f)))
G_apoio.add_edges_from(edges)

# Contam quantas lojas foram escolhidas por auditor
v_aud = [0 for i in range(len(Aud_disp))]
v_aud_CNT = [0 for i in range(len(Aud_disp))]
# Criar uma lista para guardar as lojas visitadas por auditor
G_Aud = [[] for i in range(len(Aud_disp))]

# Criar 1 grafo contendo apenas lojas continente
G_apoio_CNT = G_apoio.copy()
```

```
253     G_apoio_CNT.remove_nodes_from(loj_MHBD[retirar_qnt:])

255     # Criar 1 grafo contendo apenas lojas Modelo e Bom-Dia
        G_apoio_MHBD = G_apoio.copy()
257     G_apoio_MHBD.remove_nodes_from(loj_CNT)

259     # Atribuir agora as lojas por auditor
        while Aud_disp != []:
261         # Se nao tiver mais lojas, acabar
            if list(G_apoio_CNT.edges) == [] and list(G_apoio_MHBD.edges) == []:
263             break

265         # Atribuir as lojas CNT primeiro
            for loj in range(qnt_CNT):
267             dmax = 1000
                for a in Aud_disp:
269                 if a == "X":
                        continue
271
                    # Para cada auditor, ver a loja mais proxima das escolhidas
273                 if not Pos_Estadia:
                        path = nx.single_source_dijkstra_path(G_apoio_CNT, a, cutoff
    =120, weight="weight")
275                     for p in list(path)[1:]:
                            dist = g[saber_aud(list(path)[0])-1][Loj_total.index(p) +
     num_auds]
277                         if dist < dmax:
                                dmax = dist
279                             melhor_aresta = tuple((list(path)[0],p))
                    else:
281                     # e atribuida a loja escolhida pelo processo das permutacoes
                        melhor_aresta = tuple((a, melhor_perm[Aud_disp.index(a)]))
283
                dmax = 1000
285             # Como foi escolhida loja CNT, e preciso escolher um segundo aud
                # para a mesma loja
287             if Pos_Estadia:
                    lista_b = Aud_disp.copy()
```

```
289                 lista_b.remove(melhor_aresta[0])
            else:
291                 lista_b = list(set(Aud_disp) - set([melhor_aresta[0]]))
            for b in lista_b:
293                 if b == "X":
                        continue
295                 if not Pos_Estadia:
                        dist2 = g[saber_aud(b)-1][Loj_total.index(melhor_aresta[1]) +
 num_auds]
297                     if dist2 < dmax:
                            dmax = dist2
299                         melhor_aresta2 = tuple((b,melhor_aresta[1]))
                    else:
301                     dist2 = g[Loj_total.index(b) + num_auds][Loj_total.index(
melhor_aresta[1]) + num_auds]
                        if dist2 < dmax:
303                         dmax = dist2
                            melhor_aresta2 = tuple((b,melhor_aresta[1]))
305
            # Atribuir as lojas escolhidas as listas
307         # G_Aud - Para saber que lojas cada auditor ja visitou
            # V_aud - Para saber quantas ele ja visitou
309         G_apoio_CNT.remove_node(melhor_aresta[1])
            if not Pos_Estadia:
311             G_Aud[Aud_disp.index(melhor_aresta[0])].append(melhor_aresta)
                G_Aud[Aud_disp.index(melhor_aresta2[0])].append(melhor_aresta2)
313             v_aud_CNT[Aud_disp.index(melhor_aresta2[0])] = v_aud_CNT[Aud_disp
.index(melhor_aresta2[0])] + 1
                v_aud_CNT[Aud_disp.index(melhor_aresta[0])] = v_aud_CNT[Aud_disp.
index(melhor_aresta[0])] + 1
315         else:
                G_Aud[ind[melhor_aresta[0]][0]].append(melhor_aresta)
317             G_Aud[ind[melhor_aresta2[0]][0]].append(melhor_aresta2)
                v_aud_CNT[ind[melhor_aresta2[0]][0]] = v_aud_CNT[ind[
melhor_aresta2[0]][0]] + 1
319             v_aud_CNT[ind[melhor_aresta[0]][0]] = v_aud_CNT[ind[melhor_aresta
[0]][0]] + 1
```

