Aterragem de Aviões: Um problema de optimização utilizando programação em lógica com restrições

Ana Sousa and Augusto Silva

FEUP - PLOG, Turma 3MIEIC1, Grupo 25

Resumo O problema proposto é o escalonamento das aterragens dos aviões de um aeroporto, este problema pode ser visto como um problema de optimização, visto que se pretende minimizar o custo das aterragens, e será resolvido em linguagem Prolog utilizando programação em lógica com restrições. O principal objetivo do trabalho é a implementação de uma solução que minimize ao máximo os custos mas que, ao mesmo tempo, seja eficiente, bem como a visualização da solução gerada em modo texto. Esta implementação faz uso das capacidades da programação em lógica de restrições, destacando-se o predicado *cumulatives*. Os resultados obtidos permitem verificar a eficiência do programa principalmente para casos de várias pistas de aterragens.

1 Introdução

Este trabalho tem como principal objetivo a resolução, em linguagem Prolog, de um problema de optimização utilizando programação em lógica com restrições. O problema de optimização em causa trata-se da escalonagem das aterragens dos aviões num determinado aeroporto, tentando minimizar o custo associado a cada aterragem. Este problema deve ser resolvido como um problema de satisfação de restrições, utilizando, como referido acima, programação em lógica com restrições, e deve ser possível resolver problemas com diferentes parâmetros, nomeadamente: maior ou menor número de voos, diferentes números de pistas e custos associados a aterragens fora dos tempos preferenciais.

Mesgarpout et al.[4], num estudo financiado pela organização europeia para a segurança da navegação aérea, expõem: as variáveis de decisão; os parâmetros a considerar; as funções objectivo(minimização de atraso médio ou maximização de utilização de pistas ou minimização de custo com combustível); as restrições a considerar (utilização de pistas, tempo de inutilização após aterragem causada por torbulência, tempo, etc). Referem ainda que soluções eficazes para o modelo permanecem por desenvolver devido às soluções terem de ser obtidas rapidamente e o problema ser complexo (*NP-hard*).

A secção 2 descreve o problema de otimização em causa. A secção 3 descreve os ficheiros de dados utilizados para efeitos de teste. A secção 4 descreve as variáveis de decisão bem como os seus domínios. A secção 5 descreve as restrições utilizadas e respetiva implementação. A secção 6 descreve a forma de avaliar a

solução obtida bem como a sua implementação. A secção 7 descreve a estratégia de etiquetagem no que diz respeito à ordenação de variáveis e valores. A secção 8 descreve os predicados necessários para visualizar a solução do problema em modo texto. A secção 9 descreve as soluções obtidas em casos práticos com diferetes complexidades e a respetiva análise. A secção 10 apresenta as conclusões do trabalho bem como as perspetivas de desenvolvimento.

2 Descrição do Problema

O problema a abordar consiste na optimização do escalonamento de aterragens de aviões no contexto de um aeroporto multipista. Cada avião tem uma janela temporal que limita as suas possibilidades de aterragem, ou seja, um tempo mínimo, máximo e preferencial. O desvio ao tempo preferencial implica um custo de evolução linear e com coeficiente diferente caso se trate de uma aterragem antecipada ou atrasada. Este custo deve-se ao acréscimo de consumo de combustível por aumento da velocidade ou pelo tempo de espera no ar.

O custo é expresso por:

$$\text{Custo} = \begin{cases} Cafter * (T - Tpref) & \text{se } T \geq Tpref \\ Cbefore * (Tpref - T) & \text{se } T < Tpref \end{cases}$$

em que *Cafter* é o custo por unidade de tempo de uma aterragem atrasada, *Cbefore* é o custo por unidade de tempo de uma aterragem antecipada, T é o momento temporal de aterragem e *Tpref* o momento temporal ideal para aterragem.

Após cada aterragem, existe um período durante o qual a pista não pode ser utilizada, dependente do avião que acabou de aterrar. A optimização dá-se pela minimização do custo total associado à totalidade das aterragens.

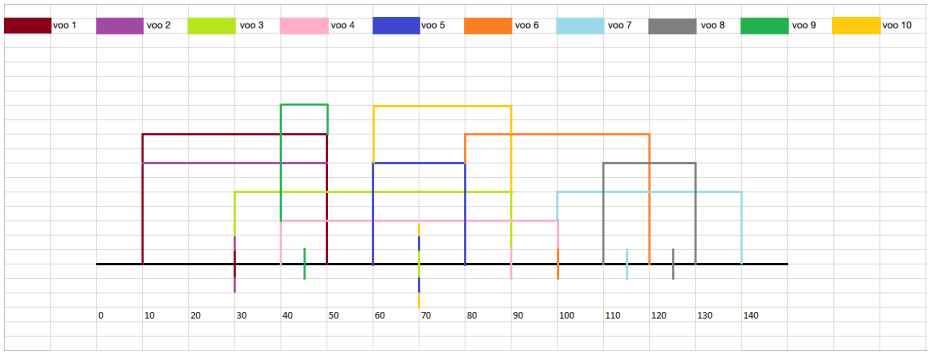

Figura 1: Distribuição temporal de um exemplo de conjunto de 10 aviões a aterrar.

