

Otimização de Aterragem de Aviões

Relatório Final

Inteligência Artifical

Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:

Carlos Teixeira – 201107928 – ei11145@fe.up.pt

Leonel Araújo – 201100640 – ei11130@fe.up.pt

Pedro Silva – 201109244 – ei11061@fe.up.pt

# Índice

[Introdução 3](#_Toc388737167)

[Objetivo 3](#_Toc388737168)

[Descrição do Problema 3](#_Toc388737169)

[Algoritmos Implementados 4](#_Toc388737170)

[Algoritmo Genético 4](#_Toc388737171)

[Algoritmo De Arrefecimento Simulado 4](#_Toc388737172)

[ALGORITMO DE CUSTO UNIFORME 4](#_Toc388737173)

[Restrições 5](#_Toc388737174)

# Introdução

Seja sobre a forma de dispositivos avançados, como smartphones, tablets ou computadores ou embebidos em dispositivos mais tradicionais como eletrodomésticos e televisões, o nosso quotidiano inclui cada vez mais o contacto com agentes inteligentes.

Este enraizamento dos agentes inteligentes suporta a necessidade e a importância de aprofundamento e estudo dos mesmos para que os possamos entender e melhorar de forma a facilitar as mais variadas atividades do nosso dia-a-dia.

Assim sendo, propusemos desenvolver um projeto em C++ que fará uso das técnicas ligadas à inteligência artificial (Algoritmo Genéticos, Arrefecimento Simulado e Custo Uniforme) com o intuito de aprofundar os conteúdos teóricos abordados na unidade curricular.

Este projeto consiste no desenvolvimento de algoritmos de otimização para um problema de escalonamento relacionado com aterragens de aviões.

# Objetivo

Tal como referido anteriormente, o presente projeto permite obter soluções a problemas de escalonamento de aterragens de vários aviões numa pista de aeroporto.

O escalonamento procura minimizar o custo da aterragem de cada avião. Este custo é calculado de acordo com uma função definida para cada avião que depende da hora a que o avião aterra, da hora preferencial para a aterragem e do custo adicional quando o avião aterra antes ou depois da hora preferencial.

Assim sendo, o objetivo principal do projeto passa por apresentar soluções utilizando diferentes algoritmos. No final, pretende-se comparar os resultados obtidos estabelecendo relações entre a eficiência dos vários métodos implementados.

# Descrição do Problema

No problema de otimização no escalonamento da aterragem de aviões cada avião tem uma hora preferencial pi e uma janela temporal [ai, bi] que limita as suas possibilidades de aterragem. Estas restrições partem de fatores externos ao problema, por exemplo: quantidade disponível de combustível. Estes valores são passados como parâmetro ao programa.

Como já referido anteriormente, é utilizada uma função de custo ramificada que varia consoante a hora de aterragem. A esse custo fica, assim, associado um αi determina o custo por unidade de tempo de uma aterragem antecipada, βi determina o custo por unidade de tempo de uma aterragem retardada.

Para além destes dois fatores, existe ainda um período δi durante o qual a pista não pode ser utilizada, dependente do tipo de avião que acabou de aterrar.

Assim sendo, cada avião cuja aterragem necessita de escalonamento deve ser representada por: voo(i,ai,pi,bi,αi,βi, δi).

# Algoritmos Implementados

Para contextualizar o leitor relativamente ao procedimento de cada um dos algoritmos e para uma melhor interpretação dos resultados que obtivemos, iremos abordar, nesta secção, o teor de cada um dos algoritmos implementados.

## Algoritmo Genético

O primeiro algoritmo implementado foi o Algoritmo Genético.

Este agloritmo baseia-se na geração inical de uma população de possíveis soluções (cromossomas) de forma aleatória.

Membros da população são combinados entre si de acordo com uma função de avaliação de forma a criar novas soluções as quais passam para a geração seguinte. Este processo repete-se e, adicionalmente, em cada geração são causadas pequenas mutações em alguns dos cromossomas as quais permitem criar um espaço de soluções mais variado. Ao longo das várias gerações espera-se que a população evolua de forma a obter uma solução próxima da solução ótima .

## Algoritmo De Arrefecimento Simulado

Um outro método de otimização utilizado para a resolução do problema apresentado passa pela implementação do Algoritmo de Arrefecimento Simulado (Simulated Annealing).

O Arrefecimento Simulado é uma metaheurística de otimização que consiste numa técnica de procura local probabilística para localizar uma boa aproximação de um ótimo global. Esta pesquisa é feita sobre um grande espaço de busca e fundamenta-se numa analogia com a termodinâmica - mais especificamente, com um processo térmico utilizado na metalúrgica denominado "annealing". Para tal, é utilizada uma variável “temperatura” que vai diminuindo gradualmente a cada iteração. Esta variável tem ainda um papel importante na validação da solução em cada iteração, embora esta validação tenha ainda um fator aleatório associado.

## Algoritmo de Custo Uniforme

Por fim, o último algoritmo a implementar passa por uma estratégia de pesquisa de Custo Uniforme.