```
                # atualiza a variavel 'ind', removendo o indice da ultima loja visitada
321             ind[melhor_aresta2[0]] = ind[melhor_aresta2[0]][1:]
                ind[melhor_aresta[0]] = ind[melhor_aresta[0]][1:]
323
            x = 0
325         # Caso o auditor tenha visitado 1 loja, remove-lo dos disponiveis
            for v in range(len(Aud_disp)):
327             if v_aud[v] + v_aud_CNT[v] == dias:
                    x += 1
329                 Aud_disp[v] = "X"
                if x == len(Aud_disp):
331                 Aud_disp = []
        if Pos_Estadia:
333         ind = index(Lojas_Iniciais)

335     # Atribuir as restantes lojas Modelo/Bom Dia
        for loj in loj_MHBD[retirar_qnt:]:
337         dmax = 1000
            for a in Aud_disp:
339             if a == "X":
                    continue
341             if Pos_Estadia:
                    i = Aud_disp.index(a)
343                 a = Lojas_Iniciais[i]
                path = nx.single_source_dijkstra_path(G_apoio_MHBD, a, cutoff=120, weight="weight")
345
                if not Pos_Estadia:
347                 for p in list(path)[1:]:
                        dist = g[saber_aud(list(path)[0])-1][Loj_total.index(p) + num_auds]
349                     if dist < dmax:
                            dmax = dist
351                         melhor_aresta = tuple((list(path)[0],p))
                else:
353                 for p in list(path)[1:]:
```

```
                    dist = g[Loj_total.index(list(path)[0]) + num_auds][
Loj_total.index(p) + num_auds]
355                 if dist < dmax:
                        dmax = dist
357                     melhor_aresta = tuple((list(path)[0],p))

359         if not Pos_Estadia:
                v_aud[Aud_disp.index(melhor_aresta[0])] = v_aud[Aud_disp.index(
melhor_aresta[0])] + 1
361             G_apoio_MHBD.remove_edge(melhor_aresta[0], melhor_aresta[1])
                G_apoio_MHBD.remove_node(melhor_aresta[1])
363             G_Aud[Aud_disp.index(melhor_aresta[0])].append(melhor_aresta)
  #saber qual auditor visita qual loja
        else:
365             v_aud[ind[melhor_aresta[0]][0]] = v_aud[ind[melhor_aresta[0]][0]]
 + 1
                G_apoio_MHBD.remove_edge(melhor_aresta[0], melhor_aresta[1])
367             G_apoio_MHBD.remove_node(melhor_aresta[1])
                G_Aud[ind[melhor_aresta[0]][0]].append(melhor_aresta)      #saber
 qual auditor visita qual loja
369             ind[melhor_aresta[0]] = ind[melhor_aresta[0]][1:]

371         x = 0
            for v in range(len(Aud_disp)):
373             if v_aud[v] + v_aud_CNT[v] == dias:
                    x = x + 1
375                 Aud_disp[v] = "X"
                if x == len(Aud_disp):
377                 Aud_disp = []

379     if Pos_Estadia:
            ind = index(Lojas_Iniciais)
381
if not Pos_Estadia:
383     Aud_disp = list(Auds)
else:
385     Aud_disp = list(Loj_Dormida)
```

```
    # Fazer print das lojas visitadas de cada auditor
    for aud in Auds:
        visitadas = []
        if aud in Aud_disp:
            index = Aud_disp.index(aud)
            for l in range(len((G_Aud[index]))):
                visitadas.append(G_Aud[index][l][1])
                Total_Visitadas[aud].append(G_Aud[index][l][1])
        if visitadas != []:
            print_output("O auditor", saber_aud(aud), "visita as lojas:",
    visitadas,"no plano numero", plano)
            Lojas_Finais[aud] = visitadas[-1]
        else:
            Total_Visitadas[aud].append('Outras Lojas')
            print_output('O auditor', saber_aud(aud), 'visita outras lojas no
    plano', plano)

