Aplicação de modelos de otimização em infraestruturas aeroportuárias: Gestão operacional das chegadas

Dissertação de Mestrado

José Manuel Araújo Mendes

Mestrado em

Gestão de Empresas (MBA)

Ponta Delgada
2023

Aplicação de modelos de otimização em infraestruturas aeroportuárias: Gestão operacional das chegadas

Dissertação de Mestrado

José Manuel Araújo Mendes

Orientador

Prof. Doutor Francisco José Ferreira Silva

Dissertação submetida como requisito parcial para obtenção do grau de Mestre em Gestão de Empresas (MBA)

RESUMO

Algo que é comum em vários aeroportos é a estagnação do crescimento devido a problemas de capacidade. Esta limitação pode ser vista de uma perspetiva física, nomeadamente das pistas e terminais destas infraestruturas. O crescimento de capacidade em termos de edificações são soluções intensivas em termos de capital, e por vezes sujeito a contestação política e social. Desta forma, pode-se optar por uma solução que passe pela otimização dos recursos existentes.

O objetivo desta dissertação foi o de obter um algoritmo, baseado no *Traveling Salesman Problem*, capaz de utilizar sequências de aeronaves na chegada a um aeroporto, e fazer os cálculos necessários para obter uma sequência ótima. Esta sequência é criada com alterações de posições originais, formando uma solução onde se consegue um aumento da capacidade, utilizando restrições operacionais para ser viável em termos práticos.

Em termos de resultados, foram obtidas soluções com melhorias entre 4 e 6%, chegando a algumas conclusões, em particular a de que utilizar um critério de mudança de posição máxima pode ter melhoria, na redução dos tempos, de 5% e a de que o tempo computacional aumenta proporcionalmente com o tamanho do problema.

Palavras-chave:

- *Aircraft Landing Problem;*
- *Maximum Position Shift;*
- Métodos heurísticos de otimização;
- *Traveling Salesman Problem*.

ABSTRACT

Something that is common at many airports is growth stagnation, due to problems in capacity constraints. This limitation can be seen from a physical point of view, specifically in the runways and terminals of these infrastructures. Capacity growth in terms of buildings are capital intensive solutions, and sometimes subject to political and social controversy. In this way, a solution can be chosen that involves optimizing existing resources.

The objective of this thesis was to obtain an algorithm, based on the Traveling Salesman Problem, capable of using sequences of aircraft upon arrival at an airport, and making the necessary calculations, to obtain an optimal sequence. This sequence is produced with changes to original positions, forming a solution, where an increase in capacity is achieved, using operational restrictions to be viable in practical terms.

In terms of results, solutions were obtained with improvements between 4 and 6%, reaching some conclusions. Two in particular: using a maximum position change criterion can improve capacity and time reduction around 5%, and the computational time increases proportionally with the size of the problem.

Keywords

- *Aircraft Landing Problem;*
- Heuristic optimization methods;
- *Maximum Position Shift;*
- *Traveling Salesman Problem.*

AGRADECIMENTOS

Esta dissertação e mestrado tiveram como objetivo a continuação do desenvolvimento pessoal e profissional, sendo uma oportunidade valiosa no caminho académico.

Desta forma gostaria de agradecer a todos os envolvidos, em primeiro lugar ao Professor Francisco Silva, pela sua disponibilidade desde o primeiro momento para orientação, ajuda no encaminhamento do trabalho e abertura para as questões que foram surgindo ao longo do processo.

Agradeço ao todos os meus professores e formadores, que ao longo destes anos permitiram alcançar os meus objetivos, bem como adicionando sempre um desafio extra à mistura, de forma a ir mais longe.

Agradeço aos meus pais, que uma vez mais me apoiaram no meu processo de crescimento pessoal, sempre apoiando nas minhas decisões, mesmo nas alturas mais conturbadas.

Por fim agradeço aos meus amigos, família, colegas e a todos aqueles que tiveram influência nesta jornada académica e profissional.

ÍNDICE

ÍNDICE DE TABELAS

ÍNDICE DE FIGURAS

LISTA DE ABREVIATURAS

ALP: *Aircraft Landing Problem*

CPS: *Constrained Position Shifting*

FCFS: *First-come-first-serve*

MPS: *Maximum Position Shift*

TSP: *Traveling Salesman Problem*

CAPÍTULO I - INTRODUÇÃO

O grande determinante na movimentação de pessoas entre cidades e países são os transportes de massas, como por exemplo os ferroviários, rodoviários e marítimos. No entanto existe um que se destaca cada vez mais, o transporte aéreo. Devido à sua rapidez e eficiência, o transporte aéreo tem tido um crescimento exponencial nas últimas décadas, resultante da massificação geral deste meio, devido ao aumento da disponibilidade de lugares e empresas, que diversifica os segmentos de mercado, relativamente a preços.

No entanto, o setor aeronáutico tem como consequência das suas exigências em termos de segurança, eficiência e elevada capacidade de transporte, a sua intensidade em termos de capital, recursos humanos e infraestruturas. A parte de recursos humanos pode facilmente ser colmatada com a formação e contratação, e o capital pode ser gerado com o crescimento económico, mas as infraestruturas são algo que podem ter algumas particularidades e adversidades.

O problema das infraestruturas, nomeadamente as aeroportuárias, deve-se ao facto que muitas destas terem sido planeadas e construídas com volumes de tráfego aéreo de algumas décadas atrás. De acordo com o descrito por Zografos *et al*. (2018), aeroportos congestionados são algo que afeta diretamente o crescimento do tráfego, que por sua vez deteriora diretamente o desenvolvimento económico de determinada região. Também descrevem que mais de 30 aeroportos são expectáveis de estarem a operar a 80% da capacidade, durante três ou mais horas por dia, em 2035, o que por sua vez aumenta as disparidades entre procura e oferta, afetando negativamente a previsibilidade e pontualidade. Os autores também evidenciam que intervenções na oferta, com o objetivo de construção de nova capacidade, são soluções intensivas em termos de capital e são muitas vezes sujeitas a debates públicos e políticos.

O foco desta dissertação é o de responder à questão de como gerir as chegadas num aeroporto utilizando ferramentas de otimização matemáticas atuais, com diferentes categorias de aeronaves, tendo como objetivo o de criar soluções que aumentem a capacidade nas chegadas a um aeroporto, por meio da redução do tempo que os aviões demoram na sequência de aproximação e aterragem.

Uma das técnicas utilizadas na sequenciação é a *First Come First Served* (FCFS), onde a primeira aeronave a chegar é a primeira aterrar. No entanto, isto traz alguma ineficiência na operação, e como consequência, impactos económicos negativos.

Várias são as soluções apresentadas na bibliografia para este tipo de problema, denominado várias vezes como *Aircraft Landing Problem* (ALP). Desde modelos com integração de incertezas dinâmicas, como atrasos e reconfigurações da sequência, a algoritmos que fazem a unificação da otimização em voo, com a gestão na rolagem para os terminais e alocação no parqueamento nos aeroportos. Por outro lado, existem modelos que não são específicos à aviação, que podem ser adaptados para o problema em questão, nomeadamente o problema do caixeiro-viajante, ou *Traveling Salesman Problem* (TSP).

A descrição do TSP será abordada nos capítulos seguintes, mas resumidamente, é um modelo que prevê qual a solução ótima de sequenciação de rotas, com diferentes destinos e distâncias entre estes. Um exemplo clássico é o de um vendedor que tem de passar por diferentes cidades numa só jornada, utilizando o caminho melhor em termos de otimização de tempo.

A utilização do TSP será adaptada ao problema em causa, uma vez que a utilização direta clássica não é possível. Esta abordagem será mencionada ao longo do trabalho, bem como a utilização de metodologias de resolução necessárias, particularmente os métodos heurísticos.

Esta dissertação será composta por 5 capítulos: a presente introdução, enquadramento teórico, formulação do modelo, resultados obtidos e análise, e por fim a conclusão. No próximo capítulo serão descritos os modelos em detalhe, com especial atenção no que se refere ao TSP. introduzindo os conceitos essenciais para o desenvolvimento da dissertação.

CAPÍTULO II – ENQUADRAMENTO TEÓRICO

Como foi referenciado no capítulo anterior, a otimização em aeroportos pode ser feita por um conjunto de ferramentas e modelos matemáticos, com diferentes graus de complexidade consoante os objetivos pretendidos. Neste enquadramento teórico serão introduzidos os modelos matemáticos base e os conceitos aeronáuticos essenciais para o desenvolvimento da dissertação.

2.1 Problema do caixeiro-viajante

O problema do caixeiro-viajante, ou em inglês *Traveling Salesman Problem* (TSP), foi estudado no século XVIII por um matemático irlandês Sir William Rowan Hamilton e pelo matemático britânico Thomas Penyngton Kirkman (Matai, Singh, & Murari, 2010). No entanto, a forma geral do TSP foi introduzia por Kalr Menger em Viena e na Universidade de Harvard.

Por definição, ao ter um conjunto de cidades e distâncias, com os respetivos custos, o TSP permite encontrar a maneira mais eficiente de visitar todas as cidades e voltar ao ponto inicial, minimizando a distância total percorrida, que por sua vez reduz o custo da viagem (Matai, Singh, & Murari, 2010). Em termos de número de rotas possíveis, é dada pela equação 1:

$$\text{Número de Rotas} = \frac{(\text{Número de Cidades} - 1)!}{2} \quad (1)$$

Ou seja, segundo a equação 1, num exemplo com 4 cidades, temos 3 rotas possíveis, e num exemplo de 24 cidades existem cerca de 1,3 x 10^{22} rotas.

Em termos de classificação existem três tipos de TSP (Matai, Singh, & Murari, 2010):

- *Symmetric Traveling Salesman Problem* (sTSP): neste caso diz-se que o TSP é simétrico, ou seja, para um conjunto de cidades e distâncias entre elas, é simétrico se todas as distâncias entre as cidades são iguais às distâncias no sentido inverso, a distância entre uma cidade x e outra y, é igual à distância de y para x. Se tivermos um conjunto de cidades V = {v,…,v_n }e o conjunto das

posições das cidades A ={(r,s): r,s ∈ V}, as distâncias e as suas correspondentes distâncias inversas são sempre iguais, ou seja $d_{rs}= d_{sr}$;

- *Asymmetric Traveling Salesman Problem* (aTSP): neste caso, existe pelo menos uma distância que não é igual à sua inversa, ou seja, para pelo menos uma $d_{rs}\neq d_{sr}$;
- *Multiple Traveling Salesman Problem* (mTSP): neste problema consiste em arranjar uma solução ótima para um conjunto "m" de vendedores.

Um exemplo do TSP aplicado à aviação são as manutenções regulares das turbinas, nomeadamente as *nozzle guide vanes* (Matai, Singh, & Murari, 2010), que são componentes responsáveis pela gestão do ar à saída da câmara de combustão para as pás da turbina. O TSP é aplicado de forma a ajustar as *vanes* para ter um fluxo de ar mais uniforme, que têm como resultado a redução na vibração e gasto de combustível. Outro exemplo, neste caso do mTSP, consiste no posicionamento de estações responsáveis pela monitorização dos sistemas de navegação por satélite, *Global Navigation Satellite System* (GNSS) (Matai, Singh, & Murari, 2010).

2.2 Formulação matemática do TSP

Segundo Matai, assumindo V como um conjunto de cidades ou pontos num gráfico, definido como $V = \{v,\ldots, v_n\}$, $E =\{r,s): r,s\in V, r<s\}$ o conjunto das posições dos pontos se for um TSP simétrico, e $A =\{(r,s): r,s \in V, r\neq s\}$ se for assimétrico. Assim, define-se a matriz custo como C, onde $C=(c_{rs})$, definido no conjunto E ou A. A matriz custo pode ser por vezes, denominada como a distância entre os pontos, pois o intervalo entre dois pontos é determinante na resolução do TSP.

Seja i e j pontos pertencentes ao gráfico V, e assumindo i,j e $k \in V$, $c_{ij}\leq c_{ik} + c_{kj}$. A desigualdade de triângulos afirma que a soma de dois lados de um triângulo é sempre maior do que o terceiro lado, é algo que se verifica anteriormente. Resumindo, seja a, b e c lados de um triângulo, têm-se $a + b \geq c$. Esta condição é satisfeita quando c_{ij} é a distância mais curta entre i e j (Matai, Singh, & Murari, 2010).

Em termos de formulação matemática para programação, o TSP é relativamente simples, na maioria da literatura e segundo Mo (2010) é definida pela equação 2, onde se minimiza a função custo:

$$\min \sum c_{ij} x_{ij} \tag{2}$$

Sujeito a:

$$\sum_{j=1}^{n} x_{ij} = 1 \;\; i = 1,2, \dots n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \;\; j = 1,2, \dots n$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad S \subset \{1,2, \dots n\}, 2 \leq |S| \leq n - 2$$

$$x_{ij} \in \{0,1\}, \qquad i, j \in \{1,2, \dots n\}, \quad i \neq j$$

Como foi visto, c_{ij} é função custo ou distância entre duas cidades ou pontos num gráfico, i e j respetivamente. x_{ij} é uma variável de decisão binária, onde x_{ij}=1 significa que foi escolhida uma determinada rota i e j, e quando x_{ij}= 0, indica que esta rota não foi usada. A primeira restrição indica que só se pode sair de i uma vez, enquanto a segunda indica que se só pode chegar a j uma vez. Estas restrições indicam que têm de passar por cada cidade pelo menos uma vez, sem deixar de parte a possibilidade de retornar novamente. A terceira restrição previne a formação de rotas parciais incompletas, passando em todas as n cidades. |S| corresponde ao número de elementos no conjunto S.

Embora o TSP tenha uma formulação simples, a resolução deste pode ser difícil consoante o aumento de variáveis, como foi visto no exemplo da equação 1, onde o número de rotas possíveis respeitante a 24 cidades era de 1,3 x 10^{22}. Segundo Mo (2010) o TSP é conhecido como um problema *NP-hard*, devido à complexidade do número de rotas, como no exemplo acima. Poderá não ser possível encontrar uma solução ótima em tempo útil, ou *polynominal time*. No próximo subcapítulo irão ser abordados os métodos de resolução do TSP, recorrendo a programação matemática.

2.3 Métodos de Resolução do TSP

Existem dois tipos de resolução para o TSP, o método tradicional e o método evolutivo, e no método tradicional podem ser utilizadas duas categorias de algoritmos, nomeadamente algoritmos exatos e aproximados (Mo, 2010).

2.3.1 Utilização de algoritmos exatos

Em termos de algoritmos exatos, segundo Mo (2010), podemos utilizar três algoritmos, consoante a necessidade do problema:

1. *Cutting plane*: consiste na aplicação do método *cutting plane*, onde num modelo com duas restrições, em cada iteração são adicionadas inequações que por sua vez vão restringindo os intervalos, de forma a obter uma solução ótima inteira;
2. Programação dinâmica: neste caso a solução é obtida por um conjunto de iterações feitas com programação dinâmica;
3. *Branch-bound*: vários autores dizem que este algoritmo assemelha-se a uma árvore, onde a analogia começa na raiz, e o problema vai-se desenvolvendo ao longo de ramificações, tentado procurar uma solução no espaço de pesquisa o mais rápido possível.

Também segundo Mo (2010), este tipo de problemas apresenta as suas desvantagens. O método *cutting plane* como depende da experiência do programador, é raramente utilizado como método comum. A programação dinâmica tem como problema o aumento da complexidade com incremento do espaço de resolução, por isso esta é normalmente utilizada para problemas mais pequenos.

O algoritmo *Branch-bound* apresenta também um problema com o aumento da complexidade com o incremento das variáveis, tornando-se limitativo para problemas de larga escala. Como é possível notar, é um ponto fraco nos algoritmos exatos, ou seja, à exceção de problemas de pequena escala, torna-se inviável a utilização deste tipo de métodos, para situações com um grande número de variáveis.

Helsgaun (2000), dá um exemplo para a resolução de um TSP simétrico com 2 392 cidades, onde são necessárias 27 horas num supercomputador para a atingir uma solução exata. Como foi mencionado, isto depende do número de rotas possíveis entre as cidades, e num exemplo com 7 397 cidades existem $10^{25\ 000}$ rotas disponíveis, quando por comparação existem 10^{87} partículas elementares no universo (Helsgaun, 2000). O interesse das 7 397 cidades deve-se ao facto que, segundo o relatório de Helsgaun, era o maior TSP simétrico solucionado, enquanto o maior TSP assimétrico era de 500 000 cidades. Desta forma conclui-se que é mais fácil encontrar soluções para TSP assimétricos do que para simétricos.

Utilizando novamente as 7 397 cidades, na altura do relatório de Helsgaun, tinham sido necessários 3 a 4 anos num largo conjunto de computadores para obter a solução exata. Como é possível verificar, estes largos períodos para resolução do TSP de grande escala não são situações ideais, principalmente quando são necessárias soluções rápidas para otimizar um sistema. Logo, para um problema de larga escala, os algoritmos aproximados por métodos heurísticos, são mais utilizados para a resolução do TSP.