Esta pesquisa trata-se de uma pequena modificação ao método de pesquisa em largura. Enquanto que na pesquisa em largura o primeiro nó a ser expandido é o primeiro filho do estado atual, na pesquisa de custo uniforme o primeiro nó a ser expandido é o que tem menor custo.

Assim sendo, a primeira solução a ser encontrada será, obrigatoriamente, a menos custosa.

# Restrições

Para agilizar os algoritmos e facilitar a tendência para uma solução final, são utilizadas penalizações ao custo de uma solução quando esta trespassa determinadas restrições relacionados com o problema em questão.

A penalização associada a cada restrição é igual para todos as metodologias usadas, facilitando a comparação de soluções geradas por diferentes algoritmos.

As restrições encontram-se divididas em dois principais grupos:

**Soft-Constraints**: são restrições que não impedem a praticabilidade da solução, mas acarretam um custo adicional relativo à solução ótima.

* Um avião aterra antes da hora preferencial.
* Um avião aterra depois da hora preferencial.

**Hard-Constraints**: são restrições que invalidam uma solução, isto é, não é possível pô-la em prática.

* Um avião aterra no intervalo de tempo de descanso da pista.
* Um avião aterra na mesma hora que outro avião.

Relativamente às Soft-Constraints, o valor de penalização utilizado é o estipulado pelo enunciado: quando se dá uma aterragem fora de horas o custo adicional é o fator associado a uma aterragem atrasada/adiantada a multiplicar pelo número de unidades de tempo a castigar.

Já as Hard-Constraints, desde a entrega intercalar, sofreram alterações que fizeram com que o incumprimento seja castigado mais severamente.

# Estruturas de Dados

Nesta secção iremos apresentar a estrutura de dados do programa que foi desenvolvido. Esta revisão compreende classes, atributos e funções utilizadas para cada um do algoritmos.

A estrutura de dados consiste em três grandes componentes:

## Avião

A primeira estrutura de dados representa um avião e para este definimos as seguintes variáveis :

- **Hora preferencial de aterragem(HPA)** : hora a que o avião pretende aterrar com custo de aterragem igual a zero.

- **Janela temporal :** intervalo de tempo que o avião tem para fazer a aterragem.

- **Valor da função de custo de aterragem retardada(VCAR) :** valor utilizado para calcular o custo de uma aterragem após HPA.

- **Valor da função de custo de aterragem adiantada(VCAA) :** valor utilizado para calcular o custo de uma aterragem antes de HPA.

- **Perído de ocupação da pista :** após a aterragem existe um período de tempo durante o qual mais nenhum avião poderá aterrar.

E o seguinte método:

-**Função de custo:** A função de custo é computada de acordo com a fórmula:

## Algoritmo

Cada um dos algoritmos implementados utiliza uma classe homónima para diferenciar as chamadas e permitir que todos possam correr num único executável.

### Algoritmo Genético

A implementação do Algoritmo Genético passa pela utilização das seguintes estruturas:

**Cromossoma** - servindo como interface para a classe "AlgoritmoGenetico" a classe **Cromossoma** possui aqueles que são os principais operadores usados na implementação de um algoritmo genético nomeadamente:

* **mutar**: cria uma mutação na solução;
* **reproduzir**: cria 2 novas soluções constituídas por material genético de dois progenitores
* **obterValor**: obtém o valor heurístico da solução

**Algoritmogenético** - procura-se com esta classe criar uma representação genérica de um algoritmo genético independente do tipo de cromossoma. Esta classe define e implementa os seguintes métodos:

* **cicloDeVida**: cria uma nova população de soluções de acordo com os operadores genéticos definidos em cromossoma
* **fazerIterações**: faz um número variável de ciclos de vida
* **obterMaisBemAdaptado**: obtém na população o cromossoma de melhor valor

### Algoritmo de Arrefecimento Simulado

O Arrefecimento Simulado, por sua vez, apenas utiliza a classe **Solução** para organizar num vetor o caminho seguido até à solução final.

Para além desse vetor de instanciações da classe **Solução** a implementação do Arrefecimento Simulado utiliza os seguintes campos:

* **fatorReducao**: contém o fator de redução a aplicar à temperatura a cada iteração;
* **temperatura**: guarda a temperatura a dado momento do algoritmo;
* **variacaoEnergia**: guarda a variação que ocorreu na energia de uma solução em determinada iteração;
* **solucaoInicial**: contém uma instanciação da classe **Solução** que reflete a solução utilizado no início da iteração;
* **solucaoAtual**: contém uma instanciação da classe **Solução** que reflete a solução após ser submetida a uma perturbação;

No que toca a funções, a classe que executa o Arrefecimento Simulado tem as seguintes funções auxiliares:

* **geraEstadoInicial():** que gera um estado inicial aleatório a ser usado na primeira iteração;
* **perturbacao():** que exerce uma perturbação na solução da iteração atual;
* **condExpAleatoria():** que calcula a expressão aritmética de aceitação da perturbação efetuada. Tem em conta a variação de energia e a temperatura atual.