    #Agora que atribuimos as lojas a cada auditor, precisamos de as retirar das
    disponiveis
    LojG = [ele for ele in LojG if ele not in loj_visit]   # remove a loja
    return(Total_Visitadas, Lojas_Finais)
```

A.2 Função - Viagens Longas

No código começa-se por dividir e tratar as variáveis recebidas como *input*. De seguida define-se qual a combinação de dias de auditoria a realizar nos dois dias de viagem longa, tendo em conta o número de lojas disponíveis para serem auditadas de cada tipo de auditoria. Depois de escolhido, executa-se as respetivas funções de auditoria, uma por cada dia, dependendo da combinação definida.

```
import random
import networkx as nx
from funcoes.func_extra import tirar_espera, len_exceto, len_intersecao, remover,
    print_output
```

```
5  def viagem_longa(LstGrafos, time_matrix, plano, Loj_Longe, todas_longe, Aud,
       tipo_loja, linha_ponte, Loj_total, num_aud, ListaEspera):
       """ Funcao para realizar planeamentos cujas viagens sao superiores a 2h
7
       Corre as 3 funcoes de planos mas desta vez apenas com lojas que estao a
9      2h ou mais dos pontos de partida dos auditores.

11     Parameters
       ----------
13     LstGrafos : list
           Lista com 3 grafos, 1 para Especifica 1, outro para Especifica 2 e outro
       para Especifica 3
15     time_matrix : numpy.ndarray
           Dataframe contendo a matriz de distncias entre todas as lojas.
17     plano : int
           Nmero do plano a ser realizado.
19     Loj_Longe : dict
           Contem todos os tipos de auditoria como key, e associado a cada key
21         todas as lojas disponiveis para serem visitadas a mais de 2h.
       todas_longe : list
23         Lista contendo as lojas a mais de 2h disponiveis de todos os tipos de
           auditoria.
25     Aud : list
           Lista de auditores para qual fazer o plano.
27     tipo_loja : dict
           Dicionario contendo todas as lojas e o seu respetivo tipo de loja,
29         CNT, MH ou BD.
       linha_ponte : dict
31         Dicionario indicando se a loja pode visitar apenas lojas acima da
           Linha do Douro, apenas abaixo, ou ambas (acima e abaixo).
33     Loj_total : list
           Lista de todas as lojas no dataframe.
35     num_aud: int
           Quantidade de auditores existentes no dataframe.
37     ListaEspera : dict
           Dicionario com todas as lojas visitadas recentemente e os respetivos
39         dias desde a ltima vez que foram auditadas.
```

```
    Returns
    -------
    planeamento : dict
        Dicionario contendo 2 keys representando o 'Dia 1' e o 'Dia 2' da
        viagem. Cada key e tem uma lista contendo uma lista de lojas visitadas,
        o tipo de auditoria, e caso o tipo seja 'Especifica 2/3', tem uma string
    que
        indica a combinacao das auditorias desse dia.
    ListaEspera : dict
        Dicionario 'Lista de Espera' atualizado

    """
    # Importar lojas disponiveis para cada tipo de auditoria
    Loj_G = Loj_Longe['G']
    Loj_E1 = Loj_Longe['E1']
    Loj_E2 = Loj_Longe['E2']
    Loj_E3 = Loj_Longe['E3']

    # Definir as combinacoes possiveis para os 2 dias de auditoria
    Comb_Aud = ['G/G', 'G/E23', 'G/E1',
                'E23/G', 'E23/E23', 'E23/E1',
                'E1/G', 'E1/E23', 'E1/E1']

