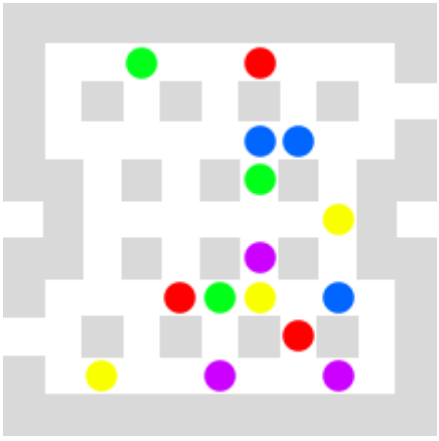
Birdwatching

Disciplina de Programação em Lógica, do 3º Ano (1º Semestre) do Mestrado Integrado em Eng. Informática e Computação da FEUP



Carlos Tiago da Rocha Babo
Felipe de Souza Schmitt
Hélder Alexandre Moreira dos Santos

FEUP-PL, Turma 3MIEIC1, Grupo 106

Resumo/Abstract.

O trabalho foi realizado no âmbito da disciplina de Programação em Lógica na Faculdade de Engenharia da Universidade do Porto, com o objectivo de aplicar os conhecimentos adquiridos ao longo do semestre, mais propriamente programação de PROLOG com restrições de forma a encontrar uma solução para o puzzle Birdwatching.

Em primeiro lugar foi desenvolvida uma solução que tinha partes onde era aplicado prolog com restrições e outras sem. Após verificar que os tempos de execução não eram satisfatórios, decidimos desenvolver uma solução baseada apenas em prolog com restrições.

Neste artigo comparamos as diferenças dos seus tempos de execução e avaliamos qual das duas soluções é mais apropriada para este tipo de problemas. Para além disso, foi ainda implementada a geração de problemas aleatórios.

1 Introdução

O objectivo principal para este trabalho passou por aplicar os nossos conhecimentos adquiridos em Programação em Lógica. Em relação ao tema, achamos que este problema era cativante e interessante, sendo estas as nossas principais motivações na resolução do trabalho.

Este artigo é constituído por três partes principais: em primeiro lugar é apresentada a descrição do problema a resolver. De seguida é demonstrado as abordagens utilizadas para a resolução desse mesmo problema, e por fim uma análise dos resultados obtidos e a comparação entre abordagens diferentes.

2 Descrição do problema

O problema Birdwatching consiste num puzzle em que é necessário encontrar um caminho entre a entrada do tabuleiro e a sua saída, que passe pelo mesmo número de pássaros de cada cor, não podendo passar na mesma posição mais do que uma vez.

Assim, o objectivo era encontrar uma solução utilizando a linguagem PROLOG com restrições.

3 Ficheiros de Dados

Escolhemos vários puzzles em que o objectivo é passar por um, dois e três pássaros de cada cor.

O primeiro problema que utilizamos para encontrar a resolução através das restrições que definimos encontra-se em anexo no ponto 1, junto com a sua solução.

O segundo problema é de mesma dificuldade que o primeiro, alterando a posição dos pássaros no tabuleiro, no ponto 2 do anexo.

O terceiro problema destaca-se dos outros pois neste caso é necessário encontrar um caminho desde a entrada à saída do tabuleiro que passe por apenas dois pássaros de cada cor e encontra-se no ponto 3 do anexo deste artigo.

O quarto problema e o mais complexo obriga o caminho a passar por três pássaros de cada cor, este caminho encontra-se exemplificado no anexo, ponto 4.

4 Variáveis de Decisão

Neste trabalho existe uma variável de decisão, designada por "Caminho" que indica o caminho percorrido para chegar à saída do tabuleiro. Esta variável de decisão tem um domínio de 1 a CaminhoMax+1, em que a variável CaminhoMax é o número máximo que o caminho pode ter e é calculada pela subtracção das casas a zero e das casas por onde o caminho não passa às 121 casas totais do tabuleiro. No pior dos casos 60 é o número máximo do caminho e 61 o número de casas por onde o caminho não passa.

Para além dessa variável, é usada a variável de domínio Birds, que contem a ordem pela qual o caminho passa nos pássaros. Com o uso de restrições, garantimos que o caminho passa por 'x' pássaros de cada cor, de acordo com o tabuleiro.

5 Restrições

As restrições aplicadas para a solução são várias. Em primeiro lugar as restrições das paredes do tabuleiro que estão assinaladas com um 0 no tabuleiro. De seguida aplicamos restrições de conectividade 4, ou seja, dada uma casa, as posições possíveis de se chegar estão em cima, em baixo, à esquerda ou à direita. Para além disso, restringe-se que se uma dada posição está no caminho, então esta tem de ter uma casa de entrada e outra de saída.