### Algoritmo de Custo Uniforme

O algoritmo Custo Uniforme faz uso de duas estruturas **Node** e **TimeInterval** assim como da estrutura **Avião** supramencionada para a definição do seu espaço de soluções.

**Node -** esta estrutura de dados é usada para representar um nó no grafo a ser precorrido pelo algoritmo, a esta estrutura estão definidos os seguintes campos:

* **level**: Campo utilizado para especificar a que profundidade da raiz do grafo, o nó se encontra.
* **departTime**: Valor contendo a hora para o aviao contido neste nó (ver “plane” abaixo).
* **parent**: Apontador usado para fazer a ligação entre o nó atual e o seu pai, efetivamente funcionando como uma aresta no grafo.
* **branches**: Vetor de apontadores para os nós-filhos de um determinado nó, tendo a mesma funcionalidade que o campo parent (usado para navegar no grafo de pai para filho).
* **plane**: Apontador para o avião a ser representado neste nó, contêndo todas as suas caracteristicas já mencionadas.
* **restrictions**: Vetor de timeInterval usado para representar todas as restrições temporais que existem para chegar ao nó. Herda as restrições dos nós ancestrais para validar ou não uma possivel junção a outros nós.

**TimeInterval** - esta estrutura de dados é usada para representar o intervalo de tempo, ao qual um avião está associado. Isto é, indica o tempo em que o avião associado irá aterrar assim como o tempo que a pista está indesponível para utilização.

* **start**: Valor indicativo do inicio do periodo a que um avião está vinculado.
* **finish**: Valor correpondente ao fim do periodo ao qual o avião está associado.

## Solução

A representação da solução é dependente do método utilizado sendo implementados diferentes métodos de acordo com o algoritmo a que se destinam.

### Algoritmo Genético

A solução apresentada quando é corrido o Algoritmo Genético implementa para além do *container* do conjunto das aterragens, contém também variáveis que guardam a penalização referente à instanciação em causa e ao custo total da solução.

Todas estas metodologias e o *container* mencionado encontram-se na classe **GenSolucao**.

### Algoritmo de Arrefecimento Simulado

Tal como referido anteriormente aquando da especificação da estrutura, a implementação do algoritmo de Arrefecimento Simulado conta com um vetor de instanciações da classe **SimAnnSolucao** que guarda o caminho seguido pelo algoritmo.

Esta classe **SimAnnSolução** é a classe que guarda a informação relativa a cada solução, isto é, os calculos associados às penalizações e a ordem de aterragens dos aviões e da sua hora exata.

### Algoritmo de Custo Uniforme

Para representar a solução ao problema proposto, utilizámos, para o algoritmo Custo Uniforme, um vetor de apontadores de **Nodes**, sendo que este se encontra ordenado de fim para início pois é construido durante a execução do programa.

A cada elemento deste vetor o melhor tempo para o aviao contido no elemento encontra-se no campo *departTime*. A esta estrutura para se descobrir o custo associado à solução apenas é utilizar a função *getTotalCost* do primeiro elemento, sendo que este navega posteriormente pelos seus antecessores somando o custo de cada *departTime*.

# Utilização da Aplicação

Ao correr a aplicação desenvolvida, é requerido ao utilizador um ficheiro de *inputs*. Este ficheiro contem toda a informação relativa aos aviões (hora preferencial, custo de aterragem atrasada, custo de aterragem adiantada, tempo de desocupação da pista após aterragem, etc.) que o algoritmo necessita .

Com a presente entrega, foram incluidos dois ficheiros junto do código com propostas de problemas que a aplicação pode correr (avioes1.txt e avioes2.txt).

Os ficheiros de *input* contém em cada linha a informação referente a um avião. Cada uma dessas linhas contém o nome do avião, a hora preferencial de aterragem, a hora inicial da janela de aterragem, a hora final da janela de aterragem, o fator de custo de aterragem adiantada, o custo de aterragem atrasada e tempo de repouso da pista, por esta ordem.

Assim sendo, um exemplo de uma linha desse ficheiro seria:

TAP-V0001 11 0 30 2 4 5

Após a leitura, com sucesso, de um ficheiro de inputs, o programa procede à execução dos algoritmos pela seguinte ordem:

1. Algoritmo Genético
2. Algoritmo de Arrefecimento Simulado
3. Alforitmo de Custo Uniforme

Os dois primeiros algoritmos apresentam uma barra de progresso e possuem um ficheiro de configurações que pode ser encontrado na pasta *config.*

O ficheiro de configuração do algoritmo genético permite alterar informações relativamente ao número de iterações a efetuar, a utilização de uma política elitista, o tamanho da população e a probabilidade de mutação.

Já o ficheiro de configuração do algoritmo de arrefecimento simulado fornece ao programa as informações acerca do número de iterações máximo a efetuar, número máximo de perturbações sobre a solução de uma dada iteração e o fator de redução da temperatura.

# Representação das Soluções

Falar no JS e HTML

# Conclusão

Foi fixe o trabalho

# Recursos Utilizados

Livros fixes