Na figura 1 está representada a janela temporal (do tempo minímo ao tempo máximo) para aterragem de cada avião assim como o seu tempo preferencial, este último representado pelas marcas no eixo temporal. Este exemplo tem sobreposição de janelas temporais, sendo assim representativo de uma situação comum em aeroportos.

3 Ficheiros de Dados

Cada ficheiro de dados representa um grupo de aterragens a escalonar num aeroporto com um certo número de pistas.

Cada voo é representado por um predicado da forma:

flight (+Id, +Tmin, +Tpref, +Tmax, +Cbefore, +Cafter, +Tafter)

- *Id* indica o número de identificação atribuído a cada avião, deve ser único;
- *Tmin* indica o tempo mínimo de aterragem;
- *Tpref* indica o tempo de aterragem preferencial;
- *Tmax* indica o temo máximo para o início da aterragem;
- *Cbefore* indica o custo associado por cada unidade de tempo de uma aterragem antecipada;
- *Cafter* indica o custo associado por cada unidade de tempo de uma aterragem atrasada;
- *Tafter* indica o período de tempo durante o qual a pista não pode ser utilizada após a aterragem do avião, a considerar como duração da mesma.

Por ficheiro de texto, deve existir um, e só um, facto da forma:

numRunways (+N)

- *N* indica o número de pistas disponíveis no aeroporto.

```
numRunways(1).
flight(1,10,30,50,1,2,2).
flight(2,10,30,50,10,15,5).
flight(3,30,70,90,2,3,4).
flight(4,40,90,100,7,8,20).
flight(5,60,70,80,5,6,3).
flight(6,80,100,120,5,1,12).
flight(7,100,115,140,10,3,4).
flight(8,110,125,130,1,10,10).
flight(9,40,45,50,25,20,20).
flight(10,60,70,90,100,1,5).
```

Figura 2: Exemplo de base de dados

O exemplo na figura 2 corresponde à ilustração da figura 1.

4 Variáveis de Decisão

As variáveis de decisão a considerar para cada aterragem são:

- O momento em que avião aterra (S) que está compreendido entre Tmin e Tmax;

- O número da pista onde aterra (M) está compreendido entre um e o número de pistas;

Considera-se ainda uma variável global soma dos custos (SumCost) que varia entre 0 e SumCost.

5 Restrições

Para além dos domínios definidos para as variáveis de decisão indicadas na secção 4, assume especial importância a não sobreposição de aterragens na mesma pista em cada momento. Esta restrição é implementada com recurso ao predicado *cumulatives(+Tasks, +Machines, +Options)* da biblioteca *clpfd* do *Sicstus*[1]. Embora este predicado tenha sido pensado para atribuir tarefas a máquinas, é aplicável ao problema abordado, dado que este recebe uma lista de tarefas e uma lista de máquinas ao qual se faz corresponder, respetivamente, uma lista de voos e uma lista de pistas.

Cada tarefa é representada por *task(Oi,Di,Ei,Hi,Mi)* em que *Oi* representa o tempo de aterragem, *Di* o tempo que a pista deve estar livre após aterragem, *Ei* o tempo em que termina o período de não utilização da pista, *Hi* os recursos utilizados e *Mi* a pista onde a aterragem ocorre.

Cada pista é representada por *machine(Mj, Lj)* em que *Mj* é um número identificador da pista e *Lj* o número de recursos da mesma.

Para o problema a resolver cada aterragem fará uso de um só recurso e cada pista terá um só recurso disponível.

Utilizando a notação das secções anteriores, os termos tomam a forma
task(S,Tafter,E,1,M), sendo *E* igual à soma de *S* e *Tafter*
e
machine(M, 1).

O predicado cumulatives recebe ainda *Options*. Neste indica-se *bound(upper)*, isto é, o número de recursos de cada máquina deve ser considerado como o número máximo de recursos disponíveis.