As restrições rígidas aplicadas resumem-se à casa de entrada, à casa adjacente à entrada, à casa de saída, à casa adjacente à saída e às paredes.

Enquanto as restrições flexíveis são aplicadas às posições por onde é possível passar um caminho, podendo assumir o valor da sua ordem, caso pertençam ao caminho ou o valor da parede.

6 Função de Avaliação

A solução obtida é avaliada verificando o caminho percorrido com a resolução existente no *site* da bibliografia, assim como analisando o output e verificando que passa correctamente pelos 'x' de cada cor.

7 Estratégia de Pesquisa

Na implementação da etiquetagem (“labeling”) utilizamos uma as opções min ,up e minimize(Size), de forma a que a etiquetagem comece pela variável mais à esquerda com o domínio esquerdo mais pequeno, ou seja pela entrada do tabuleiro. Enquanto o up é para o domínio ser explorado em ordem ascendente , desta forma processando primeiro os caminhos mais pequenos, uma vez que a etiquetagem destes é mais rápida.

8 Visualização da Solução

A visualização da solução em modo de texto é aplicada através dos predicados que se encontram no anexo A ponto 5.

O predicado printBirdsPos imprime a partir da lista dos pássaros que se encontram no caminho a sua posição e a sua ordem no caminho, sendo desta forma mais fácil confirmar que a solução obtida é válida.

O segundo predicado da visualização da solução é o printSol que a partir da lista do caminho, dos pássaros e do tabuleiro imprime a solução no tabuleiro. Caso a cabeça da lista seja um pássaro, nessa posição é impresso um ‘B’ a simbolizar um pássaro que se encontra no caminho. Caso faça parte do caminho é impresso um ‘*’ para ser mais fácil a visualização do caminho no tabuleiro, caso contrário é impresso o tabuleiro de origem.

9 Resultados

Em primeiro lugar foi desenvolvida uma resolução híbrida para o problema, pois inclui Prolog sem e com restrições, calculando os caminhos a percorrer através de Prolog sem restrições e munindo-se das restrições apenas para seleccionar os pássaros do caminho. Após o desenvolvimento desta solução, foi verificado que a solução para os problemas em que o caminho tem de passar por apenas um pássaro eram resolvidos entre 850ms a 900ms. Enquanto a solução dos problemas em que o caminho tem que passar obrigatoriamente por dois pássaros eram resolvidos entre 3 minutos e 40 segundos a 3 minutos e 50 segundos.

Uma vez que para a resolução deste segundo problema consideramos que o seu tempo de execução era demasiado, procuramos tentar abordar o problema de outra forma e criar uma resolução utilizando apenas Prolog com restrições.

Após realizar uma solução para o problema utilizando apenas restrições conseguimos observar através dos resultados que para a resolução de um problema em que o caminho tem que passar por um pássaro de cada cor o tempo de execução é de 1900ms ~2segundos. Ou seja, cerca do dobro da solução dada pela resolução híbrida. No entanto, no caso em que o caminho tem que passar por dois pássaros de cada cor o tempo de execução é consideravelmente menor, sendo acerca de 14 segundos, diminuindo assim o tempo de execução para 6,3% do tempo inicial.

10 Geração de problemas

Implementámos também a valorização que consistia na geração aleatória de puzzles. São gerados vários tabuleiros (com solução ou não) que podem depois ser resolvidos (caso tenham solução) pelo nosso programa. Desta forma garantimos que o programa gera aleatoriamente um puzzle, não testando se é um puzzle válido (capaz de ser resolvido pelo nosso programa), garantindo assim que geramos assim todo o tipo de puzzles e não apenas puzzles que fossem capazes de ser resolvidos pelo nosso programa. Para esta geração criamos o tabuleiro inicial que depois é preenchido de forma aleatória com o número de pássaros adequado para o tipo de solução que se desejar (1, 2, 3 pássaros, etc…).

11 Conclusões e Perspectivas de Desenvolvimento

Uma vez que este trabalho nos deu oportunidade de comparar duas soluções para o mesmo problema utilizando duas estratégias diferentes é possível tirar conclusões credíveis a diferenciar as duas soluções desenvolvidas.

Ao analisar os resultados podemos concluir que a solução que utiliza apenas Prolog com restrições é mais eficaz globalmente do que a solução que apenas usa parcialmente as restrições. Isto porque a solução híbrida encontra uma solução, testa se essa solução é válida e caso falhe procura uma nova solução e repete-se num ciclo até encontrar uma solução que passe as restrições. Em alternativa, a solução que utiliza apenas restrições aplica todas as restrições e só depois procura uma solução válida, tornando a procura mais rápida.

