Otimização usando Programação em Lógica com Restrições

-

Gestão Portuária de Contentores

Filipe Ribeiro – ei11141 e Ricardo Neves – up201405868

Faculdade de Engenharia da Universidade do Porto, Gestão Portuária de Contentores

Resumo: Este artigo pretende enunciar as funcionalidades pretendidas deste projeto, Gestão Portuária de Contentores. Consiste em organizar um Porto de tamanho variável para conseguir receber contentores e despachá-los da maneira mais eficiente.

KeyWords: Porto, restrições, data, eficiência, capacidade.

Introdução

A base do trabalho é fazer com que um porto virtual consiga manipular da melhor forma os contentores que recebe e despacha, através da Programação em Lógica com Restrições. Como dito no enunciado do projeto, é criada uma grelha de tamanho opcional para representar um porto inicial, vazio. Chegando navios com contentores no interior de tamanho, data e peso aleatório, cabe ao programa fazer a sua melhor disposição possível. Esta disposição, obviamente vai-se restringir pelos parâmetros referidos, que posteriormente serão detalhados neste artigo.

Descrição do Problema

Cada contentor demora um tempo a ser manipulado, a ser carregado para o camião. Quando o porto começa a encher, os contentores de maiores dimensões são obrigados a ficar na posição mais baixa de uma pilha de contentores e a otimização ocorre no tempo que resta de os contentores serem despachados para que seja possível despachar todos dentro do prazo estipulado, de forma mais eficiente e seguindo as regras obrigatórias. Estas regras regem o peso máximo que pode estar em cima de um contentor (5 vezes o peso desse mesmo), o número de contentores que pode estar em cima de um outro contentor (um terço da sua dimensão) e a data para ser despachado.

Abordagem

Uma vez que o objetivo é demorar o menor tempo possível a despachar os contentores, foi decidida a atribuição de uma data de despacho em dias, algo que o grupo considerou razoável para aplicação no mundo real, e o tempo de manipulação em horas. Através destes parâmetros, são feitas verificações para que, caso um contentor maior tenha de ser despachado primeiro, ou seja, estará no último lugar da pilha de contentores, em cima deste, estarão os contentores mais rápidos de ser manipulados. Quanto à possibilidade de haver 2 ou mais contentores com a mesma data para serem despachados, dispõe-se numa média de tempos os contentores que estão em cima dos que necessitam de ser despachados e manipula-se primeiro o que será mais rápido de o fazer.

Quanto às dimensões e peso, são valores aleatórios que definem a posição na pilha de contentores (dimensão) e o peso máximo que o contentor abaixo pode suportar.

1. Restrições

As restrições que foram utilizadas já foram enumeradas, mas detalhando o seu propósito.

A função “readPort” vai percorrer o porto e analisar se existem contentores com a data a “0”, significando que têm de ser despachados nesse dia. Nenhum contentor com data superior a outro poderá ser despachado primeiro e, caso tenham data igual, o contentor que terá menor tempo de manipulação é que será despachado primeiro.

O número de contentores numa pilha é restringido pela dimensão do contentor que se encontra no final da mesma. Este número é restringido também pelo peso máximo que o contentor em último consegue aguentar, logo, a soma dos pesos dos contentores acima deste, terá de ser menor que o peso máximo que ele suporta.

Conclusão

A interpretação do enunciado foi um pouco desafiante para o grupo, no entanto, foi tentada a melhor maneira de executar o pedido e elaborar a melhor “reposta”. Surgiram muitas dúvidas durante a confeção do código e talvez muita complicação por parte do grupo para um objetivo que se calhar estava mais ao alcance.

O programa ainda demonstra algumas falhas, por causa da complexidade implementada mas o projeto irá sofrer alterações para ser melhorado até à data de apresentação.

Este projeto permitiu uma aprendizagem mais profunda da linguagem de prolog e uma necessidade constante de repensar na resolução de um problema.