6 Função de Avaliação

Tal como abordado na secção 2 e utilizando a notação introduzida na secção 3, cada solução é avaliada por uma função de custo *C* que é calculada da seguinte forma:

- se o tempo de aterragem *S* for inferior ao tempo preferencial *Tpref*, o custo será a igual ao resultado de *Cbefore * (Tpref - S)*;

- se o tempo de aterragem *S* for igual ao tempo preferencial *Tpref*, o custo será igual a 0;

- se *S* for superior a *Tpref*, o custo será a igual ao resultado de *Cafter * (S - Tpref)*.

O cálculo do custo é implementado por:

```
S #< Tpref #<=> B,
C #= Cbefore*(Tpref-S)*B + Cafter*(S-Tpref)*(1-B)
```

Pela utilização de materialização (*reification*), a função de custo é resumida numa única condição pois os termos *Cbefore * (Tpref-S) * B* e *Cafter * (S-Tpref) * (1-B)* tomam alternadamente o valor zero, conforme Hs é maior que Tpref ou menor.

Este custo será minimizado durante a pesquisa (ver secção 7).

7 Estratégia de Pesquisa

A etiquetagem recorre ao predicado *labeling*.

As variáveis que são diretamente consideradas são: o momento de aterragem para cada avião *(S)*, a pista de aterragem de cada avião *(M)* e a soma dos custos de todas as aterragens *(SumCost)*. Como opções de etiquetagem consideramos:

- selecção para atribuição: *leftmost*. Assim, serão primeiro processados os tempos de aterragem e só depois a pista onde irão aterrar.
- divisão do domínio de pesquisa: *bisect*. Assumindo que *Tpref* se aproximará do valor central de *[Tmin,Tmax]*, *bisect* permite uma poda maior. [2]
- obtenção de solução: *minimize(SumCost)*. Assim, será obtida a solução com um custo mínimo.
- limite de tempo: *timeout(Time,Flag)*. Assim, será obtida a solução mínima até ao momento de fim de processamento.

8 Visualização da Solução

O predicado principal a invocar para testar a implementação é *landing*. Este predicado, após encontrar a solução com recurso ao predicado *scheduling*, que implementa o que foi referido nas secções anteriores, invoca

showFlights (+Lflights, +Ltime , +Lcost , +Lrunways)

para a listagem de voos em formato de texto.

É indicado um resumo do ficheiro lido, o momento de aterragem escolhido para cada avião e a pista onde essa aterragem se realiza assim como o custo. No final são apresentadas a soma dos custos e estatísticas de execução.

As estatísticas de execução incluem:

- Número de *resumptions*;
- Número de implicações detectadas;
- Número de vezes que o domínio foi podado;
- Número de vezes que a execução regrediu;
- Número de restrições criadas;
- Tempo de execução.

```
----------------------
10 flight(s) scheduled to land on 1 runway(s).

Flight No 1 -> landing time: 28; runway 1; cost 2.
Flight No 2 -> landing time: 30; runway 1; cost 0.
Flight No 3 -> landing time: 66; runway 1; cost 8.
Flight No 4 -> landing time: 89; runway 1; cost 7.
Flight No 5 -> landing time: 70; runway 1; cost 0.
Flight No 6 -> landing time: 109; runway 1; cost 9.
Flight No 7 -> landing time: 121; runway 1; cost 18.
Flight No 8 -> landing time: 125; runway 1; cost 0.
Flight No 9 -> landing time: 45; runway 1; cost 0.
Flight No 10 -> landing time: 73; runway 1; cost 3.

Total Cost: 47.

Statistics:
Resumptions: 5507512
Entailments: 615770
Prunings: 2976134
Backtracks: 58809
Constraints created: 102
task took 3.540 sec.
```

Figura 3: Exemplo de visualização

O exemplo na figura 3 corresponde à ilustração da figura 1 e aos termos da figura 2.

9 Resultados

Opções de pesquisa selecionadas As opções de pesquisa e de restrição referidas nas secções 5 e 7, para os predicados *labeling/2* e *cumulatives/3*, foram confirmadas como sendo as mais eficientes pelo teste de várias combinações para diferente número de aviões e pistas.

Análise de Tempos de Execução A tabela 1 mostra os resultados obtidos para diferente número de aviões e pistas. Regista-se uma grande diferença nos tempos de execução nos casos em que é possível encontrar uma solução de custo 0. Isto deve-se ao facto de, uma vez obtido custo 0, a pesquisa parar pois o valor menor de custo já foi atingido. Nos casos em que o custo difere de 0, a pesquisa faz-se até que sejam encontradas todas as soluções para o problema.