    # Importar os grafos
    GrafoE1 = nx.DiGraph(LstGrafos[0])
    GrafosE23 = LstGrafos[1:3]

    # Retirar as lojas que estao na Lista de Espera
    Disp_G = tirar_espera(Loj_G, ListaEspera)
    Disp_E1 = tirar_espera(Loj_E1, ListaEspera)
    Disp_E2 = tirar_espera(Loj_E2, ListaEspera)
    Disp_E3 = tirar_espera(Loj_E3, ListaEspera)

    Disp_E23 = Disp_E2 + list(set(Disp_E3) - set(Disp_E2))

    # Criar dicionario que vai guardar os dados dos 2 dias
    planeamento = {}
    planeamento['Dia 1'] = []
```

```
    planeamento['Dia 2'] = []
79
    # Primeiro ver se e possivel fazer plano em que atribuimos todas as lojas
81  # necessarias para cada auditor. Caso nao seja possivel para nenhum caso,
    # depois vemos quais as melhores auditorias que podemos atribuir
83  # (dependendo do n de lojas disponiveis)
    excluir = 0
85  exc_g = 0

87  # Condicoes para Geral:
    # Primeiro ver condicoes para ver se e possivel fazer o 1 dia para todos
89  # os casos

91  # Para ser A/A, idealmente A precisa de ter pelo menos 2*num auds
    if len(Disp_G) < 2*len(Aud):
93      exc_g += 1
        Comb_Aud = remover(Comb_Aud, 'G/G')
95      # Para ser A/A ou A/E23, precisa de ter pelo menos 1*num auds
        if len(Disp_G) < len(Aud):
97          exc_g += 1
            Comb_Aud = remover(Comb_Aud, ['G/E23', 'G/E1'])
99
    # Para ser A/A, idealmente e preciso 3 auditorias para cada um dos 2 dias
101 # ou seja, pelo menos 2*3*num auditores
    if len(Disp_E1) < 2*3*len(Aud):
103     Comb_Aud = remover(Comb_Aud, ['E1/E1'])
        # Caso nao tenha para 2 dias, ver se consegue garantir pelo menos para
105     # 1 dia
        if len(Disp_E1) < 3*len(Aud):
107         excluir += 1
            Comb_Aud = remover(Comb_Aud, ['E1/E23', 'E1/G'])
109
    # Mesmo racicionio aplicado, desta vez para 2 auditorias por dia
111 if len(Disp_E23) < 2*2*len(Aud):
        Comb_Aud = remover(Comb_Aud, ['E23/E23'])
113     if len(Disp_E23) < 2*len(Aud):
            Comb_Aud = remover(Comb_Aud, ['E23/E1', 'E23/G'])
115
```

```
    # Agora precisamos de ver as condicoes para ver se da para fazer o 2 dia

117

    if len_intersecao(Disp_G, Disp_E23) < (1 + 2) * len(Aud):
119     if len_exceto(Disp_G, Disp_E23) < len(Aud):
            Comb_Aud = remover(Comb_Aud, ['E23/G'])
121     if len_exceto(Disp_E23, Disp_G) < 2 * len(Aud):
            Comb_Aud = remover(Comb_Aud, ['G/E23'])
123
    if len_intersecao(Disp_G, Disp_E1) < (1 + 3) * len(Aud):
125     if len_exceto(Disp_G, Disp_E1) < len(Aud):
            Comb_Aud = remover(Comb_Aud, ['E1/G'])
127     if len_exceto(Disp_E1, Disp_G) < 3 * len(Aud):
            Comb_Aud = remover(Comb_Aud, ['G/E1'])
129
    if len_intersecao(Disp_E1, Disp_E23) < (2 + 3) * len(Aud):
131     if len_exceto(Disp_E23, Disp_E1) < 2 * len(Aud):
            Comb_Aud = remover(Comb_Aud, ['E1/E23'])
133     if len_exceto(Disp_E1, Disp_E23) < 3 * len(Aud):
            Comb_Aud = remover(Comb_Aud, ['E23/E1'])
135


