



Universidade do Porto
Faculdade de Engenharia
FEUP

OTIMIZAÇÃO DE ATERRAGEM DE AVIÕES

RELATÓRIO FINAL

INTELIGÊNCIA ARTIFICIAL

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO

2013-2014

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

Índice	2
Introdução	3
Objetivo	3
Descrição do Problema.....	3
Algoritmos Implementados	4
Algoritmo Genético	4
Algoritmo de Arrefecimento Simulado	4
Algoritmo de Custo Uniforme	4
Restrições.....	5
Estruturas de Dados	5
Avião.....	5
Algoritmo.....	6
Algoritmo Genético.....	6
Algoritmo de Arrefecimento Simulado	6
Algoritmo de Custo Uniforme	7
Solução	7
Algoritmo Genético.....	8
Algoritmo de Arrefecimento Simulado	8
Algoritmo de Custo Uniforme	8
Ficheiros de Input e Configurações.....	8
Representação das Soluções	9
Análise dos Resultados Obtidos.....	11
Conclusão	12
Recursos Utilizados.....	13
Apêndice.....	14
Manual de Instrução	14

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 possamos entender e melhorar estes agentes 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 p_i e uma janela temporal $[a_i, b_i]$ 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 α_i que determina o custo por unidade de tempo de uma aterragem antecipada e um β_i que determina o custo por unidade de tempo de uma aterragem retardada.

Para além destes dois fatores, existe após a aterragem de um avião um período δ_i 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: $voo(i, a_i, p_i, b_i, \alpha_i, \beta_i, \delta_i)$.

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 algoritmo baseia-se na geração inicial 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 convergência para uma solução final, são utilizadas penalizações ao custo de uma solução quando esta trespasa 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 soluções que violem Hard-Constraints são penalizadas mais acentuadamente de forma a invalidar essas soluções.

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 dos 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.

- **Período 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:

$$C(t) = \begin{cases} VCAA * (HPA - t), & t < HPA \\ VCAR * (HPA - t), & t > HPA \\ 0, & t = HPA \end{cases}$$

ALGORITMO

Cada um dos algoritmos implementados utiliza uma classe homônima 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 percorrido 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, contendo todas as suas características 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 possível 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 indisponível 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.

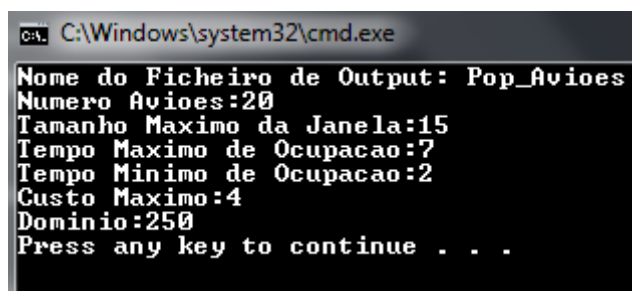
ALGORITMO DE CUSTO UNIFORME

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 construído durante a execução do programa.

A cada elemento deste vetor o melhor tempo para o avião 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*.

FICHEIROS DE INPUT E CONFIGURAÇÕES

Em conjunto com o programa em si, disponibilizamos um *spawner* de populações que podem ser utilizadas como *input*. Para tal basta correr o executável “*spawner.exe*” e inserir as configurações requeridas pelo programa para produzir uma população (número de aviões, tamanho máximo da janela de aterragem, valores máximo e mínimo do tempo de ocupação da pista, custo das aterragens fora de horas e nome do ficheiro).



```
C:\Windows\system32\cmd.exe
Nome do Ficheiro de Output: Pop_Avioes
Numero Avioes:20
Tamanho Maximo da Janela:15
Tempo Maximo de Ocupacao:7
Tempo Minimo de Ocupacao:2
Custo Maximo:4
Dominio:250
Press any key to continue . . .
```

Ilustração 1 – Exemplo da utilização do spawner

Ao correr a aplicação desenvolvida, é requerido ao utilizador um ficheiro de *inputs*. Este ficheiro deve conter toda a informação relativa aos aviões (hora preferencial, custo de aterragem atrasada, custo de aterragem adiantada, tempo de ocupação da pista após aterragem, etc.) que os algoritmos necessitam.

Com a presente entrega, foram incluídos 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. Algoritmo de Custo Uniforme

Os dois primeiros algoritmos apresentam uma barra de progresso e todos os algoritmos 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.

O único parametro controlado pelo ficheiro de configurações do algoritmo de Custo Uniforme é o tempo máximo de execução do algoritmo.

REPRESENTAÇÃO DAS SOLUÇÕES

No que toca à representação de soluções, o grupo procurou uma opção que permitisse transmitir a ideia da progressão da ocupação da pista sobre um plano temporal. Por outras palavras, procurámos uma representação que permitisse identificar facilmente não só sobreposições temporais, mas também as horas de desapertamento da pista.