2.3.2 Utilização de algoritmos aproximados

Por oposição aos algoritmos exatos, os algoritmos aproximados apresentam-se como uma alternativa de resolução do TSP com maior celeridade na obtenção de soluções. Como a sua denominação indica, os algoritmos aproximados podem não apresentar a solução ótima, mas sim uma próxima, dentro de um intervalo de diferença entre a exata e obtida aceitável. Em termos de classificação, pode-se dividir em três classes (Helsgaun, 2000):

1. Algoritmo de construção de rotas (*tour construction algorithms*);
2. Algoritmo de melhoria de rotas (*tour improvement algorithms*);
3. Algoritmos compósitos (*composite algorithms*).

Os *tour construction algorithms* consistem na construção de uma rota adicionando novas cidades em cada etapa (Helsgaun, 2000), e terminam quando uma solução é encontrada. Este tipo de algoritmos encontram uma solução 10-15% dentro da ótima (Matai, Singh, & Murari, 2010).

Um exemplo dos *tour construction algortihms* é o algoritmo do vizinho mais próximo, que é em si uma das heurísticas mais simples de resolução do TSP. Começa ao selecionar uma cidade aleatória, procura depois a cidade não visitada mais próxima e cria uma rota que passe por esta. Este processo repete-se até visitar as cidades todas, terminando na cidade inicial. (Matai, Singh, & Murari, 2010). Embora simples, este processo tem o problema de que no percurso de retorno à origem, a distância pode ser significativamente maior do que as distâncias percorridas entre as cidades.

Por outro lado, os *tour improvement algorithms* consistem na melhoria das rotas previamente geradas. Estes tipos de algoritmos apresentam grande sucesso, e um dos exemplos comuns é o *2-opt* ou *3-opt*. O conceito baseia-se num conjunto de rotas anteriormente definidas, e em trocar duas ligações com outras duas, de forma a que a distância total seja mais curta, e termina quando não é possível mais melhorias (Helsgaun, 2000). A performance da heurística depende também do algoritmo de construção de rotas iniciais (Matai *et al.*, 2010). A figura 2 demonstram um exemplo de *2-opt* e 3-*opt*.

Figura 1. Exemplo de rotas entre cidades
Adaptado de Matai et al., 2010

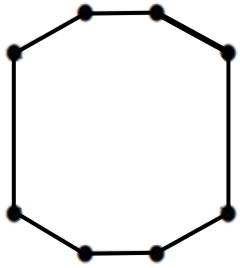
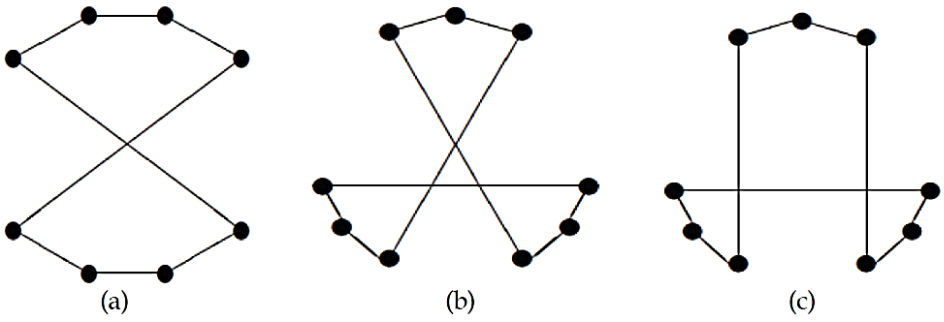


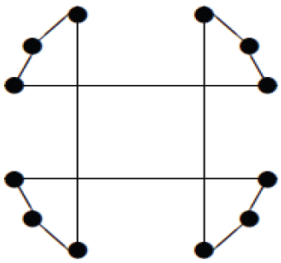
Figura 2. Exemplo de 2-opt (a), 3-opt (b) e (c)
Adaptado de Matai et al., 2010



Na figura 1 a rota entre as cidades é inicialmente traçada. O algoritmo *2-opt*, representado pela letra (a) da figura 2, reconecta duas cidades, formando duas reconexões de forma a que as rotas sejam válidas. O algoritmo é aceite se o percurso novo é mais curto do que o anterior, e é repetido até não ser possível mais melhorias. Quando terminado, a rota diz-se "dois ótima" (Matai *et al.*, 2010). As letras (b) e (c) demonstram o algoritmo *3-opt* em funcionamento, que de forma semelhante ao *2-opt*, altera três rotas em vez de duas. Também é possível verificar pela figura 2, que existem duas maneiras de ligar as cidades de forma válida alterando três rotas, que são indicadas pelas letras (b) e (c). De forma semelhante, o algoritmo termina quando não é possível melhoria. Segundo Matai e Helsgaun, uma rota três ótima é ao mesmo tempo uma rota "dois ótima".

Para casos onde existam várias alterações ao percurso, os algoritmos têm a denominação de *k-opt* ou *λ-opt* (sempre que *k* ou *λ* sejam superiores a 3). As modificações 2-*opt* e *3-opt* são casos especiais do algoritmo *λ-opt*. A figura 3 demonstra um exemplo de *4-opt*, designado de *"the crossing bridges"*. Este algoritmo não pode ser executado utilizando o *2-opt* (Matai *et al.*, 2010).

Figura 3. Exemplo de 4-*opt*
Adaptado de Matai et al., 2010



No próximo subcapítulo, será introduzido um caso especial do *λ-opt*, conhecido como o algoritmo ou heurística de Lin-Kernighan, onde existe em cada iteração uma variação do valor do *λ*.

2.3.3 Algoritmo de Lin-Kernighan

O algoritmo de Lin e Kernighan é uma das metodologias para resolução do TSP, utilizando, como foi visto, um caso do *λ-opt*. Uma vez que uma heurística é uma metodologia que permite a obtenção de resultados aproximados, também pode ser mencionado na literatura como a heurística de Lin-Kernighan.

Por definição, indica-se que a rota é *λ*-ótima se é impossível obter uma rota mais curta variando o *λ*. No entanto, com o aumento do número de cidades, o número de operações necessárias para testar todas variações em *λ* aumenta significativamente (Helsgaun, 2000). Por consequência, isto aumenta também o tempo computacional necessário para se chegar a uma solução.

Lin e Kernighan propuseram uma heurística, que colmata o tempo computacional, introduzindo um algoritmo com *λ* variável. Durante a execução do algoritmo, o valor de *λ* é alterado com cada iteração. O algoritmo em cada iteração vai aumentando o valor de *λ*, até que a combinação de nós resulte numa rota mais curta (Helsgaun, 2000).

O algoritmo original de Lin e Kernighan proposto em 1971 é eficaz, conseguindo uma probabilidade de soluções ótimas a rondar os 100% numa única tentativa, até 50 cidades. No entanto, para casos com 100 cidades, a probabilidade descia para 20 a 30% numa única tentativa, embora com vários ensaios rondava os 100% novamente (Helsgaun, 2000).

Para aumentar a eficiência, Helsgaun propôs uma modificação ao algoritmo original de Lin e Kernighan, aumentando a sua eficácia, como consequência de uma revisão das regras da heurística e das regras de pesquisa (Helsgaun, 2000).

2.4 Sequenciação da chegada de aeronaves

O problema da sequenciação de aeronaves na chegada a um aeroporto é algo que é vastamente estudado na literatura. Devido à escala, qualquer redução dos custos é multiplicada várias vezes pelo número de aviões, e resulta num impacto financeiro significativo. Ou seja, todas as aeronaves são operadas na forma mais económica possível,

obviamente dentro dos parâmetros de segurança exigidos pelos reguladores mundiais e pelos próprios passageiros.

2.4.1 Redução de custos e aumento da eficiência

Um dos métodos mais simples e mais eficaz na diminuição dos custos é a redução da distância percorrida numa viagem, pois, de forma geral, quanto menor for a distância percorrida em relação ao solo, menor será o combustível gasto numa viagem. Atendendo a que o combustível é uma das partes mais significativas dos custos variáveis de uma companhia, quanto menor o consumo de combustível, maior o retorno para a empresa.

No caso específico das chegadas a um aeroporto, isto traduz-se na sequenciação ótima dos aviões. Alterações na velocidade de cruzeiro além da ótima, ou alongamento das distâncias percorridas por via de vectorização ou esperas, são situações que provocam reduções significativas na eficiência da operação. De forma a colmatar esta situação, o controlo aéreo e as tripulações gerem parâmetros, tais como a velocidade e os tempos de chegada.

Por outro lado, e como foi mencionado na introdução, existem os constrangimentos na capacidade dos aeroportos. É do interesse de todas as entidades envolvidas, como por exemplo o gestor aeroportuário e as companhias aéreas, conseguir ter o maior número de chegadas e partidas possíveis, num menor espaço de tempo. Também é consequência do facto de que as intervenções em aeroportos são processos dispendiosos e que geram na sua maioria debates sociais. Por isso, tenta-se otimizar ao máximo as infraestruturas disponíveis de forma a serem mais rentáveis.

O próximo subcapítulo irá relacionar as massas das aeronaves com as chegadas, e quais as consequências de não as gerir de forma eficiente.

2.4.2 Gestão por classes de aeronaves

A gestão por classes nas chegadas a um aeroporto resulta do facto de que os aviões geram uma corrente de ar turbulento, quando geram sustentação (*lift*). Esta turbulência é mais proeminente nas pontas das asas, designada de turbulência de esteira, ou *wake turbulence* em inglês. Este efeito é gerado quando uma asa gera sustentação, e tem um aspeto de um

rotor, que vai sendo concebido na ponta das asas. Como a sustentação é tanto maior quanto mais massa tiver uma aeronave, maior também será este efeito.

Daí vém a necessidade de separar as aeronaves por classes de massa, pois uma aeronave maior pode gerar um efeito de turbulência de esteira suficiente para pôr em causa a segurança de uma mais pequena. Por norma, aeronaves da mesma classe têm menos probabilidade de causar turbulência de esteira que afete umas às outras, bem como aviões de classes mais leves causarem turbulência a aeronaves maiores. No entanto, ao longo dos anos foram relatadas situações de aeronaves mais pequenas que sofreram com turbulência de esteira de aeronaves maiores, causando incidentes ou mesmo acidentes. Daí a importância da separação ordenada das aeronaves.

No caso das chegadas, como a separação é mais curta comparativamente ao cruzeiro e o perfil do voo é o mesmo, são necessários critérios mínimos de separação. A tabela 1 demonstra um exemplo de separação por tempo e por classes de aeronaves:

Tabela 1. Separação de aeronaves em segundos
Adaptado de Liu et al. (2018)

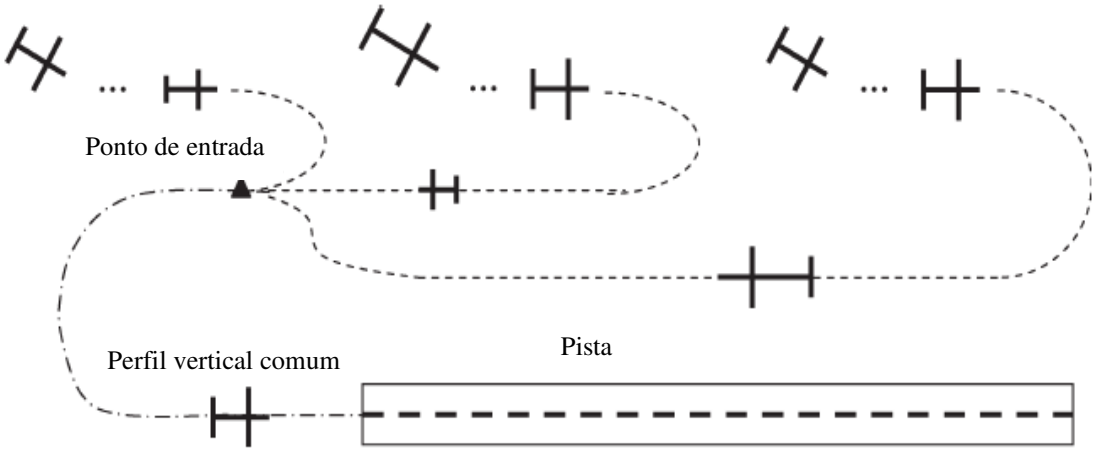
Tempos de separação (s)		**Avião Seguindo (2º)**		
		Heavy	*Large*	*Small*
Avião Frente (1º)	Grande (*Heavy)*	99	133	196
	Médio *(Large)*	74	107	131
	Pequeno *(Small)*	74	80	98

Como é possível verificar a partir da tabela 1, aviões mais pequenos requerem maior separação de aeronaves maiores, e aviões da mesma classe ou mais pequenos requerem menor separação. Esta tabela é apenas ilustrativa da separação, e será utilizada na parte matemática da dissertação como referência das distâncias temporais.

As chegadas a um aeroporto podem ser feitas por várias direções, dependendo do aeroporto de partida e da rota mais eficiente entre estes. No entanto, por norma os aviões numa pista em uso utilizam a mesma aproximação, desde um ponto de entrada, conhecido

em inglês como *Initial Approach Fix*, até à aterragem na pista. A figura 4 ilustra as chegadas a um aeroporto, demonstrando os diferentes perfis horizontais, que culmina numa aproximação com um perfil vertical comum a partir do ponto de entrada:

Figura 4. Entrada de aeronaves em aproximação a uma pista
Adaptado de Xiao-Bing & Wen-Hua (2018)



As diferentes chegadas podem ter direções que têm tipos de voo bem definidos, como por exemplo, devida à posição geográfica de um aeroporto, uma direção de onde chegam predominantemente voos internacionais e outra onde chegam voos domésticos. Como por norma os aviões que operam voos internacionais são maiores do que aqueles que operam voos domésticos, é necessário fazer a sequenciação de forma a otimizar as aproximações e evitar atrasos. Como a partir do ponto de entrada todos aviões têm um perfil comum, obedecem ao princípio de *first-come-first-serve* (FCFS), onde o primeiro a passar o ponto de entrada é o primeiro aterrar (Shone, Glazebrook, & Zografos, 2021). Desta forma, a sequenciação deverá ser feita antes do ponto de entrada na aproximação à pista, como é possível verificar na figura 4.

No próximo subcapítulo, serão analisadas as diferenças entre as chegadas *first-come-first-serve* e a otimização na sequenciação.

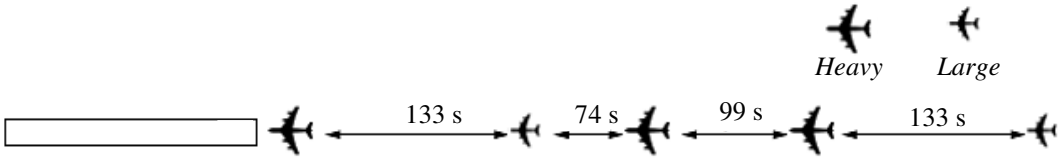
2.4.3 Gestão da sequência de aterragem

O problema na gestão das aterragens de um aeroporto, como descrito anteriormente, resulta do distanciamento e diferenciação por classes de massas dos aviões. A tabela 1, demonstra um exemplo de valores de separação temporal para as diferentes classes, e é

possível verificar que um avião pequeno seguindo um maior implica um superior espaçamento do que o inverso. Como foi mencionado no efeito de turbulência de esteira, a distância de separação é algo que é necessária, especialmente quando o avião na frente é de maior porte. Aviões de igual classe ou menor necessitam de menor espaçamento.

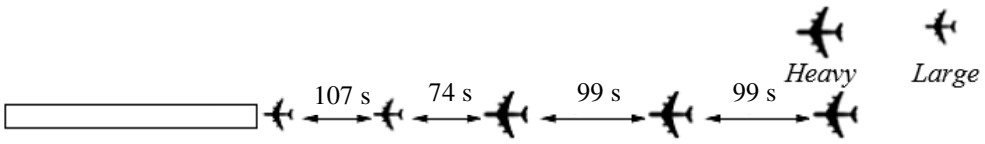
Aeroportos com menor tráfego não têm uma necessidade de otimizar a sequência de aproximações, pois geralmente existe uma margem grande na capacidade. Desta forma, pode ser implementada a sequenciação mais simples, que é simplesmente o primeiro a chegar é o primeiro a aterrar, ou como foi mencionado o FCFS. A figura 5 demonstra um exemplo desta sequência, com os valores de separação da tabela 1.

Figura 5. Aproximações pela ordem FCFS
Adaptado de Ma et.al 2019



Da figura 5 nota-se que existe uma sequência de 5 aviões, 3 *heavy* e 2 *large*, em aproximação a uma pista, sendo a sequência *heavy-large-heavy-heavy-large*. Como o segundo avião segue um avião de maior classe, verifica-se uma maior separação, bem como no quinto avião seguindo o quarto. Como princípio basilar da eficiência, quanto maior o tempo gasto na separação entre os aviões, maior a perca na capacidade e na eficácia, bem como maior o custo associado. Esta sequenciação FCFS demora um total de 439 segundos, ou 7 minutos e 19 segundos. No entanto, se tivermos como objetivo a redução no tempo gasto, alterando apenas a sequência das aeronaves, é possível um tempo total mais curto. A figura 6 demonstra esta otimização utilizando o exemplo da figura 5.