Anexos

Ficheiro porto.pl

```
:- use_module(library(random)).
:- use_module(library(lists)).

startPort :-
        write('Type the size of the Port: '),
        read(Size),
        initializePort(Size, Port),
        optionMenu(Size, Port).

test:-
        Test = [[],[],[],[1]],
        nth1(Number, Test, [1]),
        write(Number).

optionMenu(Size, Port):-
        write('YOUR PORT'), nl,
        write(Port), nl, nl,
        write('OPTIONS'), nl,
        write('1. Get new ship.'), nl,
        write('2. Exit.'), nl,
        read(Option),
        (
        Option = 1 -> arriveShip(Size, Port, NPort), optionMenu(Size, NPort);
        Option > 2 -> optionMenu(Size, Port)
        ).
```

```
arriveShip(Size, Port, NPort):-
        write('Ship is parking, handling containers'), nl,
        generateShip(Ship),
        write('Ship contains : '), write(Ship), nl,
        placeContainers(Size, Port, Ship, NPort).


placeContainers(_, Port, [], Port).
placeContainers(Size, Port, [Container|T], NPort):-
        getEmptySpots(Size, Port, EmptySpots),
        length(EmptySpots, NumberSpots),
        (
        NumberSpots > 0 -> nth1(1, EmptySpots, Empty),
                                        placeOnEmpty(Port, Container,
Empty, 1, XPort);
        getPossibleSpot(Size, Port, Container, PossPile, PossSpot),
        (
        PossPile > 0 -> (
                                PossSpot > 0 -> placeOnSpot(Port,
Container, PossPile, PossSpot, XPort);
                                write('No place available to place
container.'), nl
                                );
        write('No place available to place container.'), nl
        )
        ),
        placeContainers(Size, XPort, T, NPort).


getPossibleSpot(Size,_,_, Size, Size):-
        Size = 0.
getPossibleSpot(Size, [Pile|T], Container, PossPile, PossSpot):-
```

```
        Size > 0,
        checkPileDim(Pile, Container, 1, Possible),
        checkBottomWei(Pile, Container, 1, Possible, Check1),
        checkTopWei(Pile, Container, 1, Check1, Check2),
        (
        Check2 > 0 -> PossSpot is Check2,
                                        PossPile is Size;
        NSize is Size - 1,
        getPossibleSpot(NSize, T, Container, PossPile, PossSpot)
        ).

checkPile([C1|[]], Container, Counter, Possible):-
        Possible is Counter + 1.
checkPile([C1|[C2|T]], Container, Counter, Possible):-
        NCounter is Counter + 1,
        getContainerDimension(Container, Dim),
        getContainerDimension(C1, D1),
        getContainerDimension(C2, D2),
        (
        D1 > Dim -> (
                                D2 > Dim -> checkPile([C2|T], Container,
NCounter, Possible);
                                Possible is NCounter
                                );
        Possible is Counter
        ).

checkBottomWei(_,_,Counter, Spot, Spot):-
        Counter = Spot.
checkBottomWei([C1|T], Container, Counter, Spot, Poss):-
```

```
        Spot > Counter,
        NCounter is Counter + 1,
        getContainerWeight(C1, W1),
        getContainerWeight(Container, W2),
        getWeightOnTop(T, 0, TotalWeight),
        NewTotalWeight is TotalWeight + W2,
        MaximumWeight is W1 * 5,
        (
        MaximumWeight < NewTotalWeight -> Poss is 0;
        checkBottomWei(T, Container, NCounter, Spot, Poss)
        ).

checkTopWei(_,_,Check1,Check1):-
        Check1 = 0.
checkTopWei([_|T], Container, Counter, Check1, Check2):-
        Check1 > 0,
        Counter < Check1,
        NCounter is Counter + 1,
        checkTopWei(T, Container, NCounter, Check1, Check2).
checkTopWei([_|T], Container, Counter, Check1, Check2):-
        Check1 = Counter,
        getContainerWeight(Container, Weight),