Este projecto acabou por se tornar um grande desafio e uma boa forma de aprendizagem, uma vez que abordamos o mesmo problema de duas formas diferentes e fomos capazes de verificar a diferença entre as duas abordagens assim como comparar os seus tempos de execução e eficácia. Assim como também foi possível através deste projecto encontrar uma solução, verificar as suas limitações e posteriormente melhorar essa solução para que o resultado seja mais eficaz. Este projecto também serviu para aplicarmos os nossos conhecimentos em Prolog com restrições com o objectivo de resolver um problema de decisão.

Em relação a possíveis melhoramentos, as restrições relativas às posições do caminho podiam ser melhoradas. Para tal, seria necessário encontrar uma forma de calcular o caminho máximo e o caminho mínimo entre a entrada e uma dada posição, para restringir ainda mais o seu domínio.

Bibliografia

http://www2.stetson.edu/~efriedma/birds/

http://www.fi.muni.cz/~hanka/sicstus/doc-3.11/pdf/sicstus.pdf

Anexos

Anexo A

1. Problema 1

```
[0,0,0,0,0,0,0,0,0,0,0,
 0,1,1,2,1,1,3,1,1,1,0,
 0,1,0,1,0,1,0,1,0,1,1,
 0,1,1,1,1,1,4,4,1,1,0,
 0,0,1,0,1,0,2,0,1,0,0,
 0,0,1,1,1,1,1,1,5,0,0,
 0,0,1,0,1,0,6,0,1,0,0,
 0,1,1,1,3,2,5,1,4,1,0,
 1,1,0,1,0,1,0,3,0,1,0,
 0,1,5,1,1,6,1,1,6,1,0,
 0,0,0,0,0,0,0,0,0,0,0].
```

Onde a solução seria percorrer o seguinte caminho:

```
Solucao do problema:
0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 * * * 0
0 1 0 1 0 1 0 * 0 * *
0 1 1 1 1 1 1 B * 1 0
0 0 1 0 1 0 1 0 * 0 0
0 0 1 1 * * * * B 0 0
0 0 1 0 * 0 1 0 1 0 0
0 * * * B B 1 1 1 1 0
* * 0 * 0 * 0 1 0 1 0
0 1 1 * * B 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0
Passaro - Ordem do caminho:
83-11 82-12 41-21 64-18 105-9
```

2. Problema 2

```
[0,0,0,0,0,0,0,0,0,0,0,0,
 0,1,1,5,1,1,3,4,1,1,0,
 0,1,0,1,0,1,0,1,0,1,1,
 0,1,1,3,1,1,2,2,1,1,0,
 0,0,1,0,5,0,1,0,1,0,0,
 0,0,1,1,4,1,2,6,1,0,0,
 0,0,6,0,1,0,1,0,1,0,0,
 0,4,1,1,1,1,6,1,1,1,0,
 1,1,0,1,0,1,0,1,0,1,0,
 0,1,1,1,5,1,3,1,1,1,0,
 0,0,0,0,0,0,0,0,0,0,0].
```

E a sua solução:

```
Solucao do problema:
0 0 0 0 0 0 0 0 0 0 0
0 1 1 B * * 1 1 1 1 0
0 1 0 * 0 * 0 1 0 * *
0 1 * B 1 * B 1 * * 0
0 0 * 0 1 0 * 0 * 0 0
0 0 * 1 B * * 1 * 0 0
0 0 B 0 * 0 1 0 * 0 0
0 * * 1 * * * * * 1 0
* * 0 1 0 1 0 1 0 1 0
0 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0
Passaro - Ordem do caminho:
40-16 37-9 60-20 15-11 69-5
```

3. Problema 3

```
[0,0,0,0,0,0,0,0,0,0,0,0,
 0,6,6,1,4,3,1,2,1,1,0,
 0,1,0,4,0,4,0,1,0,1,1,
 0,1,1,1,2,1,2,5,5,5,0,
 0,0,1,0,1,0,1,0,1,0,0,
 0,0,1,1,1,3,4,1,1,0,0,
 0,0,1,0,6,0,1,0,1,0,0,
 0,1,1,1,1,6,1,1,1,1,0,
 1,1,0,1,0,3,0,1,0,1,0,
 0,1,1,1,1,1,3,5,1,2,0,
 0,0,0,0,0,0,0,0,0,0,0].
```

Sendo a sua solução o seguinte caminho:

```
Solucao do problema:
0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 B * B * * 0
0 1 0 1 0 B 0 1 0 * *
0 1 1 1 1 * B B B 1 0
0 0 1 0 1 0 1 0 * 0 0
0 0 * * * 1 B * * 0 0
0 0 * 0 B 0 * 0 1 0 0
0 * * * * B * 1 1 1 0
* * 0 * 0 B 0 1 0 1 0
0 1 1 * * * 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0
Passaro - Ordem do caminho:
19-31 40-26 17-29 94-16 28-28 62-20 41-25 42-24 71-9
83-17
```