Teste	Número de Voos	Número de Pistas	Custo	Tempo
A	4	1	10	0,02s
B	4	2	0	0,02s
C	4	3	0	0,01s
D	5	1	10	0,09s
E	10	4	0	0,38s
F	10	1	47	3,54s
G	20	8	0	2,97s
H	20	1	70	2088,170s
I	7	2	9	10,620s
J	7	1	93	0,040s

Tabela 1: Custo de aterragem obtido e tempo de execução de pesquisa em função do número de voos e pistas

O conjunto de 10 voos e 1 pista é o que se encontra representado nas figuras 1, 2 e 3.

Comparação com outros resultados Na dificuldade de encontrar publicamente resultados de resolução de problemas semelhantes usando lógica com restrições foi solicitado a um grupo com o mesmo tema de trabalho que, para os nossos testes, nos fornecesse os seus valores de tempo de execução bem como, nos fornecesse testes criados pelo grupo e resultados obtidos. Estes testes foram executados utilizando a implementação aqui apresentada.

A tabela 2 mostra os tempos de execução obtidos por Mineiro et al.[3] para os testes da tabela 1 e a tabela 3 os tempos de execução obtidos para testes de Mineiro et al.[3].

Teste	Tempo
E	1,30s
F	5,95s
G	16,82s
H	3465,91s

Tabela 2: Tempo de execução obtidos para testes da tabela 1 por Mineiro et al.[3]

Número de Voos	Número de Pistas	Custo	Tempo Mineiro et al.[3]	Tempo solução apresentada
15	3	1	1,72s	4,31s
15	2	5	11,72s	229,690s

Tabela 3: Tempo de execução obtidos para testes fornecidos por Mineiro et al.[3]

A análise dos testes de Mineiro et al.[3] permitiu-nos verificar a predominância de casos em que a diferença entre o tempo mínimo de aterragem e o tempo preferencial é muito diferente à diferença entre o tempo preferencial de aterragem e o tempo máximo. Esta análise permite-nos indicar como causa principal para a descrepância de valores (melhor tempo de execução da solução apresentada para os testes indicados na tabela 1 e pior para os testes fornecidos por Mineiro et al.[3]) a utilização da opção *bisect* no predicado de pesquisa. Tal como referido na secção 7, esta opção é utilizada pois considera-se que o tempo preferencial tende a ser o tempo central do domínio, o que não se verifica nos testes fornecidos por Mineiro et al.[3].

10 Conclusões e Perspetivas de Desenvolvimento

Após realização desta implementação conclui-se que com um pequeno número de linhas de código é possível obter soluções óptimas para o problema. Foi necessária a abstração da programação declarativa para que fosse possível explorar ao máximo as possibilidades do *Prolog* nesta área.

A solução proposta tem como principal vantagem encontrar sempre a melhor solução utilizando em grande parte predicados oferecidos pelo *Prolog* e por esse motivo expectavelmente mais eficientes. Não se pode deixar de notar que a pesquisa pela solução óptima traz grandes custos de processamento. A solução apresentada mostrou-se incapaz de apresentar soluções para um grande número de voos e reduzido número de pistas em tempo útil.

A comparação com outros resultados permitiu afirmar que a nossa solução é eficiente nos casos em que o valor preferencial de tempo de aterragem se aproxima do valor central do domínio de tempo de aterragem. Apresenta, no entanto, menor desempenho quando isso não acontece.

No futuro, o aumento do domínio de cada voo e de todos os voos (distância temporal entre o primeiro e o último voo) será certamente uma abordagem interessante. A implementação beneficiaria de computação paralela das várias soluções para posterior escolha do resultado óptimo.

Referências

1. Sicstus Prolog. Documentation of version 4.
   http://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/
   Consultado em Dezembro de 2013.
2. Artificial Intelligence. Domain Splitting.
   http://artint.info/html/ArtInt_80.html
   Consultado em Dezembro de 2013.
3. Vitor Mineiro e Rodolfo Rodrigues. Aterragem de Aviões.
   FEUP - PLOG - Grupo 37 - 2013
4. Mohammad Mesgarpour, Chris Potts e Julia Bennell. Models for Aircraft Landing Optimization.
   European Organisation for the Safety of Air Navigation
   https://www.eurocontrol.int/eec/gallery/content/public/

document/other/other_document/201007_D2Y2_Models.pdf
Consultado em Dezembro de 2013