Figura 6. Aproximações pela sequência otimizada
Adaptado de Ma et.al 2019



Ao comparar a figura 6 com a 5, é possível verificar que alterando a sequência, os tempos totais ficam reduzidos. A sequência passa de *heavy-large-heavy-heavy-large* na figura 5 para *large-large-heavy-heavy-heavy*, totalizando 379 segundos, ou 6 minutos e 19 segundos. Neste caso temos uma redução em 60 segundos ou um minuto, ou seja, uma diminuição do tempo total de aproximadamente 14%. Um minuto embora não pareça um valor grande em termos de redução, quando analisando o valor da redução relativa de 14%, verifica-se que quando aplicado a um aeroporto congestionado, pode ser uma grande diferença no aumento da capacidade, utilizando um processo que é relativamente pouco dispendioso.

Ao analisar a figura 6, também se verifica que uma solução ótima, seria sequenciar os aviões de menor categoria, seguidos dos de maior. Embora seja uma solução ótima, não é algo que seja prático do ponto de vista operacional, pois ao longo de um dia de operação poderão surgir vários aviões de diferentes categorias, e não se pode estar constantemente a adiar as aterragens de aviões de maior classe, com o objetivo de aumentar a capacidade do aeroporto. Daí surge a necessidade das ferramentas de otimização e tentar chegar a um ponto de equilíbrio, onde seja possível o aumento da capacidade de um aeroporto, sem onerar as companhias que operam diferentes classes de aeronaves.

Na literatura relacionada com a otimização do problema da gestão das aterragens de um aeroporto, é assumido um desvio limitado à ordem FCFS. Segundo Luenberger, não é viável dar a um avião um tempo de aterragem muito díspar daquele do que seria no caso do FCFS, pois isto poderia envolver ajustes de velocidade e entradas em espera, o que acarreta custos elevados para os operadores aéreos.

Assim, foi introduzido o conceito de *Constrained Position Shifting* (CPS), que basicamente indica que os aviões devem ser sequenciados consoante uma restrição

denominada de *Maximum Position Shift* (MPS), dependente da ordem FCFS (Luenberger, 1988). Estas restrições têm como objetivo tornar a sequência otimizada viável do ponto de vista operacional e económico.

Estas restrições são partes preponderantes da estimativa das soluções ótimas, caso contrário seria juntar todos os aviões da mesma categoria, e ordenando das classes menores para as maiores. No próximo capítulo será utilizado o modelo matemático descrito previamente, o problema do caixeiro-viajante (TSP), bem como as ferramentas computacionais e o *software* utilizado, nomeadamente o *MATLAB*. As restrições e os valores da separação temporal da tabela 1, serão utilizados ao longo do trabalho para determinar as soluções ótimas.

CAPÍTULO III – ENQUADRAMENTO COMPUTACIONAL

Neste capítulo serão abordados os aspetos práticos da otimização, de acordo com o que foi visto na parte referente ao enquadramento teórico. Será introduzido o programa utlizado na execução da dissertação, e descrito o procedimento de adaptação dos algoritmos descritos no capítulo anterior, com o objetivo geral da redução do tempo total nas chegadas a um aeroporto.

3.1 A ferramenta computacional

Atendendo às necessidades impostas pelo problema, foi utilizado o *MATLAB* como ferramenta computacional para obtenção das soluções necessárias para a realização da dissertação. Embora existam na bibliografia outras ferramentas para resolver o problema, como por exemplo compiladores de linguagem *Fortran*, ou então programas dedicados para a resolução de variações do TSP, como o *Concorde*.

A escolha do *MATLAB* deve-se a vários fatores, principalmente à sua versatilidade, alta performance no cálculo numérico, operações com matrizes e facilidade na elaboração de código. Além disso, o *MATLAB* é extensamente utlizado em vários ramos de engenharia e aplicações nas diversas ciências.

3.1.1 A História do *MATLAB*

MATLAB foi inicialmente introduzido no final dos anos 70 e início dos 80, programado por Cleve Moler como uma ferramenta de aprendizagem para ele e os seus alunos, na altura que era professor na Universidade do Novo México em Albuquerque, Estados Unidos da América (Moler & Little, 2020). O projeto era puramente académico, sem plano de negócio formal.

O nome *MATLAB* provém de *Matrix Laboratory*, onde o cálculo matricial era a função primária originalmente. O primeiro *MATLAB* não era uma linguagem de programação, mas sim um calculador de matrizes interativo. O programa não contava com gráficos e *toolboxes*, e tinha apenas 71 funções. Quando era necessária uma

funcionalidade nova, era requisitado a Moler o código do programa, e o utilizador era responsável por criar uma sub-rotina em *Fortran*, que era a linguagem original do *MATLAB* (Moler & Little, 2020).

No inverno de 1979, Moler foi professor na Universidade de Stanford, e introduziu o *MATLAB* aos seus alunos, pois devido às suas áreas de estudo, o programa demonstrou-se imediatamente útil. Jack Little era um aluno de engenharia em Stanford, onde um amigo que era aluno de Moler apresentou-lhe o *MATLAB*, que de imediato começou a utilizar na sua especialidade académica (Moler & Little, 2020).

Em 1983, Jack Little sugeriu a introdução do *MATLAB* como produto comercial para o IBM PC (*International Business Machines Personal Computer*), que embora Moler tenha achado uma boa ideia, não se juntou a Little inicialmente. As primeiras versões do IBM PC não tinham capacidade de computação para executar o *MATLAB*, mas devido ao seu *hardware*, Little previu que existiriam evoluções do computador com capacidade para rodar o programa (Moler & Little, 2020).

Com o apoio de Moler, Little durante ano e meio criou uma versão do *MATLAB* escrita com novo código na linguagem C, relembrando que a versão original era em *Fortran*. Um amigo de Little, Steve Bangert juntou-se ao projeto, onde foi responsável por parte do compilador e interpretador (Moler & Little, 2020). Little teve a seu cargo a escrita das bibliotecas matemáticas, traduções de *Fortran* para C, do programa original e também da criação da primeira ferramenta de controlo de sistema.

A MathWorks, que ainda é a responsável pelo *MATLAB*, foi criada em 7 de dezembro de 1984 por Little, Bangert e Moler. Na semana seguinte, o *MATLAB* fez a sua estreia na 23ª conferência do Instituto de Engenheiros Elétricos e Eletrónicos (Moler & Little, 2020). Em 1985 foi feita a primeira venda do programa a Nicholas Trefethen do Instituto de Tecnologia de Massachussets (MIT). Em 1987 o *MATLAB* ficou disponível para o *Macintosh* da Apple.

O *MATLAB* para o IBM PC, ou designado *PC-MATLAB*, teve várias alterações na escrita da programação, como consequência da passagem da linguagem *Fortran* para C, bem como alterações aos operadores matemáticos nos algoritmos. Também foram introduzidas novas funcionalidades, como funções novas, gráficos e *toolboxes*.

A introdução de *toolboxes* teve um papel preponderante no crescimento do *MATLAB*, pois permitiu a criação de ferramentas dedicadas a cada área de pesquisa, com os pareceres dos diversos utilizadores. A primeira que foi criada era simplesmente *MATLAB Toolbox*, e Little, sendo um engenheiro de controlo e sinais, foi responsável pela criação da *toolbox* de controlo de sistemas, em 1985, e em 1987 pela *toolbox* de processamento de sinais (Moler & Little, 2020). Atualmente existem várias *toolboxes*, de diferentes áreas, como por exemplo *toolboxes* de finanças, otimização, inteligência artificial, cálculo simbólico e produção de relatórios.

Por outro lado, a capacidade de criação de gráficos também é um ponto forte do *MATLAB*, onde são introduzidos valores provenientes de vários *softwares* e programas, e é responsável pela produção de gráficos para relatórios técnicos (Moler & Little, 2020).

Ao longo dos anos o *MATLAB* foi recebendo informações dos seus utilizadores sobre as necessidades de novas funcionalidades, que por sua vez foram implementadas a partir de pedidos cada vez mais formais. Em 1995 foi lançado o primeiro compilador em linguagem C para *MATLAB*, que têm como função análise de tipo, seja de variáveis, quantidades, ou outra tipologia (Moler & Little, 2020). A versão *desktop* foi lançada em 2000, que é a base da disposição das janelas, comandos e variáveis que são utilizadas hoje em dia.

Atualmente, a MathWorks conta com mais de 5 000 funcionários, e em 2018 teve lucros de cerca de mil milhões de dólares americanos, de clientes de mais de 185 países. Conta com mais de cem mil clientes comerciais, académicos e governamentais, e os softwares da MathWorks são utilizados em mais de 6 500 instituições académicas (Moler & Little, 2020).

3.1.2 Funcionamento do *MATLAB*

O *MATLAB,* como foi visto no capítulo anterior, tem como uma das funções principais o cálculo e operações com matrizes. Existem também vários programas que conseguem desempenhar cálculos matriciais, como por exemplo o *Excel* da Microsoft e o *Sheets* da Google. No entanto, ambos os *Sheets* e o *Excel* são programas baseado em folhas de

cálculo, e embora sejam versáteis, em comparação, o *MATLAB* tem como base uma linguagem de alto nível, para desenvolvimento de algoritmos.

Em termos de programação, a sintaxe da linguagem em *MATLAB* assemelha-se à escrita em linguagem C, e o próprio *MATLAB* está escrito em C e C++, como foi visto anteriormente. No entanto, a linguagem C tem utilizações muito mais generalistas, como por exemplo a criação de vários tipos de programas, com diferentes finalidades, enquanto o *MATLAB* destina-se a utilizações mais especificas, nomeadamente no cálculo numérico de diferentes áreas das ciências e engenharias.

Existem várias formas de usar o programa, por exemplo como simples calculadora, que colocando uma operação matemática na janela de comando ele devolve o resultado. Também na janela de comando é possível a introdução de variáveis temporárias, ou visualização de valores provenientes de *scripts*. Por outro lado, permite a utilização de funções prévias, como por exemplo, gerar matrizes de diferentes dimensões, obedecendo a diferentes regras implícitas anteriormente. Na janela de comando é possível a abertura de *scripts* para sua edição.

No editor do *MATLAB*, é possível a escrita e alteração de *scripts*. Estes *scripts* são ficheiros que contêm um conjunto de comandos e funções, escritos em linguagem *MATLAB*. A produção de *scripts* é bastante útil, pois permite a criação de algoritmos específicos na resolução de determinada tarefa. Por exemplo, o algoritmo de resolução desta dissertação, foi executado com recurso a um *script* elaborado de raiz, com o objetivo de otimização das chegadas de aeronaves a um aeroporto.

A escrita de *scripts* é relativamente simples, pois os comandos e a introdução de variáveis são parecidos com a linguagem C, embora com algumas diferenças. Na introdução de variáveis apenas é preciso designar o nome da variável, e corresponder ao seu valor. Na introdução de variáveis é possível também fazer a geração das mesmas a partir de comandos. Por exemplo, a criação de uma variável, como uma matriz gerada aleatoriamente, obedecendo a um determinado conjunto de restrições. Este exemplo é utilizado várias vezes no algoritmo da dissertação.

De forma análoga à linguagem C, o *MATLAB* permite a criação de ciclos ou *loops*, nomeadamente os ciclos *for* e *while*. Nestes tipos de ciclos, é repetido um conjunto de

comandos até que a condição seja satisfeita, onde no caso do ciclo *for* é normalmente utilizado para operações mais simples. Outro comando bastante utlizado é o *if*, em português "se", e neste caso é executado um comando caso a condição indicada seja verdadeira ou falsa. Ou seja, para uma determinada condição, chegar à conclusão que é verdadeira, o comando escrito no *if* é executado, por outro lado o *else* é executado se a condição for falsa, e o *else if* é executado caso a condição do *if* seja falsa e a deste bloco seja verdadeira.

Também é possível a criação de comentários, os quais não são executados pelo programa. O símbolo da percentagem (%) é inserido para introduzir comentários, os quais são importantes, pois permitem a terceiros perceberem mais rapidamente o algoritmo e a forma como o autor do *script* fez a programação do mesmo. Outra função que é utilizada é a apresentação de texto, a qual aparece na janela de comando depois de executado o *script*, e permite a apresentação de uma solução neste caso.

Neste subcapítulo foram apresentadas algumas funções básicas do *MATLAB*, que foram utilizadas na produção do algoritmo da dissertação, a qual vai ser apresentado no próximo subcapítulo. No entanto, existem muitas outras funcionalidades do programa que não foram utilizadas no *script* deste problema.

3.2 Algoritmo de resolução

O algoritmo de resolução é parte preponderante neste trabalho, e vai ser descrito ao longo deste subcapítulo. Para facilitar a explicação, o algoritmo vai ser divido em quatro partes: a introdução de parâmetros e categorização, o processo heurístico de otimização´, adaptação do algoritmo e a heurística do algoritmo da dissertação.

3.2.1 Introdução de parâmetros e categorização

O algoritmo começa com um conjunto de funções base, como a eliminação de variáveis na memória, e fecho de janelas, e começo da contagem do tempo de execução do programa.

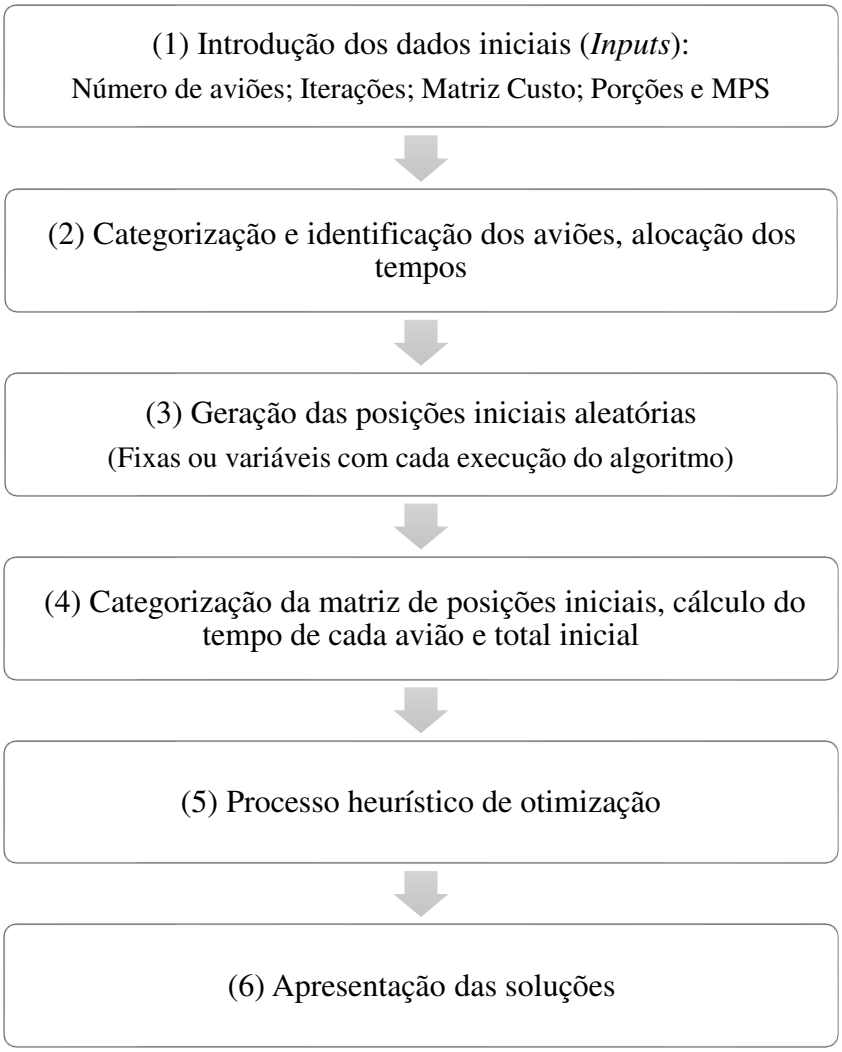
De seguida, começa a introdução de variáveis, ou *inputs*. A introdução de variáveis começa com os seguintes elementos:

- Número de aviões;
- Número de iterações;
- Matriz custo com os tempos entre categorias de aeronaves, como no exemplo da tabela 1;
- Porções parciais de cada categoria de aeronave;
- *Maximum Position Shift* (MPS).