137 # Caso tenha ficado sem combinacoes ideais, ver a segunda melhor opcao
    if Comb_Aud == []:
139     Comb_Aud = ['G/G', 'G/E23', 'G/E1',
                    'E23/G', 'E23/E23', 'E23/E1',
141                 'E1/G', 'E1/E23', 'E1/E1']

143     Disp = {'G': len(Disp_G), 'E1': len(Disp_E1)/3,
                'E2': len(Disp_E2)/2, 'E3': len(Disp_E3)/2}
145
        # Ver qual tem mais lojas em relacao aos auditores
147     # Descobrir 1 loja
        mais_lojas = max(Disp, key=Disp.get)
149     if mais_lojas == 'G':
            lim1 = 0
151         lim2 = 3
        elif mais_lojas == 'E2' or mais_lojas =='E3':
153         lim1 = 3
```

```
            lim2 = 6
        elif mais_lojas == 'E1':
            lim1 = 6
            lim2 = 9
        # Descobrir 2 loja
        Disp.pop(mais_lojas)
        mais_lojas2 = max(Disp, key=Disp.get)
        if mais_lojas2 == 'G':
            lim3 = 0
        if mais_lojas2 == 'E2' or mais_lojas2 == 'E3':
            lim3 = 1
        if mais_lojas2 == 'E1':
            lim3 = 2

        Escolha1 = Comb_Aud[lim1:lim2]
        Escolha2 = Comb_Aud[lim3::3]

        # Intersecao das 2 escolhas, de forma a sabermos qual a
        # melhor combinacao
        Comb_Aud = [aud for aud in Escolha1 if aud in Escolha2]

    comb = random.choice(Comb_Aud)


    # Dia 1 e Geral
    if comb in ['G/G', 'G/E1', 'G/E23']:
        from funcoes.func_Geral import Geral
        # Correr planeamento, guardado as lojas em que terminou o dia
        Vis_G_longe, G_Finais = Geral(time_matrix, plano, Disp_G, todas_longe,
    Aud,
                                      tipo_loja, Loj_total, num_aud, 1)

        # Guardar os dados do dia
        planeamento['Dia 1'].append(Vis_G_longe)
        planeamento['Dia 1'].append('Geral')
        # Atribuir as lojas finais a uma nova variavel
        Loj_Finais = G_Finais
        # Guardar as lojas visitadas na Lista de Espera
```

```
        for aud in Vis_G_longe:
            for ele in Vis_G_longe[aud]:
                if ele not in ListaEspera:
                    ListaEspera[ele] = 0
    # Dia 1 e Especifica 1
    if comb in ['E1/E1', 'E1/E23', 'E1/G']:
        from funcoes.func_Especifica_1 import Especifica_1
        Vis_E1_longe, E1_Finais = Especifica_1(GrafoE1, time_matrix, plano,
    Disp_E1, todas_longe,
                                           Aud, tipo_loja, linha_ponte, Loj_total,
    num_aud)
        planeamento['Dia 1'].append(Vis_E1_longe)
        planeamento['Dia 1'].append('Especifica 1')
        Loj_Finais = E1_Finais
        for aud in Vis_E1_longe:
            for ele in Vis_E1_longe[aud]:
                if ele not in ListaEspera:
                    ListaEspera[ele] = 0

    # Dia 1 e Especifica 2 ou Especifica 3
    if comb in ['E23/E23', 'E23/G', 'E23/E1']:
        from funcoes.func_Especifica_23 import Especifica_23
        comb_E23, E23_longe, E2_longe, E3_longe, E23_Finais = Especifica_23(
    GrafosE23, time_matrix, plano,
                                                            Disp_E2, Disp_E3,
     todas_longe, Aud, tipo_loja,
                                                            linha_ponte,
    Loj_total, num_aud, 1, Longe=True)
        Vis_E23_longe = E2_longe + list(set(E3_longe) - set(E2_longe))
        planeamento['Dia 1'].append(E23_longe)
        planeamento['Dia 1'].append('Especifica 2')
        planeamento['Dia 1'].append(comb_E23)
        check = False
        if check:
            if E2_longe != []:
                planeamento['Dia 1'].append(E2_longe)
                planeamento['Dia 1'].append('Especifica 2')
            if E3_longe != []:
```