4. Problema 4

```
[0,0,0,0,0,0,0,0,0,0,0,0,
 0,6,6,6,4,5,1,1,1,1,0,
 0,1,0,5,0,6,0,2,0,1,1,
 0,1,3,1,2,1,5,1,2,1,0,
 0,0,4,0,1,0,2,0,4,0,0,
 0,0,1,1,4,1,6,1,5,0,0,
 0,0,1,0,1,0,1,0,1,0,0,
 0,1,3,1,3,1,3,1,1,1,0,
 1,1,0,1,0,1,0,1,0,1,0,
 0,1,4,1,1,1,5,3,1,2,0,
 0,0,0,0,0,0,0,0,0,0,0].

Solucao do problema:
0 0 0 0 0 0 0 0 0 0 0
0 B B B B B * * * * 0
0 * 0 1 0 1 0 1 0 * *
0 * B * B * B * B 1 0
0 0 1 0 1 0 1 0 B 0 0
0 0 1 1 1 1 1 1 B 0 0
0 0 1 0 1 0 1 0 * 0 0
0 1 1 1 1 * B * * * 0
* * 0 1 0 * 0 * 0 * 0
0 * B * * * 1 B * B 0
0 0 0 0 0 0 0 0 0 0 0
Passaro - Ordem do caminho:
38-26 42-22 109-15 36-28 84-10 107-13 16-34 53-21 102-4
17-35 40-24 64-20 13-31 14-32 15-33
```

5. Predicados correspondentes ao output da solução.

```
printSol(Caminho,_, Pos,_,_,_):-length(Caminho, Pos).
printSol(Caminho,SizeC, Pos, MaxSizeCaminho,Birds,
Tab):-
  sublist(Caminho, Part, Pos, SizeC, Next),
  sublist(Tab, Part2, Pos, SizeC, Next),
  printLinhaSol(Part, MaxSizeCaminho, Birds, Part2),nl,
  length(Caminho, S),
  Next2 is S-Next,
  printSol(Caminho, SizeC, Next2, MaxSizeCaminho,
Birds, Tab).

printLinhaSol([],_,_,_).
printLinhaSol([H|T], H, Birds, [H2|T2]):-
  write(H2),write(' '),
  printLinhaSol(T, H, Birds, T2), !.
printLinhaSol([H|T], S, Birds, [_|T2]):-
  member(H, Birds),
  write('B '),
  printLinhaSol(T, S, Birds, T2), !.
printLinhaSol([_|T], S, Birds,[_|T2]):-
  write('* '),
  printLinhaSol(T, S, Birds, T2), !.
```