Estas cinco variáveis de introdução, são parâmetros que podem ser alterados para a obtenção de resultados diferentes, em cada execução do programa. Por exemplo, alteração do número de aviões, iterações e MPS são as que mais frequentemente são modificados. Relembro que o *Maximum Position Shift* significa a mudança máxima de posição permitida, ou seja, se o MPS for de 2, apenas podemos mudar, no máximo, os aviões duas posições. Como foi visto, o tempo ideal seria sempre alocar as primeiras posições a aviões mais pequenos, de seguida os médios e por fim os *heavy*. No entanto, não é viável passar para as últimas posições os aviões *large* e *heavy*, de todas as vezes que chega um avião *small*, daí a existência do MPS. O MPS também tem um papel fundamental na execução do processo heurístico.

Depois da introdução das variáveis, vem a categorização dos aviões e alocação dos dados de aviões introduzidos anteriormente, em matrizes, para começo do processo de otimização. A figura 7 explica todo o processo da introdução de parâmetros e categorização, onde o primeiro passo corresponde à introdução de variáveis descrita anteriormente, e o segundo é então a categorização dos aviões e dos tempos correspondentes á matriz custo inicial.

Figura 7. Fluxograma da introdução de parâmetros, categorização e apresentação de soluções



O segundo passo do fluxograma da figura 7, como o nome indica, é a categorização e identificação dos aviões, onde é feito o cálculo das quantidades parciais de cada categoria de avião, do número de aviões total e porções parciais, por exemplo se tivermos 100 aviões e 20% forem *heavy*, temos 20 aviões *heavy*. É nesta parte que é feito o cálculo e introdução de uma variável própria para cada categoria. Depois, é extraído da matriz custo cada valor de tempo de separação para cada categoria, embora também seja possível inseri-los manualmente para cada categoria, nas variáveis respetivas. Por fim, é gerada uma matriz de aviões, que serve apenas para a identificação de cada aeronave, pela seguinte ordem: todos os aviões *small* primeiro, de seguida os *large* e os *heavy* no fim.

Exemplificando, se tivermos 30 aviões com 10 de cada categoria, teremos a matriz de identificação com a seguinte ordem: 10 *small*, 10 *large* e 10 *heavy*. Devido a esta ordem, esta matriz representa o tempo mínimo utópico que é possível com determinada combinação de aviões.

O terceiro passo envolve a geração das posições iniciais aleatórias. É gerado um vetor com números de 1 até ao número total de aviões, que são permutados aleatoriamente. Ou seja, a ordem dos números é aleatória, por exemplo, um caso com 5 aviões, no vetor de posições aleatórias, podemos ter a seguinte ordem: 2,4,5,1,3. Este vetor representa as posições iniciais aleatórias, que depois é inserido na matriz inicial de aviões do segundo passo, gerando uma matriz de aviões com posições aleatórias, e não a ordem descrita anteriormente no segundo passo. Esta matriz de aviões com posições aleatórias é considerada a matriz inicial em termos de cálculo de tempo. Exemplificando, se tivermos 5 aviões, *Small, Small, Large, Large* e *Heavy*, pegando no exemplo do vetor de posições, obtemos a matriz de aviões inicial aleatória de: *Small, Large, Heavy, Small, Large*.

O quarto passo envolve a categorização dos aviões e contabilização dos tempos totais e parciais. Da matriz de posições iniciais aleatórias, cada aeronave é categorizada consoante o seu tipo, das categorias *Small, Large* e *Heavy.* De seguida é criada uma matriz que contém as aeronaves iniciais e seguintes, que depois é utilizada para gerar outra matriz com os tempos de separação respetivos. Dessa matriz de tempos parciais de separação são somados todos os valores, que resulta no tempo total inicial, que é utilizado como termo de comparação no final do algoritmo.

O quinto passo é o processo heurístico de otimização, e para efeitos de simplificação, será explicado nos subcapítulos seguintes.

O sexto e último passo envolve os resultados obtidos no processo de otimização, bem como a apresentação das soluções. Inicialmente é categorizada a matriz com os aviões obtidos no final do processo heurístico, e de seguida é calculada a diferença entre os tempos finais e obtidos, que resulta na redução de tempo, e a otimização total resulta da seguinte equação 3:

$$\text{Otimização} = \frac{-(\text{Tempo Total Final} - \text{Tempo Total Inicial})}{\text{Tempo Total Final}} * 100 \qquad (3)$$

Depois são apresentados na janela de comando do *MATLAB* os seguintes parâmetros:

- Número de aviões *small*;
- Número de aviões *large;*
- Número de aviões *heavy*;
- Tempo total inicial;
- Tempo total final;
- Resultado da melhoria em percentagem da equação 3;
- Tempo computacional de execução do algoritmo.

O sinal negativo na equação 3 antes da diferença entres os tempos totais final e inicial, deve-se ao facto de que o tempo total final é sempre inferior, ou no máximo igual, ao tempo total inicial, por isso o sinal negativo previne que a otimização seja um valor negativo.

3.2.2 Processo heurístico de otimização

O processo de otimização é a peça central do algoritmo, uma vez que é este que é responsável pela produção das soluções necessárias. Este processo embora simples, requer uma sequência lógica para não afetar os resultados obtidos.

Como foi descrito anteriormente, o *MATLAB* possui *toolboxes*, e uma das quais é específica para otimização. A *optimization toolbox* é uma ferramenta versátil, que possui inseridos vários algoritmos de resolução, como por exemplo *o mixed-integer linear programming* (MILP), que é utlizado em exemplos de chegadas em aeroportos. No entanto, para efeitos de resolução do TSP por métodos heurísticos e desafio acrescido, foi criado de raiz um algoritmo específico para esta dissertação.

O processo de otimização tem por base o *Traveling Salesman Problem*, que foi descrito no capítulo II. Embora não possa ser utilizado como os exemplos clássicos, pois os TSP simples têm por base cidades, com as suas respetivas distâncias, em que se tenta minimizar a distância total a ser percorrida pelo viajante. Também como foi visto no

segundo capítulo, existem TSP simétricos e assimétricos, e no caso simétrico as distâncias entre quaisquer cidades x e y, no conjunto de cidades, é igual à distância entre y e x. A tabela 2 demonstra um exemplo de um TSP simétrico.

Tabela 2. Exemplo de distâncias, em km, entre cidades para um TSP simétrico

Cidades	**A**	**B**	**C**	**D**	**E**	**F**	**G**
A	-	94	49	48	33	90	37
B	94	-	39	24	40	9	14
C	49	39	-	26	23	35	82
D	48	24	26	-	73	65	45
E	33	40	23	73	-	18	37
F	90	9	35	65	18	-	44
G	37	14	82	45	37	44	-

Da tabela 2 é possível verificar que para qualquer uma das 7 cidades, a distância entre duas cidades é sempre igual à distância do percurso inverso entre elas, por exemplo a distância de A a C é igual a de C a A, a de D a G é também a mesma que a de G a D, entre outras. O exemplo desta tabela foi gerado com recurso à função de gerar matrizes aleatórias no *MATLAB*.

Analisando a tabela 1 apuramos que ser for aplicada ao TSP, estamos perante um caso assimétrico, por oposição ao caso da tabela 2. Exemplificando, o tempo de um avião *heavy* seguindo de um *small* é de 74 segundos, mas de um *small* sucessivo de *heavy* é de 196 segundos.

Ou seja, é possível fazer a adaptação do TSP, a um caso assimétrico, onde em vez de temos distâncias entre cidades, onde o objetivo é a redução da distância total percorrida, possuímos tempos entre aeronaves, e a meta é reduzir o tempo total nas chegadas a um aeroporto, ou então antecipação da hora de chegada do último avião na sequência.

Pegando na equação 2 do segundo capítulo e adaptando a este caso em específico, temos o objetivo da redução da função custo, que é neste caso a redução do tempo total.

Em termos de restrições, temos de ter alocadas todas aeronaves, formando uma sequência completa com todas as aeronaves iniciais.

Por outro lado, comparando novamente as tabelas 1 e 2, verificamos que para cada cidade corresponde uma e apenas uma distância para outra cidade, ou seja, no problema temos casos únicos em termos de distância. No caso da sequenciação temos três categorias de aviões, e vários aviões das mesmas categorias, com tempos iguais, à exceção do caso com três aviões, cada um da sua categoria, que nesta situação específica é *Small*, *Large* e *Heavy*.

Desta forma, em vez de termos uma matriz de distâncias que utilizamos para minimizar, utilizamos uma matriz com as categorias dos aviões, cujo as entradas contêm os tempos para todos os aviões, (matriz de separação temporal correspondente à tabela 1). Esta matriz foi vista anteriormente na figura 7 no segundo passo. Assim, tendo a matriz de aviões, contendo os tempos de separação entre eles, é possível iniciar o processo de otimização.

Portanto, embora não seja possível a utilização direta do TSP, é viável a adaptação deste ao problema em causa, extraindo as formulações, princípios e conhecimento da literatura referente ao TSP.

Como foi exposto no segundo capítulo, existem várias formas de resolução do TSP, nomeadamente algoritmos exatos, como o *branch-bound*, e algoritmos aproximados como a heurística de Lin-Kernighan. Os algoritmos utilizados na resolução destes problemas foram algoritmos aproximados, utilizando como aproximação a heurística do TSP na classe dos *tour improvement algorithms* (Luenberger, 1988).

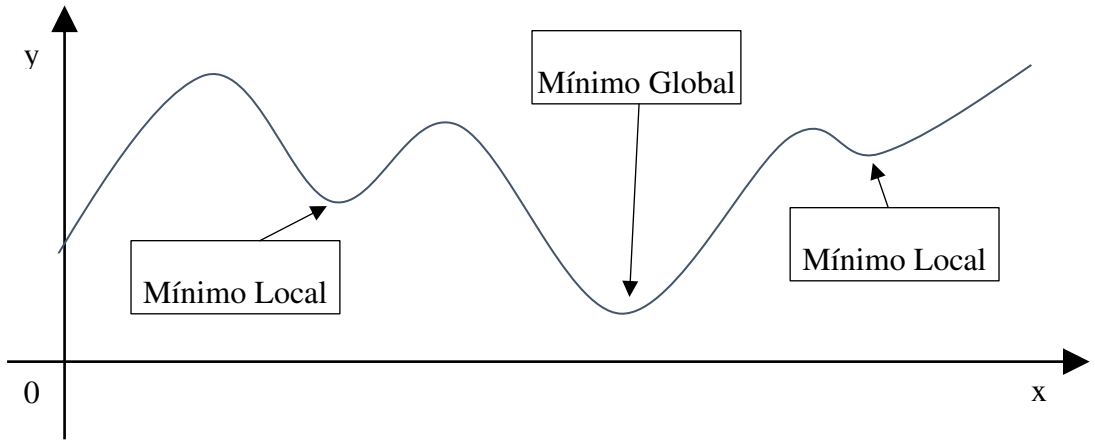
Segundo Luenberger (1988) este tipo de heurísticas, baseadas na de Lin-Kernighan, são descritas por um processo de 4 passos:

1) Gerar uma solução pseudoaleatória inicial que satisfaça as condições, que é um vetor T que satisfaça um conjunto de critério C;
2) Tentar encontrar uma solução melhor viável T’ resultante da transformação de T;

3) Se for encontrada uma melhor solução, ou seja, f(T') < f(T), então substitui-se T por T' e repete-se o segundo passo;
4) Se não for encontrada nenhuma solução melhor, T é uma solução ótima local. Repete-se o processo a partir do primeiro passo até o tempo de computação terminar, ou as soluções estiverem satisfatórias.

Quando se menciona conceitos de otimização, ou processos de procura de máximos ou mínimos, é necessário falar de conceitos de ótimos globais e ótimos locais. Os ótimos locais como o nome indica, num determinado intervalo representam os valores máximos ou mínimos, dependendo da perspetiva de otimização. Por outro lado, os ótimos globais representam os valores máximos ou mínimos na totalidade da função. A figura 8 demonstra esta descrição.

Figura 8. Exemplo de gráfico com mínimos locais e globais.



Normalmente as soluções obtidas utilizando estes processos heurísticos são ótimos locais em cada repetição, e como é visto no quarto passo do processo acima descrito, o procedimento termina quando o tempo de computação termina, ou é encontrada uma solução satisfatória, embora existam sempre a possibilidade de obtermos um ótimo global.

Na descrição do algoritmo da heurística iremos verificar que em cada iteração é obtido um ótimo local, sob a forma de mínimo local, da mesma forma que é descrito no quarto passo do processo heurístico exposto por Luenberger.

3.2.3 Adaptação do algoritmo

A adaptação do processo descrito por Luenberger (1988) requer algumas mudanças, uma vez que o seu algoritmo foi escrito em linguagem *Fortran*, e a linguagem do algoritmo da dissertação foi escrito em *MATLAB*. No entanto, os princípios descritos por Luenberger, como os quatro passos dos processos heurísticos descritos anteriormente, são utilizados, tal como alguns conceitos, usando as definições de Lin-Kernighan sobre a otimização em λ (Luenberger, 1988):

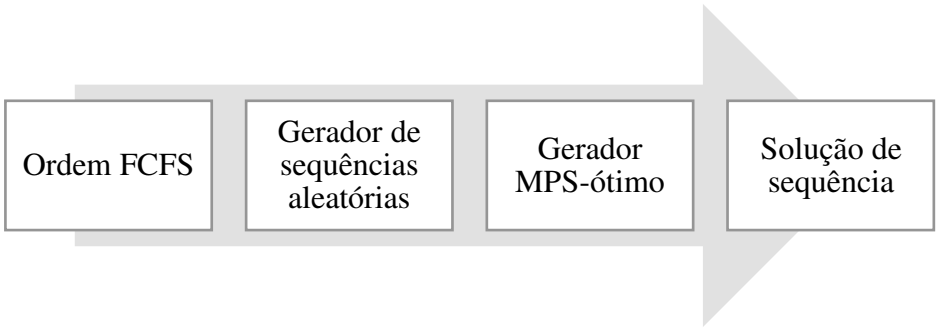
- Uma sequência de N aviões pode se dizer λ-ótima, se nenhum avião puder ser movido *r* posições, onde $r \leq \lambda$, levando a uma melhoria da sequência;
- Ao definir λ = MPS, o algoritmo produz uma quantidade grande de soluções MPS ótimas, enquanto guarda as melhores sequências;
- Uma solução λ-ótima não é necessariamente um ótimo global.

O processo adotado por Luenberger foi o seguinte:

- Utilizar a ordem FCFS como a sequência inicial e o tempo inicial para aterrar todos os aviões;
- Selecionar aviões aleatórios adjacentes na ordem FCFS, alterando a ordem gerando uma nova sequência, tendo em conta a restrição de MPS;
- A sequência MPS-ótima é gerada através da consideração de todas as recolocações possíveis, começando com o primeiro avião, e o respetivo cálculo do tempo, e escolhendo a melhor recolocação;
- O algoritmo é repetido até que não seja possível uma melhoria.