```
                planeamento['Dia 1'].append(E3_longe)
225             planeamento['Dia 1'].append('Especifica 3')
        Loj_Finais = E23_Finais
227     for ele in Vis_E23_longe:
            if ele not in ListaEspera:
229             ListaEspera[ele] = 0

231 # ----------------------------------

233 # Retirar lojas visitadas no 1 dia das lojas disponiveis
    Disp_G = tirar_espera(Loj_G, ListaEspera)
235 Disp_E1 = tirar_espera(Loj_E1, ListaEspera)
    Disp_E2 = tirar_espera(Loj_E2, ListaEspera)
237 Disp_E3 = tirar_espera(Loj_E3, ListaEspera)

239 # Dia 2 e Geral
    if comb in ['G/G', 'E1/G', 'E23/G']:
241     from funcoes.func_Geral import Geral
        Vis_G_longe = Geral(time_matrix, plano, Disp_G, todas_longe,
243                             Aud, tipo_loja, Loj_total, num_aud,
                                1, True, Loj_Finais)[0]
245     planeamento['Dia 2'].append(Vis_G_longe)
        planeamento['Dia 2'].append('Geral')
247     for aud in Vis_G_longe:
            for ele in Vis_G_longe[aud]:
249             if ele not in ListaEspera:
                    ListaEspera[ele] = 0
251
    # Dia 2 e Especifica 1
253 if comb in ['E1/E1', 'E23/E1', 'G/E1']:
        from funcoes.func_Especifica_1 import Especifica_1
255     Vis_E1_longe = Especifica_1(GrafoE1, time_matrix, plano, Disp_E1,
    todas_longe,
                            Aud, tipo_loja, linha_ponte, Loj_total,
257                         num_aud, True, Loj_Finais, ListaEspera)[0]
        planeamento['Dia 2'].append(Vis_E1_longe)
259     planeamento['Dia 2'].append('Especifica 1')
        for aud in Vis_E1_longe:
```

```
            for ele in Vis_E1_longe[aud]:
                if ele not in ListaEspera:
                    ListaEspera[ele] = 0

    # Dia 2 e Especifica 2 ou Especifica 3
    if comb in ['E23/E23', 'G/E23', 'E1/E23']:
        from funcoes.func_Especifica_23 import Especifica_23
        comb_E23, E23_longe, E2_longe, E3_longe = Especifica_23(GrafosE23,
    time_matrix, plano,
                                                            Disp_E2, Disp_E3,
    todas_longe, Aud, tipo_loja,
                                                            linha_ponte, Loj_total,
    num_aud, 1, True,
                                                            True, Loj_Finais,
    ListaEspera)[0:4]
        Vis_E23_longe = E2_longe + list(set(E3_longe) - set(E2_longe))
        planeamento['Dia 2'].append(E23_longe)
        planeamento['Dia 2'].append('Especifica 2')
        planeamento['Dia 2'].append(comb_E23)
        check = False
        if check:
            if E2_longe != []:
                planeamento['Dia 2'].append(E2_longe)
                planeamento['Dia 2'].append('Especifica 2')
            if E3_longe != []:
                planeamento['Dia 2'].append(E3_longe)
                planeamento['Dia 2'].append('Especifica 3')
        for ele in Vis_E23_longe:
            if ele not in ListaEspera:
                ListaEspera[ele] = 0