Anexo B

Código Fonte:

```
% RESOLUCAO DE PUZZLES BIRDWATCHING COM RESTRICOES E
COM UMA SOLUCAO HIBRIDA
% AUTORES: TIAGO BABO, HELDER MOREIRA E FELIPE SCHMITT

% IMPORTACAO DE MODULOS
:-use_module(library(random)).
:-use_module(library(clpfd)).
:-use_module(library(lists)).

% ---UTILITARIOS---
%   Verde = 2 = A
%   Vermelho = 3 = B
%   Azul = 4 = C
%   Amarelo = 5 = D
%   Roxo = 6 = E

% REMOVE UMA POSICAO DE UMA LISTA
remove_at(X,[X|Xs],1,Xs).
remove_at(X,[Y|Xs],K,[Y|Ys]) :- K > 1,
   K1 is K - 1, remove_at(X,Xs,K1,Ys).

% INSERE ELEMENTO NUMA CERTA POSICAO DE UMA LISTA
insert_at(X,L,K,R) :- remove_at(X,R,K,L).

% OBTER POSICAO DA LISTA CORRIDA, DADA UMA COLUNA E UMA
LINHA
getPos(X,Y, Pos) :-
  Pos is Y*11+X.

% DADO UM TABULEIRO, RETIRA OS PASSAROS DA MESMA COR E
AGRUPA-OS EM LISTAS SEPARADAS
parser([],[],[],[],[],[],_).
parser([2|R],[Indice|RA],BOut,COut,DOut,EOut, Indice)
:-
  Indice1 is Indice +1,
  parser(R,RA,BOut,COut,DOut,EOut, Indice1) ,!.
parser([3|R],AOut,[Indice|RB],COut,DOut,EOut, Indice)
:-
  Indice1 is Indice +1,
  parser(R,AOut,RB,COut,DOut,EOut, Indice1) ,!.
```

```
parser([4|R],AOut,BOut,[Indice|RC],DOut,EOut, Indice)
:-
  Indice1 is Indice +1,
  parser(R,AOut,BOut,RC,DOut,EOut, Indice1) ,!.
parser([5|R],AOut,BOut,COut,[Indice|RD],EOut, Indice)
:-
  Indice1 is Indice +1,
  parser(R,AOut,BOut,COut,RD,EOut, Indice1) ,!.
parser([6|R],AOut,BOut,COut,DOut,[Indice|RE], Indice)
:-
  Indice1 is Indice +1,
  parser(R,AOut,BOut,COut,DOut,RE, Indice1) ,!.
parser([_|R],AOut,BOut,COut,DOut,EOut, Indice) :-
  Indice1 is Indice +1,
  parser(R,AOut,BOut,COut,DOut,EOut, Indice1) ,!.

% --- \UTILITARIOS ---

% TABULEIROS DE JOGO
tabuleiro(T) :-
  T = [0,0,0,0,0,0,0,0,0,0,0,
     0,1,1,2,1,1,3,1,1,1,0,
     0,1,0,1,0,1,0,1,0,1,1,
     0,1,1,1,1,1,4,4,1,1,0,
     0,0,1,0,1,0,2,0,1,0,0,
     0,0,1,1,1,1,1,1,5,0,0,
     0,0,1,0,1,0,6,0,1,0,0,
     0,1,1,1,3,2,5,1,4,1,0,
     1,1,0,1,0,1,0,3,0,1,0,
     0,1,5,1,1,6,1,1,6,1,0,
     0,0,0,0,0,0,0,0,0,0,0].

tabuleiro_2(T) :-
  T = [0,0,0,0,0,0,0,0,0,0,0,
     0,1,1,5,1,1,3,4,1,1,0,
     0,1,0,1,0,1,0,1,0,1,1,
     0,1,1,3,1,1,2,2,1,1,0,
     0,0,1,0,5,0,1,0,1,0,0,
     0,0,1,1,4,1,0,6,1,0,0,
     0,0,6,0,1,0,1,0,1,0,0,
     0,1,1,1,1,1,1,1,1,1,0,
     1,1,0,1,0,1,0,1,0,1,0,
     0,1,1,2,3,4,5,6,1,1,0,
     0,0,0,0,0,0,0,0,0,0,0].
```

```
tabuleiroTwoBirds(T) :-
  T = [0,0,0,0,0,0,0,0,0,0,0,0,
     0,6,6,1,4,3,1,2,1,1,0,
     0,1,0,4,0,4,0,1,0,1,1,
     0,1,1,1,2,1,2,5,5,5,0,
     0,0,1,0,1,0,1,0,1,0,0,
     0,0,1,1,1,3,4,1,1,0,0,
     0,0,1,0,6,0,1,0,1,0,0,
     0,1,1,1,1,6,1,1,1,1,0,
     1,1,0,1,0,3,0,1,0,1,0,
     0,1,1,1,1,1,3,5,1,2,0,
     0,0,0,0,0,0,0,0,0,0,0].

tabuleiroThreeBirds(T) :-
  T = [0,0,0,0,0,0,0,0,0,0,0,0,
 0,6,6,6,4,5,1,1,1,1,0,
 0,1,0,5,0,6,0,2,0,1,1,
 0,1,3,1,2,1,5,1,2,1,0,
 0,0,4,0,1,0,2,0,4,0,0,
 0,0,1,1,4,1,6,1,5,0,0,
 0,0,1,0,1,0,1,0,1,0,0,
 0,1,3,1,3,1,3,1,1,1,0,
 1,1,0,1,0,1,0,1,0,1,0,
 0,1,4,1,1,1,5,3,1,2,0,
 0,0,0,0,0,0,0,0,0,0,0].


% MASCARA DO TABULEIRO
tabuleiroMask(T) :-
  T = [0,0,0,0,0,0,0,0,0,0,0,0,
     0,1,1,1,1,1,1,1,1,1,0,
     0,1,0,1,0,1,0,1,0,1,1,
     0,1,1,1,1,1,1,1,1,1,0,
     0,0,1,0,1,0,1,0,1,0,0,