A figura 9 descreve o processo adotado por Luenberger:

Figura 9. Algoritmo ALP de Luenberger
Adaptado de Luenberger 1988



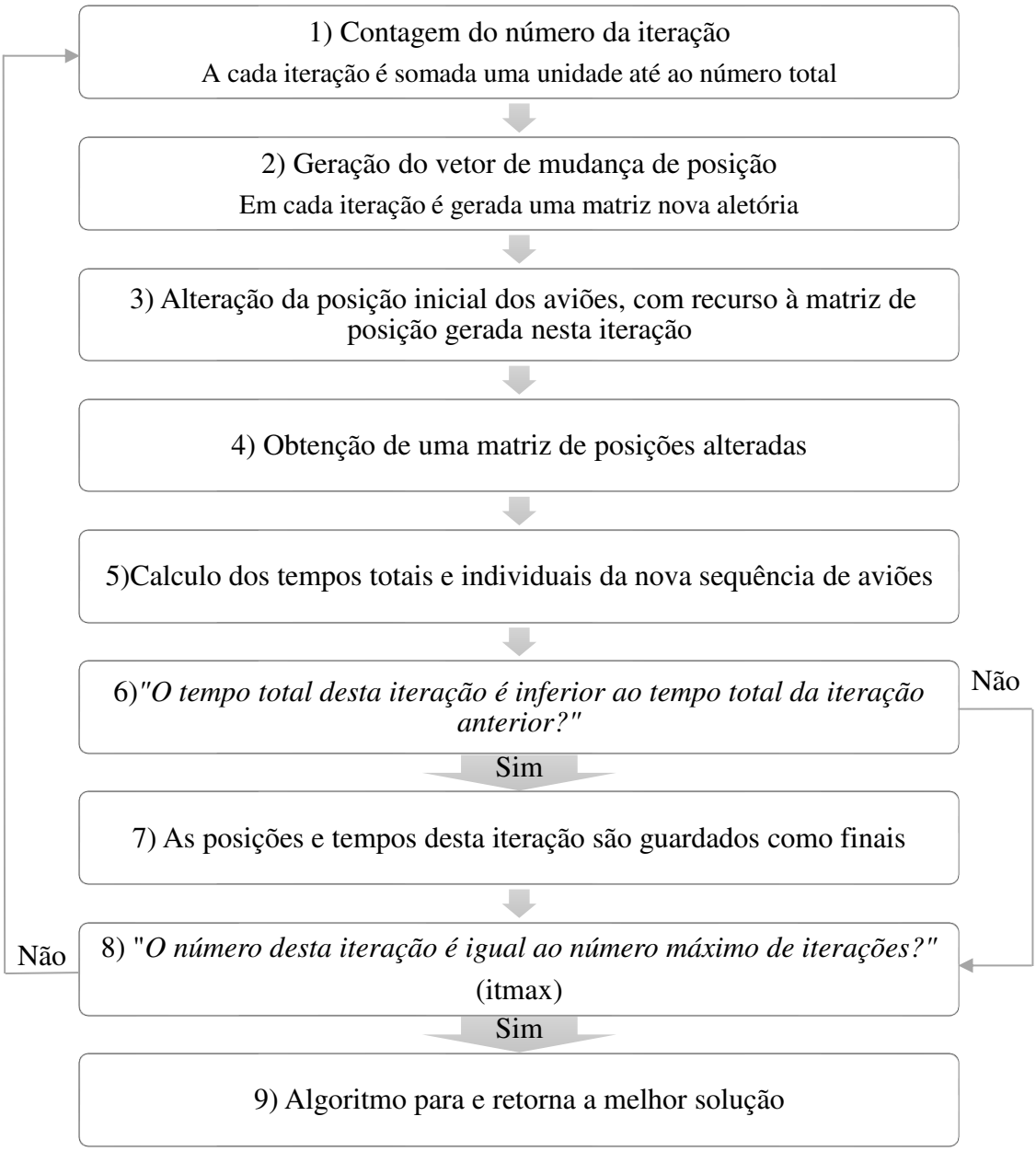
Em termos de produção da sequência MPS-ótima, existem dois mecanismos de procura: *random descent* e *steepest descent*. *Random descent* ou descida aleatória, procura de forma aleatória as melhorias em cada passo, enquanto a *steepest descent* procura em cada passo a melhor solução possível (Luenberger, 1988). Em termos de soluções, ambos os mecanismos produzem soluções satisfatórias, embora a *steepest descent* produza soluções melhores. As sequências de *random descent* não são substancialmente inferiores às do outro mecanismo. No entanto, a utilização *random descent* é mais simples, e reduz o tempo computacional quando comparado com o outro método, foi também utilizado por Luenberger e no algoritmo desta dissertação.

3.2.4 Heurística do algoritmo da dissertação

O processo heurístico utlizado no algoritmo da presente dissertação é uma adaptação de todos os conceitos mencionados anteriormente. Tem como objetivo o retorno de soluções de sequências de aeronaves com tempos totais finais, inferiores aos das sequências FCFS do início do algoritmo.

Ao utilizar os dados introduzidos, da figura 7, nomeadamente as sequências FCFS, MPS, matriz de separações temporais, e o número de iterações, é possível dar início ao processo de otimização. Utilizando a heurística descrita por Luenberger como base, é possível criar um ciclo no *script* do *MATLAB*, adaptado aos objetivos pretendidos. A figura 10 explica o processo heurístico utilizado:

Figura 10. Fluxograma do processo heurístico de otimização



Na figura 10, o processo heurístico começa com a contagem do número da iteração. Esta função serve para contabilizar o progresso do procedimento, que começa em zero e termina no número máximo de iterações introduzido anteriormente. É utlizado um ciclo "*while*", que a cada repetição soma um valor a uma variável que armazena o número da iteração, e quando esta variável iguala o número máximo de iterações introduzido, o processo termina, e retorna com a melhor solução.

O segundo passo é responsável por gerar uma matriz aleatória de mudança de posição, que é um vetor cuja dimensão é o número de aviões, e valores entre -MPS e MPS, por exemplo um MPS de 1 corresponde a um intervalo [-1;0;1], e um MPS de 2 a um intervalo [-2;-1;0;1;2]. Uma vez que o MPS é a mudança de posição máxima permitida, admite-se que uma aeronave pode mudar a posição em um valor no intervalo compreendido entre -MPS e MPS. A figura 11 apresenta um exemplo de matriz de mudança de posição, para um caso com 5 aviões e um MPS de 1 e 2, respetivamente:

Figura 11. Exemplos de vetores de mudança de posição, para 5 aviões, com MPS 1 e 2 respetivamente

MPS=1				
1	0	-1	-1	1

MPS=2				
2	1	0	2	-2

Este vetor é gerada a cada iteração, e em cada repetição do processo, um novo vetor de mudança de posição aleatória é gerado. Ou seja, em cada iteração este vetor é diferente, e é esta função que permite ter diferentes resultados. Desta forma, utilizando o MPS como intervalo de mudança de posição, é possível ter um processo heurístico, da mesma forma que no capítulo II se descrevia o processo de alteração de percurso, por meio do *λ-opt*. No segundo passo, também existe um mecanismo que previne que as primeiras posições até MPS não tenham valores de mudança de posição negativos. Isto significa que se tivermos um MPS de 2, a mudança de posição do primeiro avião só pode ser 0, 1 ou 2, e a do segundo -1, 0, 1, 2, caso contrário teríamos posições negativas.

Também pode surgir a questão se este processo aleatório, através do vetor da mudança de posição, é capaz de obter uma larga variedade de soluções que se aproxima de valores ótimos satisfatórios. A equação 4 demonstra o número de combinações de vetores de mudança de posição possíveis:

$$\text{Combinações MPS} = \frac{\text{Número Aviões!}}{\text{Dimensão[MPS]!} * (\text{Número Aviões} - \text{Dimensão[MPS]})!} \quad (4)$$

Se pegarmos na equação 4, e num exemplo com 40 aviões e um MPS de 2, correspondente a uma dimensão de 5, pois o intervalo é [-2; -1; 0; 1; 2], temos um total de 658 008 vetores de mudança de posição possíveis. Ao utilizar valores elevados de iterações, obtemos uma maior probabilidade de termos gerado todos os vetores de mudança de posição possíveis, por exemplo na execução do algoritmo foram utlizados valores respeitantes ao número de iterações de 1 milhão e 100 milhões.

O terceiro passo envolve a alteração das posições com recurso ao vetor inicial e à vetor de mudança de posição, gerando uma nova sequência para aquela iteração. Este processo é feito para cada aeronave, começando na primeira, e à sua posição é somada o valor de mudança de posição. Por exemplo se uma aeronave na posição 5, tiver um valor de mudança de 2, irá passar para a posição 7, ou então uma aeronave que esteja na posição 10, e tenha um valor de mudança de posição de -1, passa para a posição 9. Ao mudar a posição de um avião, o avião que estava na posição nova, passa para a posição antiga do avião original, ou seja, um avião que passe da posição 5 para a 7, obriga ao avião que estava originalmente na posição 7 passar para a posição 5. A figura 12 exemplifica este passo:

Figura 12. Exemplo de alteração de posição do terceiro passo da figura 10

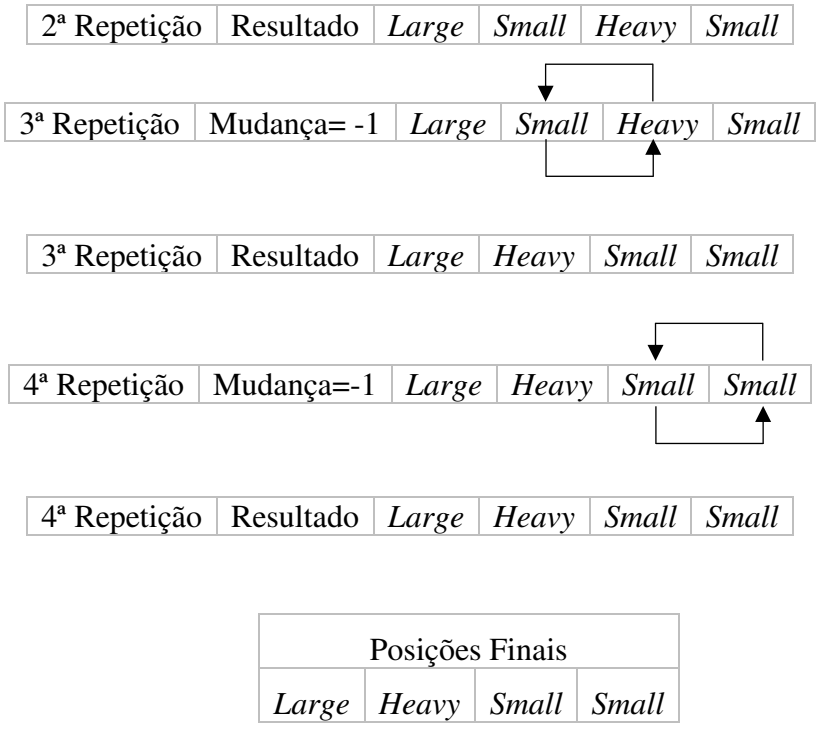
Posições Originais			
Small	*Large*	*Heavy*	*Small*

Mudança de Posição			
1	0	-1	-1

1ª Repetição	Mudança=1	*Small*	*Large*	*Heavy*	*Small*

1ª Repetição	Resultado	*Large*	*Small*	*Heavy*	*Small*

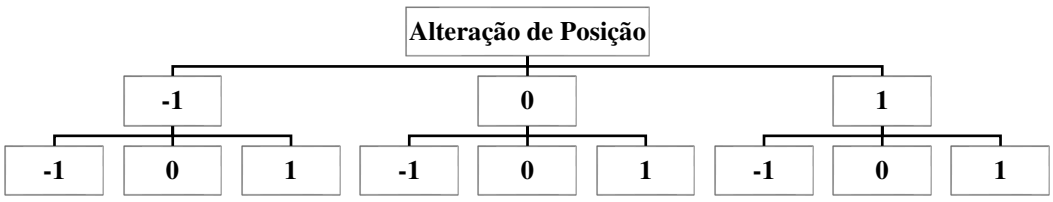
2ª Repetição	Mudança=0	*Large*	*Small*	*Heavy*	*Small*



Relembro que também é possível verificar na figura 12, que uma mudança de posição positiva corresponde uma alteração de posição crescente, por exemplo da 5 para a 7, e uma mudança negativa implica uma alteração decrescente, por exemplo da 7 para a 5.

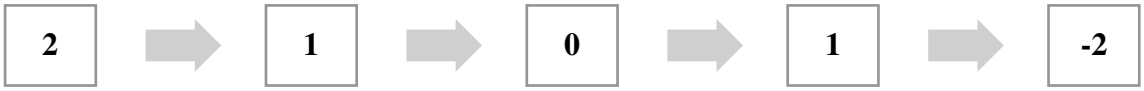
Como foi descrito anteriormente, o vetor da mudança de alteração de posição aleatória é gerado na sua totalidade no início da iteração. No entanto, poderia ter sido utilizado outra forma da alteração da posição, através de passos, ou seja, em cada avião. Assim, em cada avião é escolhida a alteração de posição que promove a maior redução de tempo. A figura 13 exemplifica este procedimento por passos, com dois aviões e o MPS de 1:

Figura 13. Exemplo de otimização por passos, com um MPS de 1



Por outro lado, e da forma que foi utlizada no algoritmo, a alteração de posição é feita com base num vetor de alteração de posição, que é aleatório para cada iteração, e é predefinido na sua totalidade antes de qualquer alteração de posição. Este método tem as vantagens de ser mais simples de implementar, exigir um menor tempo computacional, e permitir alterações nas posições iniciais. Ou seja, no caso anterior, quando é tomada uma decisão sobre a alteração de posição nos primeiros aviões, esta mantém-se até ao final do processo heurístico, enquanto na metodologia do vetor de alteração de posição é possível calcular ramificações alternativas, que embora produzam piores resultados inicialmente, no final do processo podem produzir soluções melhores. A figura 14 exemplifica este processo, com um exemplo de cinco aviões e um MPS de 2.

Figura 14. Exemplo de alteração de posição de acordo com um vetor de mudança de posição aleatória e MPS de 2



Voltando novamente à figura 10, o quarto passo envolve a obtenção do vetor com as posições alteradas, através da mudança de posição. Este passo foi exemplificado na figura 12, este vetor é dedicado a cada iteração, guardada como vetor do ciclo. O vetor de posições do ciclo é um transformado do vetor de posições iniciais, e é necessário contabilizar o tempo total desta nova sequência.

O quinto passo é responsável pela categorização do vetor de posições do ciclo, bem como a contabilização dos tempos de separação novos para cada aeronave. Este processo é análogo aquele visto inicialmente na figura 7, que da mesma forma categorizava, e calculava os tempos parciais para a matriz de posições iniciais. Obtidos os tempos parciais, este são somados, e obtemos o tempo total do ciclo ou da iteração.

O sexto passo é uma decisão, onde é comparado o tempo total da iteração e o tempo total inicial. Caso o tempo da iteração seja inferior ao tempo inicial, este tempo é guardado numa variável como “tempo total final”. Por outro lado, caso o tempo do ciclo seja superior ao tempo inicial, estas sequências não têm utilidade do ponto de vista do

problema, por isso é eliminada, e o processo passa para o passo oito, que retorna ao início caso o número de iterações seja inferior ao número máximo de iterações prefixado.

O sétimo passo são guardados os tempos totais do ciclo, como mencionado no sexto passo, e também são memorizadas as posições das aeronaves para serem depois comparadas e categorizadas na apresentação das soluções.

No oitavo passo estamos perante outra decisão, que no *script*, é tomada no início do bloco no ciclo *"while"*, sendo responsável pela longevidade do processo heurístico. Assim, enquanto o número da iteração for menor do que o número máximo de iterações o processo recomeça no primeiro passo, e quando o número for igual ao máximo o processo termina, e em cada repetição é somada uma unidade ao contador. O número máximo de iterações tem influência sobre os resultados obtidos e o tempo computacional, e será estudado em melhor detalhe no próximo capítulo.

O nono e último passo retorna a melhor solução obtida no sétimo passo, enquanto o processo está a decorrer. Esta solução é a final, e é utlizada para os cálculos de otimização, redução de tempo e análise de posições, com a posterior categorização.

O algoritmo descrito neste capítulo é a peça fundamental nesta dissertação, no entanto se não for aplicado a casos práticos a sua utilidade fica reduzida. O próximo capítulo é responsável pela execução do algoritmo com diversos exemplos práticos, bem como a respetiva análise comparativa dos resultados.

CAPÍTULO IV – RESULTADOS E ANÁLISE

O enquadramento computacional teve como objetivo a produção de um algoritmo, que conseguisse uma sequência final de aeronaves com um tempo total final menor do que o inicial, ou seja uma sequência otimizada. O capítulo anterior descreveu o algoritmo, bem como o processo heurístico de otimização, que é essencial à produção de soluções. Neste capítulo será executado o algoritmo, fazendo variações de parâmetros como forma de obtenção de resultados diferentes.

4.1 Exemplo inicial

Luenberger utilizou o seguinte exemplo de aviões e porções, que também vai ser utilizado como exemplo base inicial em termos de comparação:

- Número de aviões total: 40;
- Porção de aviões *small*: 15%, 6 aviões;
- Porção de aviões *large*: 42,5%, 17 aviões;
- Porção de aviões *heavy*: 42,5%, 17 aviões;
- Valores de MPS entre 1,2 e 3.

A matriz das separações utilizada foi a de Ming *et al*. (2018), representada pela tabela 1 desta dissertação, considerando que provêm de uma fonte mais recente do que Luenberger (1988), e também introduz alguma diferença para comparação de resultados. O MPS é algo que pode ser variável, mas neste exemplo vai ser considerado como 2.

Como foi já mencionado, o caso mais eficiente em termos temporais possível é se tivéssemos uma sequência composta por todos os aviões *small* em primeiro lugar, seguidos dos *large* e por fim os *heavy*. Pegando no exemplo, temos 6 aviões *small*, 17 aviões *large* e 17 aviões heavy, por esta ordem, e a sequência destes 40 aviões totaliza num tempo de 3 940 segundos, ou 1 hora 5 minutos e 40 segundos. Este tempo é o menor tempo possível, considerando as quantidades parciais e o número total de aviões. Este tempo será denominado de tempo base, e será utilizado para comparação.