A Código Prolog

```
:- use_module(library(clpfd)).

buildTasks(Lf,Ls,Le,Lc,Lm,Lt) :- length(Lf,Nflights),
length(Ls,Nflights), length(Le,Nflights),length(Lc,Nflights),
                                        length(Lm,Nflights),
numRunways(Nmac), domain(Ls,0,sup),
                                        domain(Le,0,sup),
domain(Lm,1,Nmac), domain(Lc,0,sup),
                                        iterateflights(Lf,Ls,Le,Lm,Lc,
[],Lt).

iterateflights([],_,_,_,_,Lt,Lt).
iterateflights([flight(Id,Tmin,Tpref,Tmax,Cbefore,Cafter,D)|Tf],[Hs|Ts],
[He|Te],[Hm|Tm],[Hc|Tc],Ltemp, Lt) :-
        Tendmin is Tmin + D, Tendmax is Tmax + D,

        (Tmin>Tmax -> write('Flight '), write(Id), write(' not valid'),
break;true),
        (Tpref>Tmax -> write('Flight '), write(Id), write(' not valid'),
break;true),
        (Tpref<Tmin -> write('Flight '), write(Id), write(' not valid'),
break;true),
        (D=<0 -> write('Flight '), write(Id), write('not valid'),
break;true),
        (Cafter<0 -> write('Flight '), write(Id), write('not valid'),
break;true),
        (Cbefore<0 -> write('Flight '), write(Id), write('not valid'),
break;true),

        domain([Hs],Tmin,Tmax), domain([He],Tendmin,Tendmax),
        Hs #< Tpref #<=> B, Hc #= Cbefore*(Tpref-Hs)*B +
Cafter*(Hs-Tpref)*(1-B),
        append(Ltemp,[task(Hs,D,He,1,Hm)],Out),
        iterateflights(Tf,Ts,Te,Tm,Tc,Out,Lt).

buildmachines(N,N,Temp,Lout) :- append(Temp,[machine(N,1)],Lout),!.
buildmachines(I,N,Temp,Lout) :- append(Temp,[machine(I,1)],Out), I1 is
I+1, buildmachines(I1,N,Out,Lout).

schedule(MilisecondsTimeOut,Lf,Ls,Le,Lc,Lm,SumCost,FlagTimeOout) :-
        findall(flight(Id,Tmin,Tpref,Tmax,Cbefore,Cafter,Tafter),flight(
Id,Tmin,Tpref,Tmax,Cbefore,Cafter,Tafter),Lf),
        numRunways(Nmac),
        buildmachines(1,Nmac,[],Machines),
```

```
        buildTasks(Lf,Ls,Le,Lc,Lm,Lt),
        domain([SumCost],0,sup),
        sum(Lc,#=,SumCost),
        cumulatives(Lt,Machines,[bound(upper)]),
        append(Ls,Lm,V1),
        append(V1,[SumCost],Vars),
        (MilisecondsTimeOut > 0 -> Options =
[bisect,down,minimize(SumCost),time_out(MilisecondsTimeOut,FlagTimeOout)
];
         Options = [bisect,down,minimize(SumCost)]),
        labeling(Options, Vars).

showFlights([],[],[],[]).
showFlights([flight(Id,_,_,_,_,_,_)|Tf],[Hs|Ts],[Hc|Tc],[Hm|Tm]) :-

                    write('Flight No '), write(Id),

                    write(' -> landing time: '),

                    write(Hs),

                    write('; runway '),

                    write(Hm),

                    write('; cost '),

                    write(Hc),

                    write('.'),nl,

                    showFlights(Tf,Ts,Tc,Tm).

readSeconds(Out) :- write('Seconds to timeout (0 for no
timeout)'),nl,read(NumSecs), isNumber(NumSecs,Out).
isNumber(X,X) :- number(X), X >= 0, !.
isNumber(_,Z) :- write('Invalid'), read(Y), !, isNumber(Y, Z).


landing :- write('Landing System. Input name of
file:'),nl,read(FileName),[FileName],
   readSeconds(NumSecs), MilisecondsTimeOut is NumSecs*1000,
           write('Starting...'),nl,
           statistics(runtime, [T0|_]),
           schedule(MilisecondsTimeOut,Lf,Ls,_,Lc,Lm,SumCost,
```

```
FlagTimeOout),
           statistics(runtime, [T1|_]),
           (NumSecs > 0 -> write('Timeout: '), write(FlagTimeOout),
nl;true),
           length(Lf,N), numRunways(Nmac), write(N), write(' flight(s)
scheduled to land on '), write(Nmac), write(' runway(s).'), nl,
           nl,
           showFlights(Lf,Ls,Lc,Lm),nl, write('Total Cost: '),
write(SumCost),write('.'),nl,nl,nl,
           write('Statistics:'),nl,
           fd_statistics,nl,
           T is T1 - T0,
           format('task took ~3d sec.~n', [T]).
```