     0,0,1,1,1,1,1,1,1,0,0,
     0,0,1,0,1,0,1,0,1,0,0,
     0,1,1,1,1,1,1,1,1,1,0,
     1,1,0,1,0,1,0,1,0,1,0,
     0,1,1,1,1,1,1,1,1,1,0,
     0,0,0,0,0,0,0,0,0,0,0].

% SOLUCAO COM TOTALMENTE COM RESTRICOES

% COLOCA AS RESTRICOES NAS VARIAS PECAS DO TABULEIRO
processaCaminho2(_,112,_,_,_,_,_,_):- !.
```

```
% SE A POSICAO CORRESPONDE A UMA PAREDE NO TABULEIRO,
LOGO A ORDEM E' ZERO, E NAO E' NECESSARIO RESTRINGIR
(FEITO ANTERIORMENTE)
processaCaminho2(Caminho, PosAct, Posicoes, PecasP,
Tab, Fim, Inicio, SizeCaminho):-
  element(PosAct, Tab, X),
  X = 0,
  PosAct2 is PosAct+1,
  processaCaminho2(Caminho, PosAct2, Posicoes, PecasP,
Tab, Fim, Inicio,SizeCaminho), !.

% PARA A POSICAO DE ENTRADA E SAIDA NAO E' NECESSARIO
COLOCAR RESTRICOES
processaCaminho2(Caminho, PosAct, Posicoes, PecasP,
Tab, Fim, Inicio, SizeCaminho):-
  Inicio2 is Inicio+1,
  Fim2 is Fim+1,
  Restricoes = [Inicio,Fim, Inicio2, Fim2],
  member(PosAct, Restricoes),
  PosAct2 is PosAct+1,
  processaCaminho2(Caminho, PosAct2, Posicoes, PecasP,
Tab, Fim, Inicio, SizeCaminho), !.

% PARA PECAS QUE NAO SAO PAREDE, ENTRADA OU SAIDA
processaCaminho2(Caminho, PosAct, Posicoes, PecasP,
Tab, Fim, Inicio, SizeCaminho):-
  Redondeza = [A1,B1,C1,D1],
  A is PosAct-1,
  B is PosAct+1,
  C is PosAct-11,
  D is PosAct+11,
  element(A, Caminho, A1),
  element(B, Caminho, B1),
  element(C, Caminho, C1),
  element(D, Caminho, D1),
  element(PosAct, Caminho, X),
  element(_, Redondeza, ValEntrada),
  element(_, Redondeza, ValSaida),
  (X #> 0 #=> ValEntrada #= X-1 #/\ ValSaida #= X+1)
#\/ X #= SizeCaminho,
  isBird(PosAct, Caminho, Posicoes, PecasP),
  PosAct2 is PosAct+1,
  processaCaminho2(Caminho, PosAct2, Posicoes, PecasP,
Tab, Fim, Inicio, SizeCaminho).
```

```
isBird(PosAct, Caminho, Posicoes, PecasP):-
  element(PosBird, Posicoes, PosAct),
  element(PosAct, Caminho, Val),
  element(PosBird, PecasP, Val2),
  Val #= Val2.
isBird(_,_,_,_).

% GARANTE QUE O CAMINHO PASSA POR UM PASSARO DE CADA
garanteUm([]).
garanteUm([A,B,C|P]):-
  Ordens = [A,B,C],
  count(61, Ordens, #=, X),
  X #= 2,
  garanteUm(P).

% GARANTE QUE O CAMINHO PASSA POR DOIS PASSAROS DE CADA

garanteDois([], _).
garanteDois([A,B,C,D|P], SizeCaminho):-
  Ordens = [A,B,C,D],
  count(SizeCaminho, Ordens, #=, X),
  X #= 2,
  garanteDois(P, SizeCaminho).


garanteXBirds(Birds,_,_,_,Pos):-length(Birds, Pos), !.
garanteXBirds(Birds, SizeCaminho, XBirds, NBirds,
Pos):-
  sublist(Birds, Part, Pos, NBirds, Next),
  W is NBirds-XBirds,
  count(SizeCaminho, Part, #=, X),
  X #= W,
  length(Birds, S),
  Next2 is S-Next,
  garanteXBirds(Birds, SizeCaminho, XBirds, NBirds,
Next2).

% RESTRINGE AS CASAS PAREDE AO VALOR ZERO
casasZero([],[], _).
casasZero([H2|T2], [H|T], SizeCaminho):-
  H #=0 #=> H2 #=SizeCaminho,
  casasZero(T2, T, SizeCaminho).
```

```
% ROTINA RESPONSAVEL POR ENCONTRAR O CAMINHO E FAZER
LABELING

xBirdsComRestricoes(Caminho, Birds, Tabuleiro, SizeC,
XBirds, NBirds):-
  write('Tabuleiro a processar: '),
  nl,
  printTabuleiro(Tabuleiro, 1, 11),
  tabuleiroMask(T),
  parser(Tabuleiro,AOut,BOut,COut,DOut,EOut,1),
  append(AOut, BOut, P),
  append(P, COut, P2),
  append(P2, DOut, P3),
  append(P3, EOut, P4),