Dadas estas prioridades, o grupo achou que a melhor forma seria procurar uma solução sobre a forma de recta real que seguisse a variável que representa o tempo. Para tal, optámos por utilizar os nossos conhecimentos em HTML e Javascript para simular essa

recta real e animar a posição que o avião ocupará na recta e o respectivo tempo de ocupação da pista.



Ilustração 2 - Exemplo das soluções geradas por cada algoritmo.

Assim sendo, é necessário guardar a informação num ficheiro HTML. Este registo é feito quando chegamos a uma solução final, nesta altura, um array JSON com a informação dessa solução é gerado.

Esta informação é impressa no ficheiro com o mesmo nome do ficheiro de *input* concatenado com a expressão "results.html" dentro de um *div* criado para o efeito, ou seja, cada um dos algoritmos implementados imprime as informações relativas à sua solução apenas no seu *div* correspondente.

Este ficheiro HTML ao ser corrido irá utilizar o ficheiro "results.js" que trata a informação de cada "div" devidamente e desenha a recta esperada.

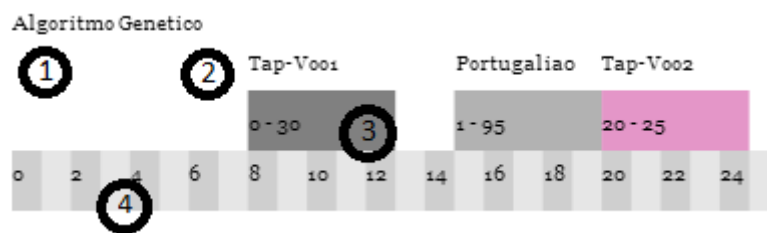


Ilustração 3 – Exemplo dos elementos que mostram uma solução.

Com o intuito de ajudar a perceber melhor o que cada elemento da página representa, segue abaixo uma breve explicação sobre os mesmos:

1. Nome do algoritmo cuja a representação é mostrada abaixo.
2. Nome do avião.

3. Espaço colorido mostra o tempo que o avião vai ocupar a pista (neste caso de 8 a 13) e a indicação numérica do intervalo de tempo que o avião pode aterrar (0 a 30 para o exemplo dado).
4. Recta temporal.

As cores de cada avião são atribuídas aleatoriamente.

ANÁLISE DOS RESULTADOS OBTIDOS

Após correremos os algoritmos implementados e registarmos os dados obtidos e elaboramos um conjunto de gráficos a estes referentes. Estes gráficos ajudam-nos a perceber melhor o desempenho de cada um dos algoritmos e a importância das suas variáveis.

Todos os testes abaixo foram executados com uma população de 10 aviões. Os ficheiros relativos às estatísticas abaixo podem ser encontrados no apêndice.

No contexto do Algoritmo Genético, sem adotar uma política elitista, podemos observar os seguintes resultados:

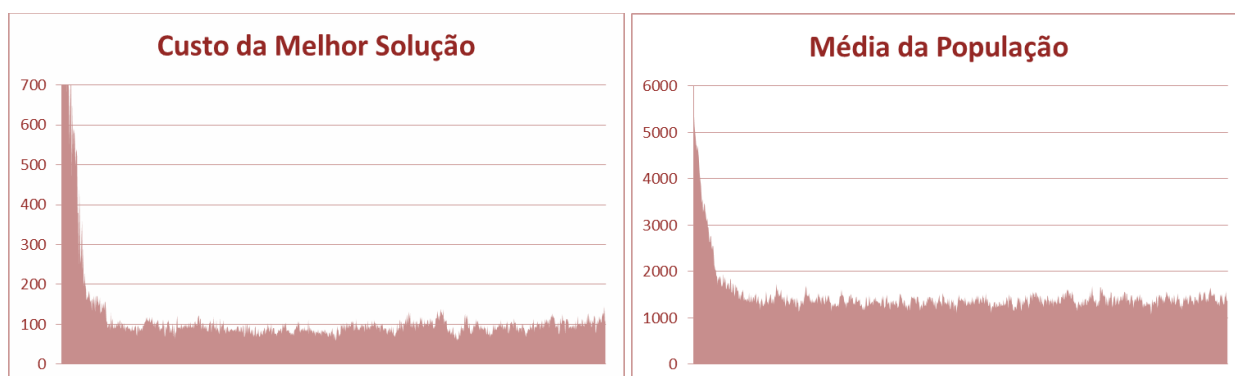


Ilustração 4- Gráfico que relaciona Número Iterações com o Custo Solução.

Uma análise superficial dos dados permite-nos concluir que o algoritmo genético, sem utilizar uma política elitista, após uma rápida descida no valor do custo da solução, estabiliza com algum ruído. O que contrasta com o mesmo algoritmo quando este adota uma política elitista.