Utilizando este exemplo, é possível fazer a execução completa do algoritmo, onde as posições iniciais dos aviões são aleatórias. Desta forma, o tempo inicial será sempre

superior ao tempo base. Considerando as variáveis de introdução anteriores, temos os seguintes resultados de um exemplo de execução:

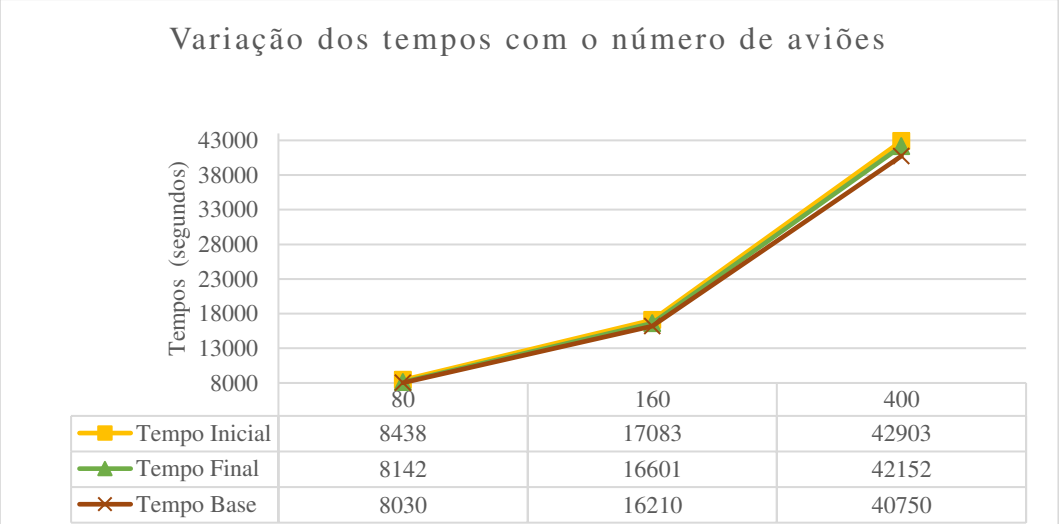
- Tempo total inicial: 4 240 segundos, ou 1 hora 10 minutos e 40 segundos;
- Tempo total final: 3 975 segundos, ou 1 hora 6 minutos e 15 segundos;
- Redução de tempo 265 segundos, ou 4 minutos e 25 segundos;
- Melhoria de 6,25%

Ao analisarmos este exemplo, verificamos que temos uma melhoria de apenas 6,25%, que embora pareça pequeno, o tempo total final de 3 975 segundos do exemplo, para o tempo base de 3 940 segundos, tem uma diferença de apenas 35 segundos. Esta redução de 265 segundos, do tempo inicial para o final, permite introduzir mais 1 a 3 aviões, dependendo da categoria, onde neste exemplo estamos perante aproximadamente uma hora de operação. No entanto, analisando a totalidade de um dia de operação de um aeroporto, a capacidade em termos de aviões extra é significativa e multiplicada.

4.2 Variação do número de aviões

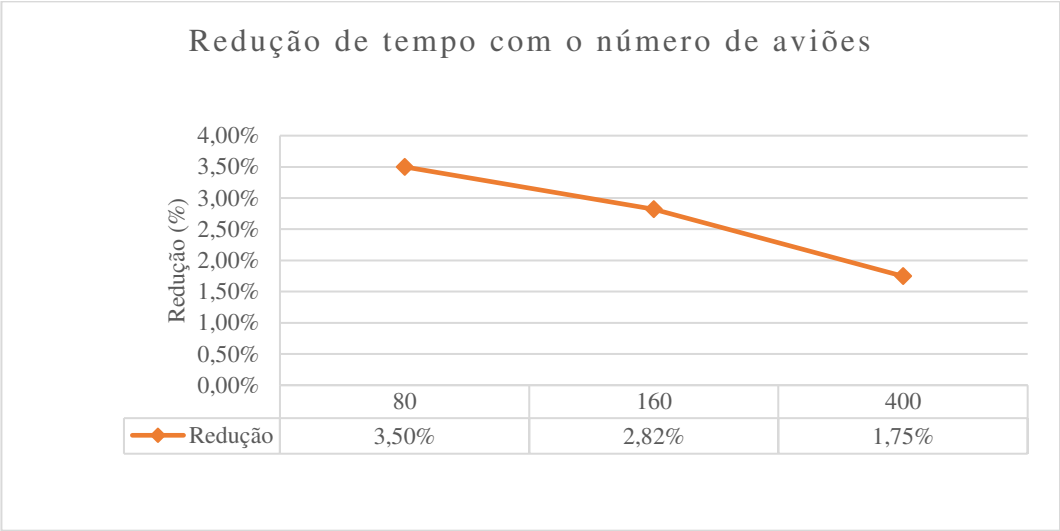
A variação de parâmetros é uma das formas de obter resultados diferentes. Começamos com a variação dos números de aviões, com exemplos com 80, 160 e 400 aviões, MPS 2 e as porções do exemplo base. A figura 15 demonstra os exemplos:

Figura 15. Gráfico da variação dos tempos com o número de aviões, com 80, 160 e 400 aviões



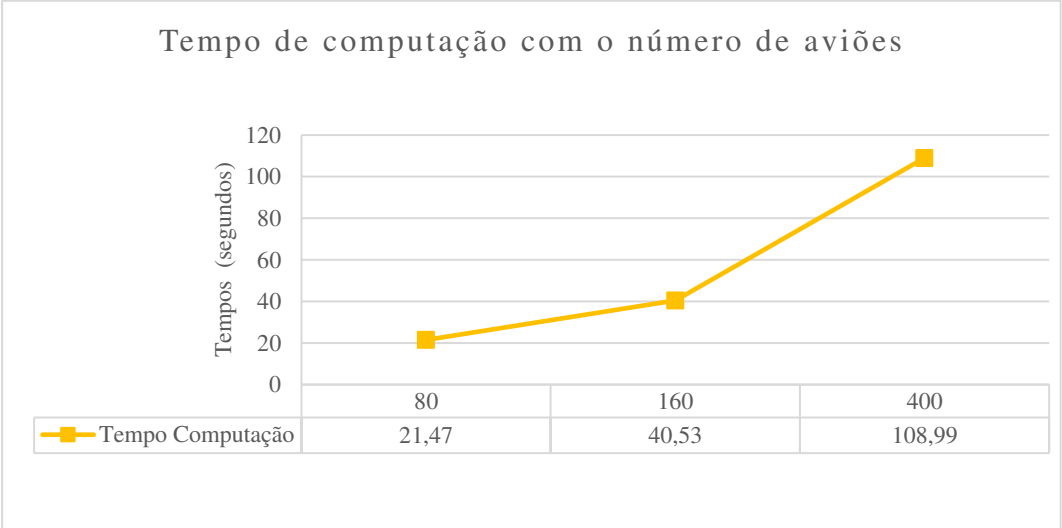
Da figura 15, analisamos que quanto maior o número de aviões maiores são todos os tempos, quer os tempos iniciais, finais e mesmo o base. Também é possível averiguar que a diferenças entre os três tempos é mais acentuada quanto maior o número de aviões. A redução de tempo com o número de aviões é dada pela figura 16:

Figura 16. Gráfico da redução dos tempos com o número de aviões, com 80, 160 e 400 aviões



A figura 16 consegue complementar a figura 15, na medida em que a redução de tempo é menor quanto maior o número de aviões. Uma hipótese de explicação desta situação pode ser que ao aumentar o número de aviões, as alterações de posições têm um menor efeito, havendo uma maior absorção da melhoria. A figura 17 mostra, como esperado, que o aumento do número de aviões implica o aumento do tempo de computação.

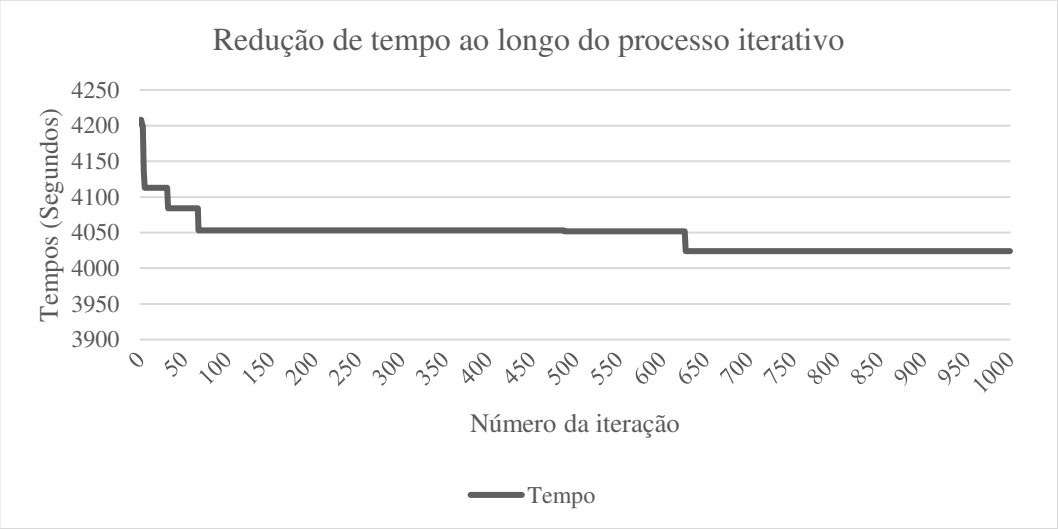
Figura 17. Gráfico da variação do tempo de computação com o número de aviões.



4.3 Variação do número de iterações

O número de iterações é um dos parâmetros de introdução que afeta os resultados, na medida em que quanto maior o número de iterações, maior a probabilidade de ser gerada um vetor de mudança de posição que gere uma melhor sequência. A figura 18 é um exemplo da redução de tempo pelo processo heurístico, explicado no capítulo III:

Figura 18. Gráfico de um exemplo redução de tempo durante o processo iterativo, ou processo heurístico de otimização, com 1 000 iterações



Como é possível retirar da figura 18, o processo heurístico tem maior efetividade nas primeiras iterações, e quanto maior o número da iteração menor a redução ao longo do processo. No entanto, a partir da iteração 500 existe uma redução do tempo de 4 052 segundos para 4 024, especificamente na iteração número 629. Ou seja, caso o exemplo tivesse parado na iteração número 500, o algoritmo tinha retornado o tempo da sequência ótima como 4 052 segundos, e não os 4 024 das 1 000 iterações, que é uma diferença de 28 segundos.

Assim, verificamos que quanto maior o número de iterações, menor deverão ser os tempos finais, bem como a redução de tempo. Embora, a partir de certo número de iterações esta redução deixe de ter valor de redução muito significativos. As figuras 19 e 20 demonstram este facto, onde a figura 19 é a variação dos tempos com as iterações e a 20 a redução do tempo, ambas com 100 000, 1 000 000 e 100 000 000 de iterações, utilizando um MPS de 2, bem como 40 aviões e as porções do exemplo inicial.

Figura 19. Gráfico da variação dos tempos com o número de aviões com o número de iterações

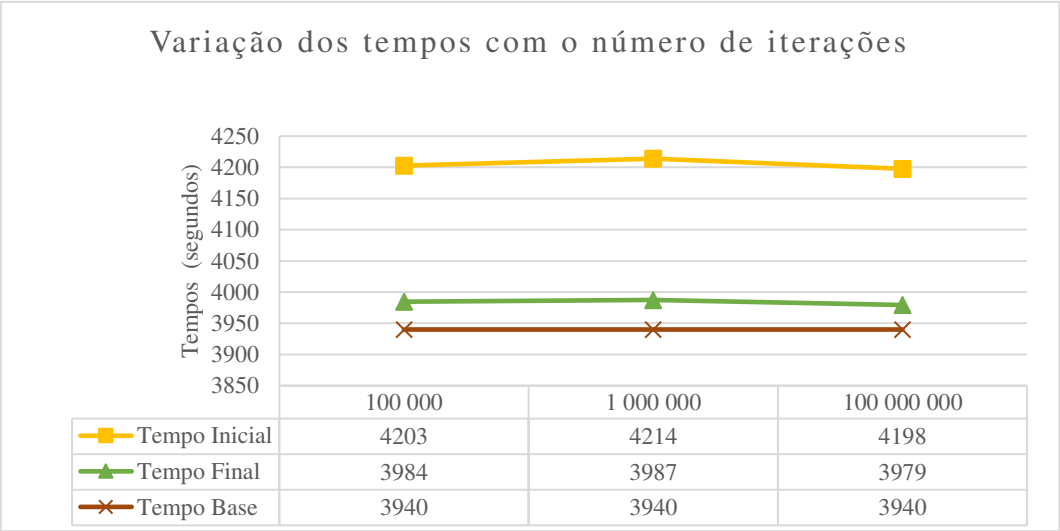
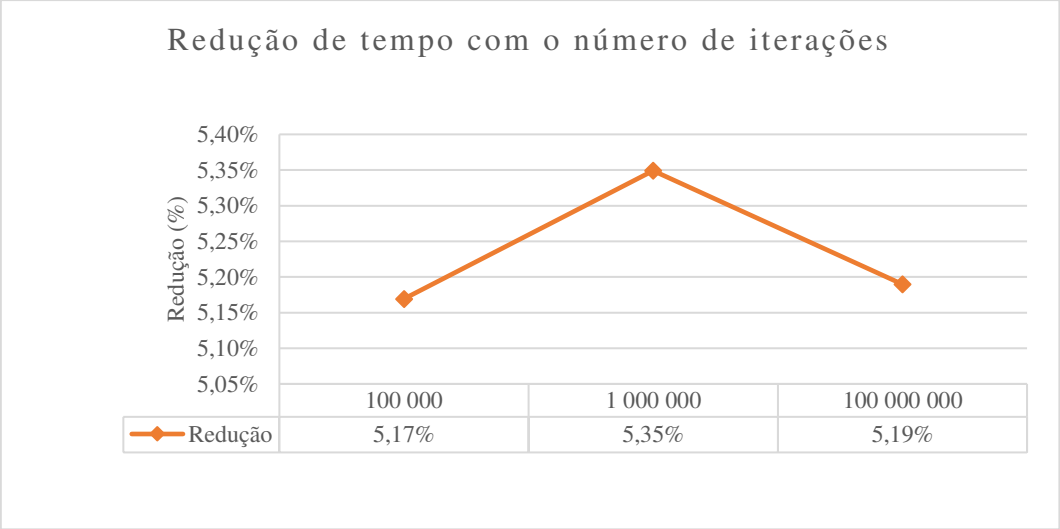
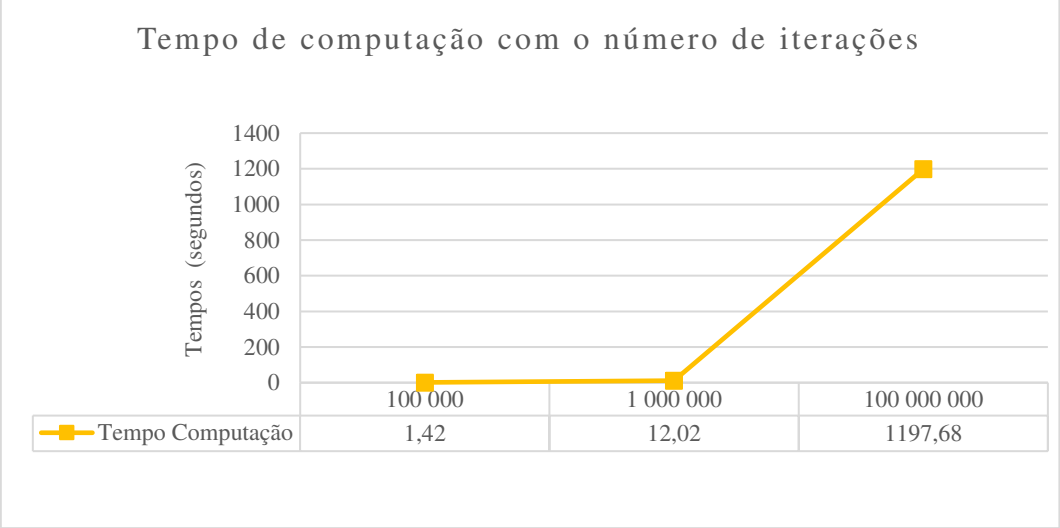


Figura 20. Gráfico da redução de tempo com o número de aviões com o número de iterações



Como evidenciado anteriormente, as figuras 19 e 20 mostram que não existem diferenças significativas quer nos tempos finais, quer na redução, com o aumento do número de iterações. O valor da redução, da figura 20, para 100 000 000 de iterações é inferior ao 1 000 000, pois as diferenças são tão pequenas que qualquer variação pode provocar um comportamento diferente do esperado. No entanto, se traçarmos uma linha de tendência, o valor da redução aumenta com o número de iterações, embora em cada vez quantidades mais pequenas. Um parâmetro que aumenta com o número de iterações é o tempo de computação, e como esperado, o número de iterações aumenta de forma exponencial o tempo computacional do algoritmo, como se apura na figura 21:

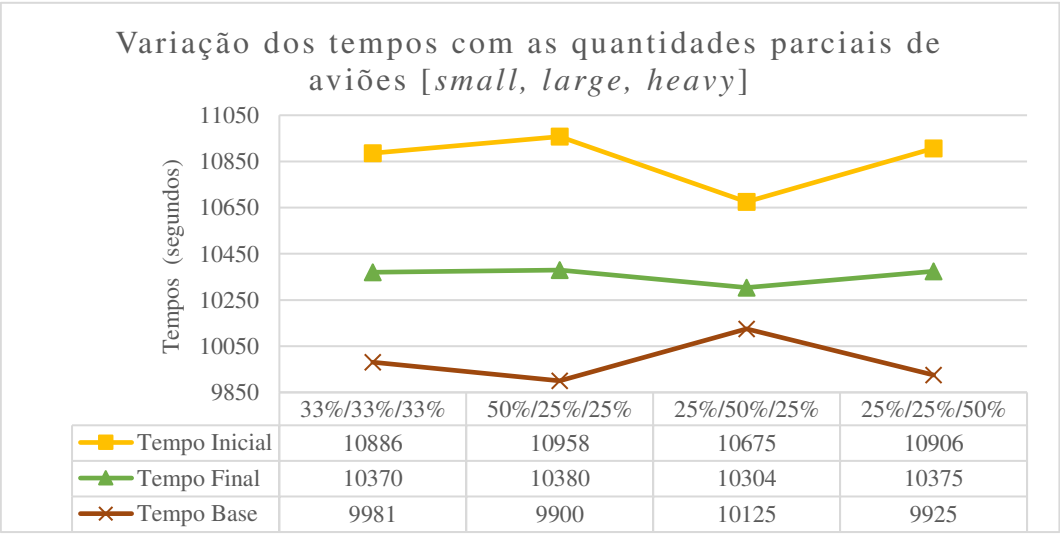
Figura 21. Gráfico do tempo de computação com o número de aviões com o número de iterações



4.4 Variação das quantidades parciais de aviões

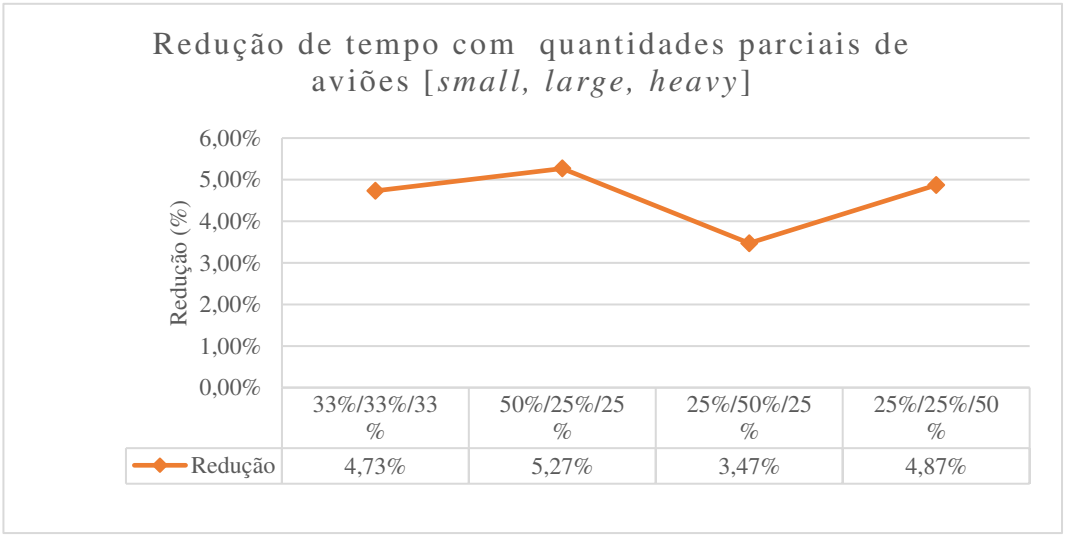
Outra variação de parâmetros é a variação das quantidades parciais de aviões, nomeadamente as categorias *small*, *large* e *heavy*. O exemplo inicial utilizava uma porção de 15% para os aviões *small*, e o remanescente para os *large* e *heavy*, ou seja 42,5% para cada. A figura 22 exemplifica a variação dos tempos com diferentes porções, onde a ordem das percentagens é *small*, *large* e *heavy*. Todos os exemplos utilizam 100 aviões, para termos números inteiros de aviões, MPS de 2 e 1 000 000 de iterações:

Figura 22. Gráfico da variação dos tempos com as quantidades parciais de aviões, *small*, *large* e *heavy*



Da figura 22 é possível tirar algumas conclusões: o menor tempo base é dado quando a maior quantidade é de aviões *heavy*, e o maior é quando os aviões *large* apresentam a maior percentagem; por outro lado, o menor tempo final é dado pela quantidade maioritária de avião *large* e o maior pela maioria de aviões *heavy*; a maior diferença entre o tempo inicial e o tempo base é dada pela maior porção de aviões *small*. A divisão igualitária em 33% para cada categoria também serve como variável de controlo, e verificamos que a única porção com maior tempo final do que a de controlo é quando a maior quantidade é de aviões *large*. Por outra perspetiva, a figura 23 dá a variação da redução:

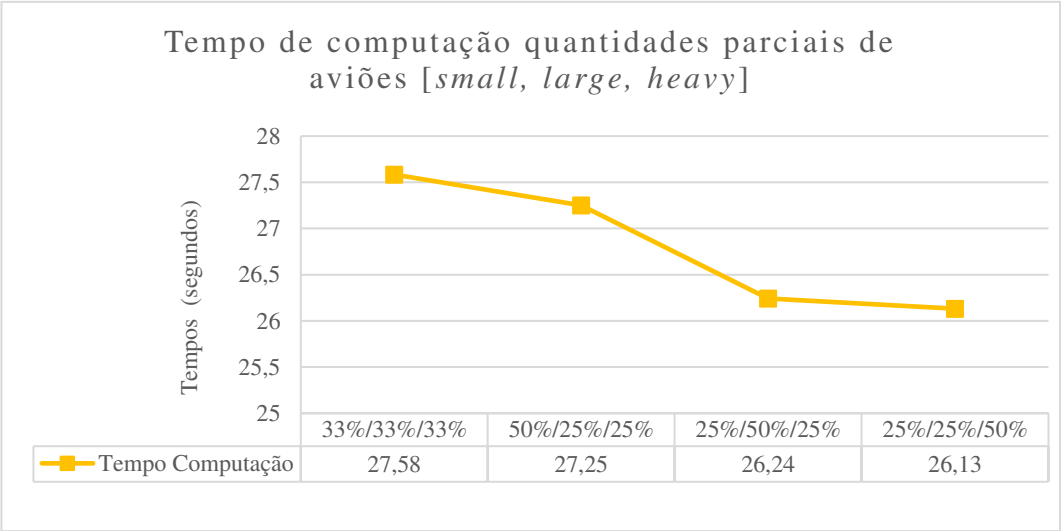
Figura 23. Gráfico da redução de tempo com as quantidades parciais de aviões, *small*, *large* e *heavy*



Analisando a figura 23, concluímos que a maior redução de tempo é dada pela porção com 50% de aviões *small*, o que é concordante com a figura 22, onde é a fração com maior tempo inicial e maior diferença para o tempo final. Por oposição, a porção com 50% de aviões *large* é aquela que apresenta uma menor redução em tempo, mas que pela figura 22 era aquela que o tempo final mais se aproximava do tempo base da sequência ótima. Também, examinando as figuras, a porção *large* é aquela em que a sequência inicial está mais perto da sequência ótima. A porção com 50% de aviões *small*, da figura da redução do tempo, é aquela com maior efetividade na mudança de posições.

Em termos de computação, as porções apresentam resultados similares, embora exista uma tendência de quanto maior a quantidade de aviões maiores, menor o tempo, exceto para a porção de controlo. A figura 24 demonstra esta situação:

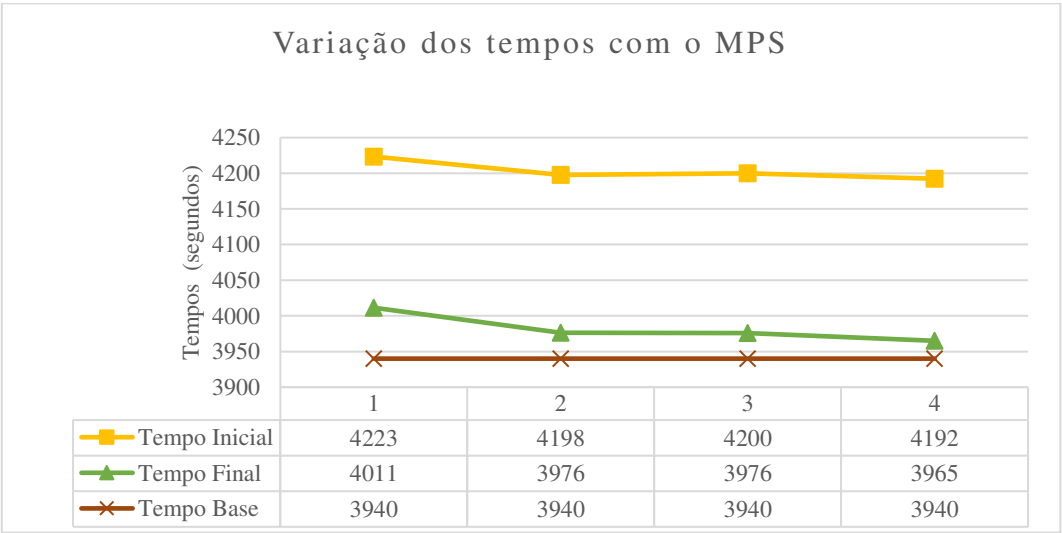
Figura 24. Gráfico do tempo de computação com as quantidades parciais de aviões, *small*, *large* e *heavy*



4.5 Variação do MPS

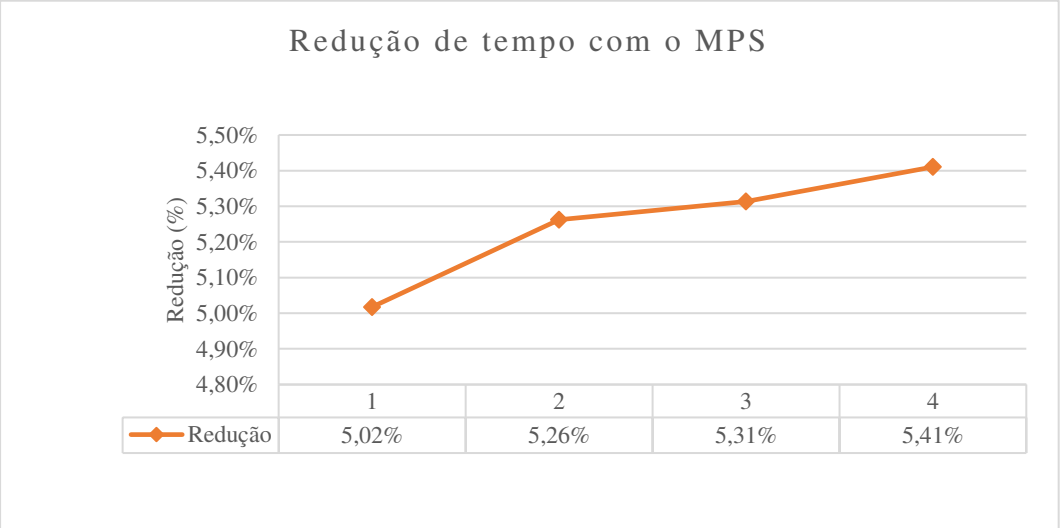
Por fim, temos a variação do *Maximum Position Shift*, onde, em termos práticos, é aquele que será mais fácil de aplicar. Segundo Luenberger (1988), o MPS máximo em termos práticos é de 3, pois alterar mais do que 3 posições um avião na sequência deixa de ser conveniente. Para efeitos de demonstração, foram utilizados valores de MPS de 1, 2, 3 e 4, usando 40 aviões, 1 000 000 de iterações, e as quantidades parciais do exemplo inicial. A figura 25 apresenta a variação dos tempos para diferentes valores de MPS:

Figura 25. Gráfico da variação dos tempos com o MPS



Examinando a figura 25, apuramos que quanto maior o MPS, menor o tempo final, considerando que a variação dos tempos iniciais não é significativa. A figura 26 apoia esta conclusão com a exposição da redução do tempo com o MPS:

Figura 26. Gráfico da redução e tempo com o MPS



Analisando ambas as figuras 25 e 26, concluímos então, que de facto quanto maior o valor de MPS, maior é a redução de tempo. Isto é concordante com as conclusões de Luenberger (1988), pois em termos lógicos, quanto maior o número de mudanças máximas de posição permitido, maior será a probabilidade de gerarmos uma sequência com menores tempos finais.

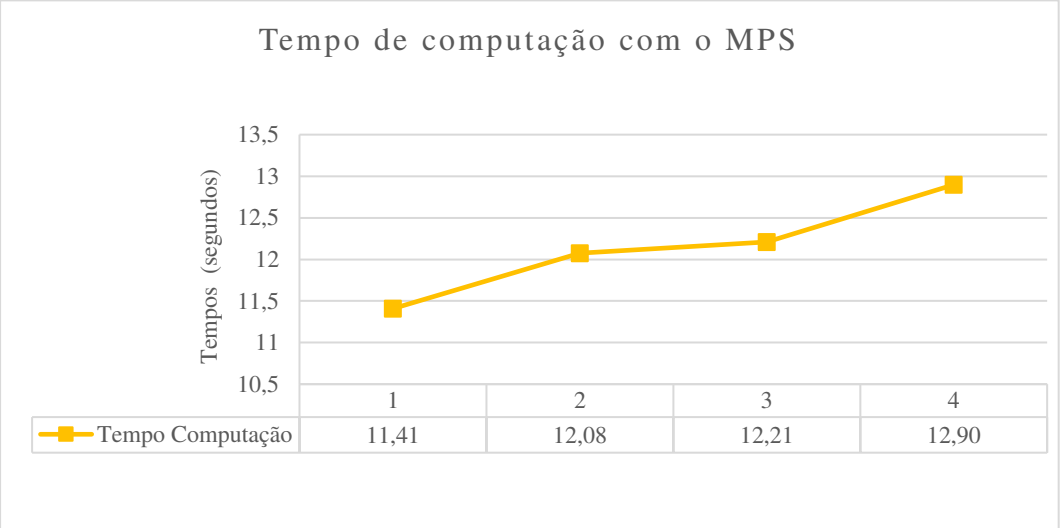
Também verificamos que só admitindo uma mudança de uma posição, ou seja, um MPS de 1, que implica a alteração da sequência com a aeronave imediatamente na vizinhança, obtemos uma redução significativa na ordem dos 5%. Por outra perspetiva, o aumento do MPS conduz a reduções que podem ser consideradas significativas, embora os valores de diminuição de tempo com MPS de 1 a 4 sejam relativamente próximos. Por isso é necessário averiguar em termos de custo-benefício qual o valor ótimo de MPS, uma vez que a alteração da sequência em várias posições acresce custos, bem como logística.

Pegando novamente na figura 25, verificamos que a diferença de tempos entre o final e o base para o MPS de 1 é de 71 segundos, para o de 2 e 3 é de 36 segundos e para o MPS de 4 é de 25 segundos. Isto é concordante com as conclusões anteriores, onde o MPS de 4 é o valor que se mais se aproxima da sequência ótima, fazendo uma disparidade de

46 segundos para a diferença do MPS de 1, que é quase o tempo necessário para alocar mais um avião.

Luenberger (1988) conclui que o aumento do MPS implica também um aumento do tempo computacional. Deve-se ao facto de aumentarmos as mudanças de posição, que por sua vez aumenta a complexidade do algoritmo. A figura 27 demonstra a variação do tempo computacional com o aumento do MPS de 1 a 4:

Figura 27. Gráfico do tempo de computação com o MPS



Concluindo, a figura 27 demonstra que o menor tempo computacional é dado pelo MPS de 1, e que de facto o maior MPS aumenta esta duração de cálculo. Embora os tempos sejam relativamente próximos, e utilizando os argumentos anteriores, o valor de MPS deve ser também ótimo, de forma a não aumentar a complexidade e custos de operação num aeroporto.

Desta forma, na variação de parâmetros deve ter em conta quais aqueles que mais se adequam às necessidades de um aeroporto. Como foi visto neste capítulo, os diferentes critérios de introdução têm influências muito díspares nos resultados, e atendendo ao que foi apresentado, alguns, como a variação das porções e do número de aviões, não são úteis do ponto de vista prático, uma vez que são valores fixos da operação, previamente definidos. Por isso, a variação do MPS, e o número de iterações são aqueles que tiveram resultados que mais facilmente se podem aplicar à realidade.

CAPÍTULO V – CONCLUSÃO

O objetivo da otimização nas chegadas num aeroporto foi o objeto de estudo desta dissertação. No entanto, as ferramentas de otimização são diversificadas, e é necessário um foco de forma a tentar obter uma solução ótima da forma mais eficiente. A bibliografia referente ao tema apresenta várias maneiras e métodos de obter resultados. A escolha do *Traveling Salesman Problem* deveu-se a vários fatores: a grande quantidade de literatura disponível e exemplos, a facilidade de compreensão e adaptação ao exemplo da dissertação, a existência de autores que utilizaram o TSP para problemas similares e a versatilidade geral da metodologia.