        getWeightOnTop(T, 0, TotalWeight),
        MaxWeight is Weight * 5,
        (
        MaxWeight < TotalWeight -> Check2 is 0;
        Check2 is Check1
        ).

getEmptySpots(0,_,[]).
```

```
getEmptySpots(Size, List, [H|T]):-
        Size > 0,
        nth1(Size, List, Spot),
        NSize is Size - 1,
        (
        Spot = [] -> H is Size, getEmptySpots(NSize, List, T);
                        getEmptySpots(NSize, List, [H|T])
        ).

initializePort(0, []).
initializePort(N, [[]|T]):-
        N > 0,
        N1 is N - 1,
        initializePort(N1, T).

generateShip(Ship):-
        random(1, 5, Size),
        write('Size of ship: '), write(Size), nl,
        generateNContainers(Size, Ship).

printPort([]).
printPort([Pile|T]):-
        printPile(Pile),
        printPort(T).

printPile([]).
printPile([[H|T]|T2]):-
        write([H|T]), nl,
        printPile(T2).
```

```
printShip([]).
printShip([Container|T]):-
        write(Container), nl,
        printShip(T).



generatePile(Pile):-
        createContainer(Container, 30, 30),
        getContainerWeight(Container, Weight),
        LimWei is Weight * 5,
        getContainerDimension(Container, LimDim),
        N is div(LimDim,3),
        generateNContainers(Container, N, LimWei, LimDim, Pile).

generateNContainers(0, []).
generateNContainers(N, [Container|T]):-
        N1 is N - 1,
        generateContainer(Container, 30, 20),
        generateNContainers(N1, T).

generateContainer(Container, LimWei, LimDim):-
        Container = [Dimension, Weight, Date],
        (
        LimWei > 30 -> random(1, 30, Weight);
                                random(1, LimWei, Weight)
                                ),
        (
        LimDim > 30 -> random(5, 30, Dimension);
                                random(5, LimDim, Dimension)
                                ),
```

```
        random(5, 30, Date).

getContainerDimension(Container, Dimension):-
        nth1(1, Container, Dimension).
getContainerWeight(Container, Weight):-
        nth1(2, Container, Weight).
getContainerDate(Container, Date):-
        nth1(3, Container, Date).
getContainerManipTime(Container, Time):-
        nth1(1, Container, Dimension),
        nth1(2, Container, Weight),
        Time is Dimension / 5 + Weight / 4.

readPort([], C).
readPort([H|T], C):-
        getContainerDate(H, D),
        (
                D > C -> readPort(T, C);
                D = 0 -> dispatcheContainer(H), readPort(T, C);
                readPort(T, D)
        ).

placeOnSpot([Pile|T], Container, PossPile, PossSpot, [Pile|T2]):-
        PossPile > 1,
        NPossPile is PossPile - 1,
        placeOnSpot(T, Container, NPossPile, PossSpot, T2).
placeOnSpot([Pile|T], Container, PossPile, PossSpot, [NPile|T]):-
        PossPile = 1,
        placeOnPile(Pile, Container, PossSpot, NPile).
```

```
placeOnPile([C|T], Container, PossSpot, [C|T2]):-
        PossSpot > 1,
        NPossSpot is PossSpot - 1,
        placeOnPile(T, Container, NPossSpot, T2).
placeOnPile([H|T], Container, PossSpot, NPile):-
        PossSpot = 1,
        append(Container, H,Aux),
        append(Aux, T, NPile).

placeOnEmpty([_|Told], Container, Empty, Counter, [[Container]|Told]):-
        Empty = Counter.
placeOnEmpty([Hold|Told],Container, Empty, Counter, [Hold|Tnew]):-
        Empty > Counter,
        NCounter is Counter + 1,
        placeOnEmpty(Told,Container, Empty, NCounter, Tnew).

getWeightOnTop([], Counter, Counter).
getWeightOnTop([C|T], Counter, Weight):-
        getContainerWeight(C, Weight),
        NWeight is Weight + Counter,
        getWeightOnTop(T, NWeight, Weight).

getNextElement([H|T], E):-
        E = H.

dispatcheContainer(H):-
        H = 0 .
```