  length(Tabuleiro, SizeTab),
  length(Caminho, SizeTab),
  length(P4, SizeBirds),
  length(Birds, SizeBirds),
  SizeCaminho is (SizeC)*6,
  domain(Caminho, 1, SizeCaminho),
  domain(Birds, 1, SizeCaminho),
  Entrada is SizeC*SizeC-3*SizeC+1,
  Entrada2 is Entrada+1,
  element(Entrada, Caminho, 1),
  element(Entrada2, Caminho, 2),
  count(SizeCaminho, Caminho, #=, Sz),
  Sz2 #= SizeTab-Sz #/\ Sz3 #= Sz2-1,
  Saida is SizeC*3,
  Saida2 is Saida-1,
  element(Saida2, Caminho, Sz3),
  element(Saida, Caminho, Sz2),
  casasZero(Caminho, T, SizeCaminho),
  garanteXBirds(Birds, SizeCaminho, XBirds, NBirds, 0),
  processaCaminho2(Caminho, 13, P4, Birds, T, Entrada,
Saida, SizeCaminho),
  append(Birds, Caminho, List),
  write('Labeling...'),nl,
  !,
  labeling([min,up, maximize(Sz)], List),
  write('Solucao do problema:'),nl,
  printSol(Caminho, SizeC, 0, SizeCaminho, Birds, T),
  write('Passaro - Ordem do caminho: '),nl,
  printBirdsPos(Birds, P4, SizeCaminho),nl,
  fd_statistics,
  statistics(runtime, [_,Time]),
```

```
  write('Tempo de execucao: '),
  write(Time),nl.

% IMPRIME AS POSICOES E ORDENS DOS PASSAROS NO CAMINHO
printBirdsPos([],_,_).
printBirdsPos([H|T], [H2|T2], Max):-
  H \= Max,
  write(H2-H), write(' '),
  printBirdsPos(T, T2, Max), !.
printBirdsPos([_|T], [_|T2], Max):-
  printBirdsPos(T, T2, Max).

% IMPRIME A SOLUCAO NUM TABULEIRO
printSol(Caminho, _, Pos, _, _,_):-length(Caminho,
Pos).
printSol(Caminho, SizeC, Pos, MaxSizeCaminho, Birds,
Tab):-
  sublist(Caminho, Part, Pos, SizeC, Next),
  sublist(Tab, Part2, Pos, SizeC, Next),
  printLinhaSol(Part, MaxSizeCaminho, Birds, Part2),nl,
  length(Caminho, S),
  Next2 is S-Next,
  printSol(Caminho, SizeC, Next2, MaxSizeCaminho,
Birds, Tab).

printLinhaSol([], _, _, _).
printLinhaSol([H|T], H, Birds, [H2|T2]):-
  write(H2),write(' '),
  printLinhaSol(T, H, Birds, T2), !.
printLinhaSol([H|T], S, Birds, [_|T2]):-
  member(H, Birds),
  write('B '),
  printLinhaSol(T, S, Birds, T2), !.
printLinhaSol([_|T], S, Birds,[_|T2]):-
  write('* '),
  printLinhaSol(T, S, Birds, T2), !.

printTabuleiro([],_,_):- nl.

printTabuleiro([H|T], Num, Size):-
  Size2 is Size+1,
  Num == Size2,
  nl,
  printTabuleiro([H|T], 1, Size), !.
```

```
printTabuleiro([H|T], Num, Size):-
  Num2 is Num+1,
  write(H),
  write(' '),
  printTabuleiro(T,Num2, Size).


% TESTES VARIOS
testeXBirds:-
  write('Resolucao de puzzles de Birdwatching'),nl,
  tabuleiro(T1),
  xBirdsComRestricoes(_, _, T1, 11, 1, 3),
  tabuleiroTwoBirds(T2),
  xBirdsComRestricoes(_, _, T2, 11, 2, 4),
  tabuleiroThreeBirds(T3),
  xBirdsComRestricoes(_, _, T3, 11, 3, 5).

% SOLUCOES ALEATORIAS
randomBirdsWay(T, 1, T, _):-!.
randomBirdsWay(T, B1, TFim, N):-
  random(12, 110, X),
  X \= 89, X \= 33,
  X \= 90, X \= 32,
  element(X, T, Y),
  Y = 1,
  remove_at(_,T,X,TNovo),
  insert_at(N,TNovo,X, TNovo2),
  B2 is B1-1,
  randomBirdsWay(TNovo2, B2, TFim, N), !.

randomBirdsWay(T, B1, TFim, N):-
  randomBirdsWay(T, B1, TFim, N).

randomPuzzle(NBirds):-
  tabuleiroMask(T),
  Birds is NBirds+3,
  randomBirdsWay(T, Birds, T2, 2),
  randomBirdsWay(T2, Birds, T3, 3),
  randomBirdsWay(T3, Birds, T4, 4),