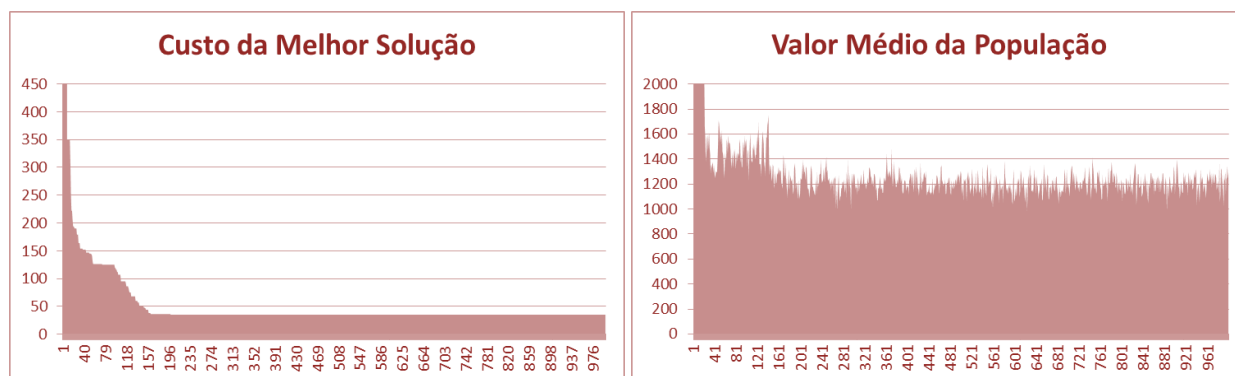


Ilustração 5 - Algoritmo genético, mas, desta feita, com a utilização de uma política elitista.

Seguem, agora, as conclusões acerca do gráfico referente aos resultados obtidos com o Algoritmo de Arrefecimento Simulado.

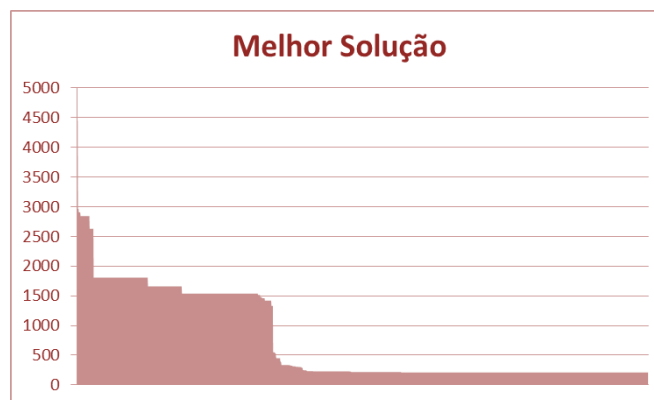


Ilustração 6 – Relação do custo da solução com o número de iterações realizadas utilizando o Algoritmo de Arrefecimento Simulado com 3000 iterações.

Como podemos verificar no gráfico acima, a partir de um certo número de iterações (~1400) nenhuma das perturbações exercidas sobre a solução altera o valor do custo da melhor solução, isto é, o algoritmo tem dificuldade em exercer uma perturbação capaz de melhorar a solução atual.

De notar ainda que as perturbações que trouxeram uma otimização à solução são facilmente identificáveis, e que com o “arrefecimento” estamos a tender para uma solução com custo cada vez inferior.

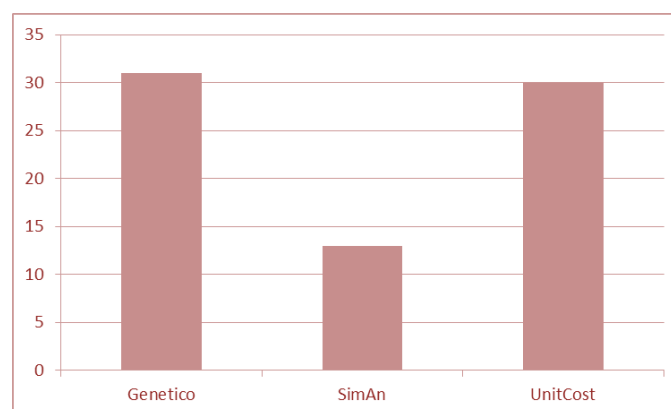


Ilustração 7 - Duração dos algoritmos nos testes acima.

Este último gráfico mostra o tempo que os algoritmos levaram a correr a população passada por parâmetro. De notar que o algoritmo de Custo Uniforme não apresentou uma solução final dentro do tempo limite definido no ficheiro de configuração (30 segundos para o teste efectuado) e apenas mostrou uma solução com 6 dos aviões (apenas desceu 6 níveis na árvore de busca).

