Universidade do Porto

Faculdade de Engenharia

FEUP

Optimização da Aterragem de Aviões

Relatório Intercalar

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:

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

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

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

24 de Abril de 2014

ÍNDICE

Introdução 3

Objetivo 3

Descrição do Projeto 3

Algoritmos A Implementar 3

Algoritmo Genético 3

Algoritmo de Arrefecimento Simulado 4

Algoritmo de Custo Uniforme 4

Restrições 4

Estruturas de Dados 4

Avião 5

Algoritmo 5

Solução 6

Trabalho Efetuado 7

Resultados esperados 8

Conclusões 8

Recursos 8

INTRODUÇÃO

Quer sobre a forma de dispositivos avançados como smartphones, *tablets* ou computadores quer embebidos em dispositivos aparentemente mais simples como eletrodomésticos e televisões os agentes inteligentes ocupam atualmente um papel muito importante no nosso dia a dia.

Com o intuito de compreender como funcionam estes agentes e as bases teóricas que os suportam propusemos desenvolver um projeto em C++ o qual fará uso das técnicas ligadas à inteligência artificial(algoritmos genéticos, arrefecimento simulado e *Custo Uniforme*).

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

OBJETIVO

Tal como referido anteriormente, tencionamos desenvolver um projeto com o objetivo de escalonar aterragens de vários aviões numa pista de aeroporto. O escalonamento procurará minimzar o custo da aterragem de cada avião, o qual é calculado de acordo com uma função definida para cada avião que por sua vez depende da hora a que o avião aterra.

Assim sendo, esperamos ser capazes de, no final do projeto, apresentar soluções para o problema utilizando diferentes algoritmos e estabelecendo relações entre a eficiência dos vários métodos implementados.

DESCRIÇÃO DO PROJETO

No problema de otimização no escalonamaento da aterragem de aviões cada avião tera 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.

Para o efeito, 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 A IMPLEMENTAR

Nesta secção iremos prodeceder a uma breve explicação de o que consistem os algoritmos implementados para contextualizar o leitor relativamente ao procedimento para encontrar uma solução do problema.

- ALGORITMO GENÉTICO

O primeiro algoritmo a implementar é o Algoritmo Genético. Relativamente a este, inicialmente é criada 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 a utilizar 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 fulcral na validação da solução em cada iteração.

▪ 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.

As restrições encontram-se divididas em dois principais grupos:

Soft-Constraints: que são restrições que não impedem a realização da solução.

- Um avião aterra antes da hora preferencial;
- Um avião aterra depois da hora preferencial.

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

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

A cada restrição é aplicado um valor de penalização ao custo da solução. Os valores dessa penalização utilziados aquando desta entrega são ainda preliminares e estão ainda a ser estudados de forma refletir melhor a importância de cada restrição para que nos possamos aproxomar ainda mais de um cenário real.

ESTRUTURAS DE DADOS

Para uma mais sucinta interpretação do trabalho e da sua estrutura, apresentamos, de seguida, a estrutura adotada e uma breve explicação da mesma. Esta compreende 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:

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

- *ALGORITMO*

Cada um dos algoritmos implementados utiliza uma classe 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

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.

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.

Faz parte do esqueleto comum um recipiente "aterragens" que contém informação relativa às mesmas. Desta forma relacionamod um avião com uma hora da sua aterragem. Para além deste recipiente é também comum a todos os algoritmos um método capaz de calcular o valor de cada solução tendo em conta penalizações para *hard-constraints* e *soft-constraints*.

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.

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 Solucao que guarda o caminho seguido pelo algoritmo.

Quanto à classe **Solução** em si, o algoritmo de Arrefecimento Simulado apenas adiciona a função *intersectaIntervalo*() que se é utilizada para detetar sobreposições de aterragens.

Custo Uniforme

Para representar a solução ao problema proposto, utilizamos 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*.

TRABALHO EFETUADO

Encontram-se no momento presente implementadas versões preliminares dos três algoritmos em C++. As implementações ainda não apresentam as melhores soluções e, por conseguinte, necessitam de melhoramentos ao nível da otimização.

Algoritmo Genético

Neste momento encontram-se já implementadas todas as funcionalidades relativas à implementação do algoritmo genético faltando resolver apenas alguns conflitos com as várias versões da classe Avião das outras implementações e fazer alguns testes para assegurar o correto funcionamento do algoritmo.

Arrefecimento Simulado

Até ao presente ponto de avaliação, o algoritmo de Arrefecimento Simulado já foi correctamente implementado e é capaz de devolver uma solução boa para os problemas apresentados. No entanto, na nossa opinião, a implementação ainda pode ser mais agilizada e os resultados podem se aproximar ainda mais da solução ótima.

O programa não apresenta problemas no que toca a tempo de execução e corre sem problemas qualquer tipo de problema, sendo que, os problemas impossíveis apresentarão um custo elevado e imprático e terminarão com a restrição de número máximo de iterações.

Custo uniforme

O trabalho realizado neste algoritmo conta já com a especificação e correto funcionamento das estruturas de dados inerentes, assim como um esboço do preliminar do algoritmo, sendo que está já gera

ramificações apartir de um nó com as restrições inerentes ao próprio e avalia corretamente o custo da solução temporária encontrada.

Neste momento o algoritmo está a funcionar para os casos mais simples, isto é o algoritmo consegue calcular a solução, embora para um numero substâncial de aviões, cada um com um intervalo de tempo considerável, o programa torna-se bastante lento, pelo é necessário melhorar a performance a nível de código para evitar fazer a funções redundantes assim como possivelmente limitar mais à fila de espera que contêm os nós a serem percorridos pelo algoritmo. Embora o algoritmo em si, não seja eficaz para resolver o problema proposto, o grupo acredita que é possível melhorar a implementação atual.

RESULTADOS ESPERADOS

Espera-se que no final do projeto sejamos capazes de gerar soluções viáveis usando diferentes algoritmos com o objetivo de comparar a eficiência e validade das mesmas. Para tal, daremos prioridade à reavaliação do peso das restrições que estão a ser aplicadas.

Adicionalmente seria interessante criar uma interface que permitisse visualizar estatísticas sobre os diferentes algoritmos e o caminho que seguiram para alcançar a solução.

Para além destas interfaces, tencionamos criar um suporte para importação de aviões utilizando ficheiros de texto.

CONCLUSÕES

Relativamene ao presente projeto, assumimos um balanço extremamente positivo no que toca consolidação os conhecimentos obtidos e no aumento do domínio sobre problemas de escalonamento.

Conseguimos ainda familiarizarmo-nos com as diferenças entres os diferentes algoritmos e as respetivas implementações, desta feita, em C++.

Concluindo, esperamos não só conseguir cumprir as tarefas esperadas mas também melhorar as otimizações já implementadas relativamente ao problema dado.

RECURSOS

Para além dos slides das aulas teóricas que foram disponibilizados na página da unidade curricular, no desenvolvimento do projeto, utilizámos o Microsoft Visual Studio como ambiente de desenvolvimento integrado.

Adicionalmente, é de notar a importante orientação prestada pelo monitor da disciplina Tiago Azevedo.