    return(planeamento, ListaEspera)
```

Bibliografia

[1] A.-A. M. Mohamed, "Operations research applications in audit planning and scheduling," *International Journal of Economics and Management Engineering*, vol. 9, no. 6, pp. 2056–2064, 2015. [Cited on page 5.]

[2] C. J. Chang *et al.*, "A decision support system for audit staff scheduling of multiple and large-scaled engagements," *Review of Business Information Systems (RBIS)*, vol. 6, no. 1, pp. 27–40, 2002. [Cited on page 5.]

[3] R. Rossi, S. A. Tarim, B. Hnich, S. Prestwich, and S. Karacaer, "Scheduling internal audit activities: a stochastic combinatorial optimization problem," *Journal of combinatorial optimization*, vol. 19, no. 3, pp. 325–346, 2010. [Cited on page 5.]

[4] B. Dodin, A. A. Elimam, and E. Rolland, "Tabu search in audit scheduling," *European Journal of Operational Research*, vol. 106, no. 2-3, pp. 373–392, 1998. [Cited on page 5.]

[5] E. L. Summers, "The audit staff assignment problem: a linear programming analysis," *The Accounting Review*, vol. 47, no. 3, pp. 443–453, 1972. [Cited on page 5.]

[6] B. V. Balachandran and A. A. Zoltners, "An interactive audit-staff scheduling decision support system," *Accounting Review*, pp. 801–812, 1981. [Cited on page 5.]

[7] R. C. Morey and D. A. Dittman, "Optimal timing of account audits in internal control," *Management Science*, vol. 32, no. 3, pp. 272–282, 1986. [Cited on page 5.]

[8] P. H. Burgher, "Pert and the auditor," *The Accounting Review*, vol. 39, no. 1, p. 103, 1964. [Cited on page 5.]

[9] L. V. Tavares, *Investigação operacional*. Mc Graw-Hill, 1996. [Cited on page 6.]

[10] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear programming and network flows*. John Wiley & Sons, 2008. [Cited on page 6.]

[11] E. L. Lawler, "Fast approximation algorithms for knapsack problems," *Mathematics of Operations Research*, vol. 4, no. 4, pp. 339–356, 1979. [Cited on page 7.]

[12] F. d. P. Marques, "O problema da mochila compartimentada," Ph.D. dissertation, Universidade de São Paulo, Brasil, 2000. [Cited on page 7.]

[13] G. H. Bradley, "Transformation of integer programs to knapsack problems," *Discrete mathematics*, vol. 1, no. 1, pp. 29–45, 1971. [Cited on page 7.]

[14] C. H. Papadimitriou, "On the complexity of integer programming," *Journal of the ACM (JACM)*, vol. 28, no. 4, pp. 765–768, 1981. [Cited on page 7.]

[15] M. Conforti, G. Cornuéjols, G. Zambelli *et al.*, *Integer programming*. Springer, 2014, vol. 271. [Cited on page 7.]

[16] L. Parada, C. Herrera, M. Sepúlveda, and V. Parada, "Evolution of new algorithms for the binary knapsack problem," *Natural Computing*, vol. 15, no. 1, pp. 181–193, 2016. [Cited on page 8.]

[17] J. A. Barnes and F. Harary, "Graph theory in network analysis," *Social networks*, vol. 5, no. 2, pp. 235–244, 1983. [Cited on page 12.]

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022. [Cited on page 12.]

[19] S. Irnich and G. Desaulniers, "Shortest path problems with resource constraints," in *Column generation*. Springer, 2005, pp. 33–65. [Cited on page 14.]

[20] N. A. Ojekudo and N. P. Akpan, "Anapplication of dijkstra's algorithm to shortest route problem," *IOSR Journal of Mathematics (IOSR-JM)*, vol. 13, no. 3, 2017. [Cited on page 14.]

[21] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Cited on page 14.]

[22] K. Magzhan and H. M. Jani, "A review and evaluations of shortest path algorithms," *International journal of scientific & technology research*, vol. 2, no. 6, pp. 99–104, 2013. [Cited on page 14.]

[23] V. Kann, "On the approximability of np-complete optimization problems," Ph.D. dissertation, Citeseer, 1992. [Cited on page 19.]

[24] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995. [Cited on page 23.]