  randomBirdsWay(T4, Birds, T5, 5),
  randomBirdsWay(T5, Birds, T6, 6),
  Y is NBirds+2,
  xBirdsComRestricoes(_, _, T6, 11, NBirds, Y).
```

```
% SOLUCAO HIBRIDA

oneBird(PosicoesEscolhidas) :-
  tabuleiro(T),
  Caminho = [A,B,C,D,E],
  Posicoes = [P1,P2,P3,P4,P5],
  PosicoesEscolhidas = [X1,X2,X3,X4,X5],
  all_distinct(PosicoesEscolhidas),
  parser(T,AOut,BOut,COut,DOut,EOut,1),
  domain(Caminho,1,5),
  element(A,AOut,P1),
  element(B,BOut,P2),
  element(C,COut,P3),
  element(D,DOut,P4),
  element(E,EOut,P5),
  member(X1, Posicoes),
  existeCaminho(89, X1, T, T1),
  member(X2, Posicoes),
  existeCaminho(X1, X2, T1, T2),
  member(X3, Posicoes),
  existeCaminho(X2, X3, T2, T3),
  member(X4, Posicoes),
  existeCaminho(X3, X4, T3, T4),
  member(X5, Posicoes),
  existeCaminho(X4, X5, T4, T5),
  existeCaminho(X5,33,T5,_).

twoBirds(PosicoesEscolhidas) :-
  tabuleiroTwoBirds(T),
  Posicoes = [P1,P2,P3,P4,P5,P6,P7,P8,P9,P10],
  PosicoesEscolhidas =
[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10],
  all_distinct(PosicoesEscolhidas),
  all_distinct(Posicoes),
  parser(T,A,B,C,D,E,1),
  element(_,A,P1),
  element(_,A,P2),
  element(_,B,P3),
  element(_,B,P4),
  element(_,C,P5),
  element(_,C,P6),
  element(_,D,P7),
  element(_,D,P8),
  element(_,E,P9),
  element(_,E,P10),
```

```
  member(X1, Posicoes),
  existeCaminho(89, X1, T, T1),
  member(X2, Posicoes),
  existeCaminho(X1, X2, T1, T2),
  member(X3, Posicoes),
  existeCaminho(X2, X3, T2, T3),
  member(X4, Posicoes),
  existeCaminho(X3, X4, T3, T4),
  member(X5, Posicoes),
  existeCaminho(X4, X5, T4, T5),
  member(X6, Posicoes),
  existeCaminho(X5, X6, T5, T6),
  member(X7, Posicoes),
  existeCaminho(X6, X7, T6, T7),
  member(X8, Posicoes),
  existeCaminho(X7, X8, T7, T8),
  member(X9, Posicoes),
  existeCaminho(X8, X9, T8, T9),
  member(X10, Posicoes),
  existeCaminho(X9, X10, T9, T10),
  existeCaminho(X10,33,T10,_).

adjacente(Inicial,Final, Tabuleiro):- Inicial >= 0,
Inicial < 121, Final is Inicial-1, nth1(Final,
Tabuleiro,1).
adjacente(Inicial,Final, Tabuleiro):- Inicial >= 0,
Inicial < 121, Final is Inicial+1, nth1(Final,
Tabuleiro,1).
adjacente(Inicial,Final, Tabuleiro):- Inicial =< 110,
Final is Inicial+11, nth1(Final, Tabuleiro,1) .
adjacente(Inicial,Final, Tabuleiro):- Inicial >= 11,
Final is Inicial-11,  nth1(Final, Tabuleiro,1).

adjacente2(Inicial,Final, Tabuleiro):- Inicial >= 0,
Inicial < 121, Final is Inicial-1, \+ nth1(Final,
Tabuleiro,0).
adjacente2(Inicial,Final, Tabuleiro):- Inicial >= 0,
Inicial < 121, Final is Inicial+1, \+ nth1(Final,
Tabuleiro,0).
adjacente2(Inicial,Final, Tabuleiro):- Inicial =< 110,
Final is Inicial+11, \+ nth1(Final, Tabuleiro,0) .
adjacente2(Inicial,Final, Tabuleiro):- Inicial >= 11,
Final is Inicial-11,  \+ nth1(Final, Tabuleiro,0).
```

```
existeCaminho(_,Final,T, _) :-
  nth1(Final,T,0),!, fail.

existeCaminho(Inicial,Final, T, TNovo2) :-
  adjacente2(Inicial,Final,T),
  remove_at(_,T,Inicial,TNovo),
  insert_at(0,TNovo,Inicial, TNovo2).

existeCaminho(Inicial,Final,T, TRet) :-
  adjacente(Inicial,P,T),
  Pos is Inicial,
  remove_at(_,T,Pos,TNovo),
  insert_at(0,TNovo,Pos, TNovo2),
  existeCaminho(P,Final,TNovo2, TRet).
```