Como foi visto, não é possível a aplicação direta do TSP clássico, no entanto a polivalência do método, exemplos disponíveis, e a facilidade de adaptação ao *MATLAB*, foram responsáveis pela criação de um algoritmo baseado no TSP, com resolução por métodos heurísticos. Embora o *MATLAB* possua ferramentas de otimização incorporadas, devido à natureza do problema, como maneira de ter maior controlo sobre o algoritmo de resolução e também como desafio extra, foi criado um *script* de raiz, exclusivo para o problema em causa.

O capítulo IV retorna resultados que são aqueles pretendidos quer na forma de otimização geral, quer na forma operacional necessária num aeroporto. O exemplo inicial apresenta uma redução de 265 segundos, o que corresponde a uma melhoria de 6,25%. O intervalo médio de melhoria, para as condições iniciais do exemplo, anda entre os 3 e os 7 %. Um dos fatores mais importantes é a sequência inicial, pois quando esta mais se aproxima da sequência ótima, menor é a necessidade de melhoria. No entanto, como a sequência inicial é aleatória para todas as execuções do algoritmo, existe uma diferença para os tempos iniciais. Por outro lado, os tempos finais andam sempre bastante próximos uns dos outros.

O mesmo capítulo retorna valores que nos permitem tirar algumas conclusões sobre a variação dos parâmetros. A variação do número de aviões, no caso do aumento, implica um aumento de todos os tempos, o que seria esperado. No entanto, a redução de tempo é menor com a quantidade, onde uma hipótese de explicação pode ser que com o aumento do número de aviões, as alterações de posição tenham um menor efeito na otimização geral.

As variações de quantidades, especificamente de aviões e iterações, têm um impacto direto no tempo computacional, o que também seria esperado. Todavia, o aumento do número de iterações, a partir de determinado valor, não têm impacto significativo nas melhorias dos resultados obtidos, sendo os valores temporais muitos próximos uns dos outros, tendo ainda a agravante de o tempo computacional aumentar significativamente com o aumento das iterações, exemplificado pelos casos de 1 000 000 e 100 000 000 de iterações, onde o aumento em cem vezes do número de iterações implica um incremento na mesma proporção do tempo de execução, de 12 segundos para 20 minutos.

A variação das quantidades parciais teve algum impacto nas variações de tempo, e é possível verificar que as porções com aviões *small* são as que apresentam uma maior redução de tempo. A resposta por detrás disto, pode-se dever ao facto de, pegando na tabela 1, as separações com aviões *small*, são aquelas que apresentam ambas separações maiores e menores. Ou seja, ao alternar um avião *small* para uma posição mais eficiente, o ganho temporal é superior quando comparada a outras categorias.

O *Maximum Position Shift* revelou-se como uma variável de extrema importância, quer por ser uma restrição base na resolução do problema, quer no próprio processo heurístico de otimização, pois é esta que permite a viabilidade operacional, na medida em que restringe os processos de otimização de sequência. Verifica-se que o aumento do MPS tem um impacto direto nos tempos finais, e que o maior valor de MPS implica uma maior redução de tempo. Em termos lógicos faz algum sentido, pois existe maior capacidade de alternar as posições, otimizando as sequências. No entanto, há que pesar quais os impactos do aumento do MPS, pois implica o aumento de custos associados, bem como a exigência da operação, por isso Luenberger (1988) já afirmava que um MPS superior a três não tinha sentido em termos práticos. Também comparando os valores de redução de tempo, apura-se que o próprio valor de MPS igual a 1 já apresenta um resultado significativo de 5%.

Desta forma, valores de otimização a rondar os 4 a 6%, embora pareçam diminutos, podem implicar ganhos significativos em termos de capacidade num aeroporto congestionado, com uma solução que é pouco intensiva em termos de capital, quando comparada com a construção de novas infraestruturas aeroportuárias, como pistas.

REFERÊNCIAS

Helsgaun, K. (2000). An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Research,* 126, 1, 106-130.

Luenberger, R. (1988). A Traveling-Salesman-Based Approach to Aircraft Scheduling in the Terminal Area. *Nasa Techinal Memorandum 100062*. Ames Research Center, Moffett Field California.

Ma, J., Delahaye, D., Sbihi, M., Scala, P., & Mota, M. (2019). Integrated optimization of terminal maneuvering area and airport at the macroscopic level. *Transportation Research Part C: Emerging Technologies, 98*, 338-357. doi:10.1016/j.trc.2018.12.006

Matai, R., Singh, S., & Murari, M. (2010). Traveling Salesman Problem:An Overview of Applications, Formulations, and Solutions Approaches. Em D. Davendra, *Traveling Salesman Problem, Theory and Applications.* InTech.

Ming, L., Bian, L., Feifeng, Z., Chengbin, C., & F. C. (2018). A Two-stage Stochastic Programming Approach for Aircraft Landing Problem. *15th International Conference on Service Systems and Service Management (ICSSSM 2018)*. Hangzhou, China. doi:10.1109/ICSSSM.2018.8465107

Mo, Y. (2010). The Advantage of Intelligent Algorithms for TSP. Em D. Davendra, *Traveling Salesman Problem, Theory.* InTech.

Moler, C., & Little, J. (2020). A history of MATLAB. *Proceedings of the ACM on Programming Languages,* 4, 81, 1-67. doi:10.1145/3386331

Shone, R., Glazebrook, K., & Zografos, K. (2021). Applications of stochastic modeling in air traffic management: Methods, challenges and opportunities for solving air traffic problems under uncertainty. *European Journal of Operational Research,* 292, 1, 1-26. doi:10.1016/j.ejor.2020.10.039

Xiao-Bing, H., & Wen-Hua, C. (Junho de 2005). Receding horizon control for aircraft arrival sequencing and scheduling. *IEEE Transactions on Intelligent Transportation Systems,* 6, 2, 189-197. doi:10.1109/TITS.2005.848365

Zografos, K. G., Androutsopoulos, K. N., & Madas, M. A. (2018). Minding the gap: Optimizing airport schedule displacement and acceptability. *Transportation Research Part A: Policy and Practice,* 114, 203-221. doi:10.1016/J.TRA.2017.09.025

ANEXOS

LISTA DE ANEXOS

Anexo A: *Script MATLAB*

```
%Algortimo TSP chegadas num aeroporto, dissertação JM.Mendes

clc;
clear all;
close all;

t=cputime; %Contagem do tempo de CPU
tic

%% Inputs
nAvioes=40;  %Número total de avioes
%Atencao aos valores das porçoes serem valores inteiros
iteracoes=1000000; %Número de iterações no processo heuristico
CostMatrix= [99 133 196;74 107 131;74 80 98]; %Matriz custo com os tempos entre↙
aeronaves

%Porções de categorias de aviões
smallPortion=0.15;
largePortion=0.425;
heavyPortion=0.425;
%Nota: Atenção que as quantidades parciais de aviões têm que ser números inteiros

MPS=2; %Maximum Position Shift

%Fim dos Inputs


%% Categorização
small=smallPortion*nAvioes;
large=largePortion*nAvioes;
heavy=heavyPortion*nAvioes;


%Tempos da matriz custo
tHH=CostMatrix(1,1);%Heavy-Heavy
tHL=CostMatrix(1,2);%Heavy-Large
tHS=CostMatrix(1,3);%Heavy-Small
tLH=CostMatrix(2,1);%Large-Heavy
tLL=CostMatrix(2,2);%Large-Large
tLS=CostMatrix(2,3);%Large-Small
tSH=CostMatrix(3,1);%Small-Heavy
tSL=CostMatrix(3,2);%Small-Large
tSS=CostMatrix(3,3);%Small-Small

%Matriz com a posiçao/identificação do aviao idAvioes
idAvioes(1:small,1)=1;   %Alocaçao na matriz idAvioes os avioes small
idAvioes(small+1:(small+large),1)=2; %Alocaçao na matriz idAvioes os avioes large
idAvioes((small+large+1):nAvioes,1)=3;%Alocaçao na matriz idAvioes os avioes heavy

%Small=1  Large=2  Heavy=3


%% Geração das posições iniciais
%Gerar posições aleatórias das posições das aeronaves a partir de idAvioes
r=(randperm(nAvioes))';
randAvioes =idAvioes(r(:,1),:);
%randAvioes=idAvioes; %Calculo da sequencia otima mínima
```

```
%% Categorização dos aviões e e cálculo dos tempos parciais e totais
%Nomeação das categorias das aeronaves na matriz das posições aleatórias
for i = 1:nAvioes
  if (randAvioes(i,1)==1)
      PosAvioesInicial(i,1)="Small";
  elseif (randAvioes(i,1)==2)
      PosAvioesInicial(i,1)="Large";
  elseif (randAvioes(i,1)==3)
      PosAvioesInicial(i,1)="Heavy";
  end
end

%Alocação das Categorias dos Avioes consoante a aeronave seguinte
for i = 1:nAvioes-1
  if (randAvioes(i,1)==1) && (randAvioes(i+1,1)==1)
      CatAvioes(i+1,1)=11;
  elseif (randAvioes(i,1)==1) && (randAvioes(i+1,1)==2)
      CatAvioes(i+1,1)=12;
  elseif (randAvioes(i,1)==1) && (randAvioes(i+1,1)==3)
      CatAvioes(i+1,1)=13;
  elseif (randAvioes(i,1)==2) && (randAvioes(i+1,1)==1)
      CatAvioes(i+1,1)=21;
  elseif (randAvioes(i,1)==2) && (randAvioes(i+1,1)==2)
      CatAvioes(i+1,1)=22;
  elseif (randAvioes(i,1)==2) && (randAvioes(i+1,1)==3)
      CatAvioes(i+1,1)=23;
  elseif (randAvioes(i,1)==3) && (randAvioes(i+1,1)==1)
      CatAvioes(i+1,1)=31;
  elseif (randAvioes(i,1)==3) && (randAvioes(i+1,1)==2)
      CatAvioes(i+1,1)=32;
  elseif (randAvioes(i,1)==3) && (randAvioes(i+1,1)==3)
      CatAvioes(i+1,1)=33;
  end
end

%Alocacação dos tempos consoante a catergoria do aeronave e matriz custo
for ii = 1:nAvioes
  if CatAvioes(ii,1)==11
      TempAvioes(ii,1)=tSS;
  elseif CatAvioes(ii,1)==12
      TempAvioes(ii,1)=tSL;
  elseif CatAvioes(ii,1)==13
      TempAvioes(ii,1)=tSH;
  elseif CatAvioes(ii,1)==21
      TempAvioes(ii,1)=tLS;
  elseif CatAvioes(ii,1)==22
      TempAvioes(ii,1)=tLL;
  elseif CatAvioes(ii,1)==23
      TempAvioes(ii,1)=tLH;
  elseif CatAvioes(ii,1)==31
      TempAvioes(ii,1)=tHS;
  elseif CatAvioes(ii,1)==32
      TempAvioes(ii,1)=tHL;
  elseif CatAvioes(ii,1)==33
      TempAvioes(ii,1)=tHH;
  end
end

tempoTotalInicial=sum(TempAvioes); %Calculo do tempo total necessário para os nAvioes↙
incialmente
```

```
% A funçao objetivo é minimizar o tempo total

AvioesInicial= [PosAvioesInicial];
tempoTotalFinal=tempoTotalInicial;%Apenas para gerar a variável


n=0;
%% Inicio das Iterações
while n < iteracoes
    n=n+1;
PositionShift=randi([-MPS MPS],nAvioes,1);

%Valores de mudança de posição da posição 1 até MPS não podem ser
%negativos, exceto o posição-shift maior ou igual ao MPS
for e = 1:MPS

if -PositionShift((e),:) - MPS <= 0
    PositionShift((e),:) = abs(PositionShift((e),:));
  else
end

end

PositionShift(1,1) = abs(PositionShift(1,1)); %Valor de shift da posição 1 não pode↙
ser negativo


PosAvioesCiclo=randAvioes;
for ee = 1:nAvioes
    xx=ee+PositionShift(ee,:);
   if xx<=nAvioes
    PosAvioesTemp=PosAvioesCiclo;
    PosAvioesCiclo(ee,:) = PosAvioesCiclo(xx,:);
    PosAvioesCiclo(xx,:) = PosAvioesTemp(ee,:);
   elseif xx>nAvioes
       xe=(xx-nAvioes);
       PosAvioesCicloTemp(ee+xe,:) = PosAvioesCiclo(ee,:);
       PosAvioesCicloTemp(xe,:)=[];

   end
   PosAvioesCicloTemp=PosAvioesCiclo;
end
PosAvioesCiclo=PosAvioesCicloTemp;
%[a,b]=groupcounts(PosAvioesCiclo) %Verificação se o numero de categorias se mantem


for i = 1:nAvioes-1
  if (PosAvioesCiclo(i,1)==1) && (PosAvioesCiclo(i+1,1)==1)
      CatAvioesCiclo(i+1,1)=11;
  elseif (PosAvioesCiclo(i,1)==1) && (PosAvioesCiclo(i+1,1)==2)
      CatAvioesCiclo(i+1,1)=12;
  elseif (PosAvioesCiclo(i,1)==1) && (PosAvioesCiclo(i+1,1)==3)
      CatAvioesCiclo(i+1,1)=13;
  elseif (PosAvioesCiclo(i,1)==2) && (PosAvioesCiclo(i+1,1)==1)
      CatAvioesCiclo(i+1,1)=21;
  elseif (PosAvioesCiclo(i,1)==2) && (PosAvioesCiclo(i+1,1)==2)
      CatAvioesCiclo(i+1,1)=22;
  elseif (PosAvioesCiclo(i,1)==2) && (PosAvioesCiclo(i+1,1)==3)
      CatAvioesCiclo(i+1,1)=23;
  elseif (PosAvioesCiclo(i,1)==3) && (PosAvioesCiclo(i+1,1)==1)
```

```
        CatAvioesCiclo(i+1,1)=31;
    elseif (PosAvioesCiclo(i,1)==3) && (PosAvioesCiclo(i+1,1)==2)
        CatAvioesCiclo(i+1,1)=32;
    elseif (PosAvioesCiclo(i,1)==3) && (PosAvioesCiclo(i+1,1)==3)
        CatAvioesCiclo(i+1,1)=33;
    end
end

for ii = 1:nAvioes
    if CatAvioesCiclo(ii,1)==11
        TempAvioesCiclo(ii,1)=tSS;
    elseif CatAvioesCiclo(ii,1)==12
        TempAvioesCiclo(ii,1)=tSL;
    elseif CatAvioesCiclo(ii,1)==13
        TempAvioesCiclo(ii,1)=tSH;
    elseif CatAvioesCiclo(ii,1)==21
        TempAvioesCiclo(ii,1)=tLS;
    elseif CatAvioesCiclo(ii,1)==22
        TempAvioesCiclo(ii,1)=tLL;
    elseif CatAvioesCiclo(ii,1)==23
        TempAvioesCiclo(ii,1)=tLH;
    elseif CatAvioesCiclo(ii,1)==31
        TempAvioesCiclo(ii,1)=tHS;
    elseif CatAvioesCiclo(ii,1)==32
        TempAvioesCiclo(ii,1)=tHL;
    elseif CatAvioesCiclo(ii,1)==33
        TempAvioesCiclo(ii,1)=tHH;
    end
end

tempoTotalCiclo=sum(TempAvioesCiclo);

if tempoTotalCiclo <= tempoTotalFinal
    tempoTotalFinal =tempoTotalCiclo;
    PosAvioesFinal=PosAvioesCiclo;
else
end

end %end final

%% Display das Soluções

for iv = 1:nAvioes
    if (PosAvioesFinal(iv,1)==1)
        AvioesFinal(iv,1)="Small";
    elseif (PosAvioesFinal(iv,1)==2)
        AvioesFinal(iv,1)="Large";
    elseif (PosAvioesFinal(iv,1)==3)
        AvioesFinal(iv,1)="Heavy";
    end
end

%Calculo na melhoria de tempo
otimizacao=(-((tempoTotalFinal-tempoTotalInicial)/tempoTotalInicial)*100);

disp('O número de aviões "small" foi')
disp(small)

disp('O número de aviões "large" foi')
disp(large)
```

```
disp('O número de aviões "heavy" foi')
disp(heavy)

disp('O tempo inicial foi de (segundos)')
disp(tempoTotalInicial)

disp('O tempo final foi de (segundos)')
disp(tempoTotalFinal)

disp('A redução de tempo foi de (segundos)')
disp(tempoTotalInicial-tempoTotalFinal)

disp('A melhoria foi de (%) ')
disp(otimizacao)

disp('Tempo CPU (segundos)')

toc
%Fim do algoritmo
```

UNIVERSIDADE DOS AÇORES
Faculdade de Economia e Gestão

Rua da Mãe de Deus
9500-321 Ponta Delgada
Açores, Portugal

2023

DM

Aplicação de modelos de otimização em infraestruturas aeroportuárias: Gestão operacional das chegadas

José Manuel Araújo Mendes