Todos estes valores são gerados quando o utilizador executa os algoritmos sobre uma população, isto é, quando corre a aplicação. Os ficheiros “.csv” correspondentes a estes dados podem ser encontrados na pasta “Solucao”.

Os resultados obtidos permitem-nos comprovar o poder das meta-heurísticas comparativamente a estratégias de brute-force, que não são capazes de lidar com o grande número de estados associados a este tipo de problemas.

CONCLUSÃO

Finalizando e tendo em consciência todo trabalho desenvolvido ao longo do semestre, assumimos, sobretudo, um balanço positivo no que toca à consolidação de conhecimentos obtidos e no domínio sobre problemas de escalonamento.

Uma possível extensão não implementada do problema passaria por alargar os algoritmos à possibilidade de existência de mais que uma pista.

De salientar a familiarização com os algoritmos de otimização que implementámos e as conclusões obtidas relativamente aos mesmos.

Em suma, tendo em conta que a área desenvolvida é uma área bastante pertinente e com aplicações viáveis na vida real, consideramos uma mais valia ter explorado estes conceitos.

RECURSOS UTILIZADOS

No desenvolvimento deste projeto utilizamos, para além dos slides das aulas teóricas, o seguinte leque de manuais:

- Stuart Russell, Peter Norvig; Artificial intelligence. ISBN: 978-0-13-207148-2
- Ernesto Costa e Anabela Simões; Inteligência artificial. ISBN: 972-722-269-2

De notar ainda o apoio prestado pelo monitor da disciplina Tiago Azevedo.

Resta apenas acrescentar que todos nós partilhamos da opinião que a distribuição de tarefas foi equitativa.

APÊNDICE

MANUAL DE INSTRUÇÃO

Para correr a aplicação, apenas é necessário executar o ficheiro “iart.exe”.

Este executável irá solicitar ao utilizador um ficheiro com as informações relativas aos aviões a aterrar numa dada pista. Um ficheiro deste tipo pode ser facilmente gerado utilizando o executável “spawner”, onde o utilizador poderá especificar várias opções para a sua população aleatória.

Este “spawner” contém, no entanto, limitações relativamente aos *inputs* que lhe são dados:

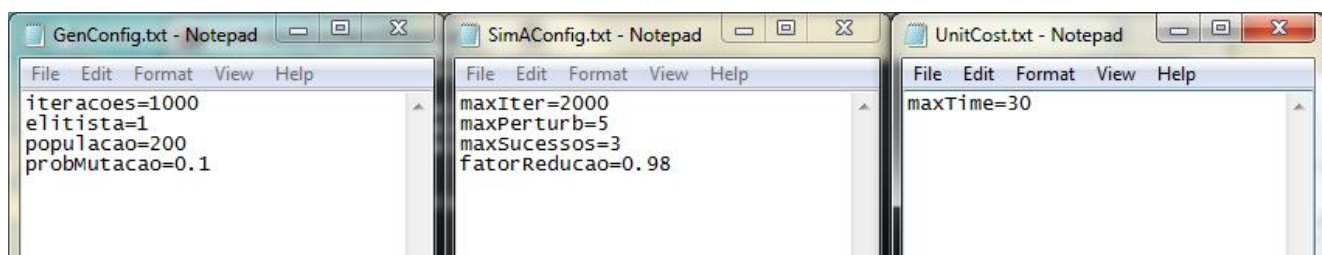
- O domínio deve ter um valor, igual ou superior ao produto do número de aviões pelo tamanho máximo da janela de aterragem;
- O tempo máximo de ocupação de ocupação deve ser superior ao tempo mínimo de ocupação.
- Todos os valores introduzidos devem ser positivos.

De seguida, os algoritmos são corridos sequencialmente, e os resultados podem ser consultados na pasta “Solucao”.

Para visualizar a solução basta abrir o ficheiro html num *browser*.

FICHEIROS UTILIZADOS NOS TESTES

Ficheiros de Configuração:



População:

The image shows a Notepad window titled 'avioes.txt' containing a table of aircraft data. The table has 10 rows and 7 columns: Nome, HoraPreferencial, HoraJanInicial, HoraJanFim, CustoAterragemAntecipada, CustoAterragemRetardada, and TempoReposoPista.

Nome	HoraPreferencial	HoraJanInicial	HoraJanFim	CustoAterragemAntecipada	CustoAterragemRetardada	TempoReposoPista
Aviao1	11	0	30	2	4	5
Aviao2	22	20	25	3	5	2
Aviao3	33	30	45	4	6	4
Aviao4	44	40	65	5	7	4
Aviao5	55	40	75	6	8	5
Aviao6	66	50	95	7	9	6
Aviao7	77	20	95	7	8	2
Aviao8	88	54	95	7	7	5
Aviao9	98	80	99	7	6	3
Aviao10	13	1	95	7	5	5