Port Management of Containers

An Optimisation Problem using Constraint Logic Programming

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Authors
Nuno Miguel Ladeira Neto - 201406003
Vasco Manuel dos Santos Pereira - 201302819

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

23 de Dezembro de 2016

1 Introduction

The port management of containers is a very large and challenging task. Containers have different sizes and weights and that causes some stack difficulties in small size restrictions, further more the container time in the port is another variable that concerns optimal management cargo expeditions considering arrivals and departure containers.

The main goal of this optimisation problem is regarded while meeting some constraints and minimizing some parameters, namely the time of container expedition. Section 2 describes this problem in detail, explaining the criteria for optimisation. Section 3 describes how the problem has been modelled as both a constraint satisfaction problem, and an optimisation problem. Subsection 3.1 describes the decision variables and their domains; subsection 3.2 describes the constraints which have been taken into account, as well as their Prolog implementation; subsection 3.3 discusses the objective function in this optimisation problem; subsection 3.4 goes through the set of labelling options which implement the search strategy chosen for this problem. Section 4 explains how the output of the Prolog program is presented to the user. Section 5 discusses some of the achieved results. At last, section 6 presents the main conclusions of this project, and discusses possible future work.

2 Problem Description

Our port has limited space and the operations are handled by cranes which stack them in the port. Handling each container takes an amount of time related to their dimensions and weight. A container can only be handled if you do not have any containers on top. When a ship arrives, it has containers of different sizes stacked according to the space. The stacking of containers in the port have restrictions: larger containers can not be stacked over smaller ones; the total weight of the containers placed on top of a container can not exceed a constant times the weight of the container itself; a stack of containers can have no more than N containers, where N depends on the size of the container. Each container entering in the port has a date scheduled for shipment by lorries. By ordering these outputs temporarily, it is intended to minimize the shipping time of the set of containers. Shipping a container means manipulating it, that is, picking it up with a crane and place it on top of the truck that will carry it. If there is containers above it will have to be handled first.

3 Approach

This section explains how the problem described in section 2 has been modelled as both a constraint satisfaction problem, and an optimisation problem.

This Porject goes to the config file to retrieve information about the Port, Height, Lines and Columns. And a Boat containing Containers

Each container is represented like: [ID, Type, Weight, Expedition]

The Port is a list of ID's, each representing a Container ID. The Port contains all the possible Positions for Containers to be placed. So Height * Lines * Columns

The Port can be represented with small lists of 'Height' elements. This small lists represent a Position were Containers will be pilled. Those small list have restricitons, the first ones represent the top containers, so this containers must have less or equal Weight and must have more or equal Type, expedition Times do not matter in this ocasion.

The relative Matrix position has great value, from the Heigth and Columns, we obtain the the XY position Time starting at (1,1). So if The pilling distances from

(0,0) position, then the bigger is the time. So there is a tendency to pille up near (0,0) coordinates. Also in this case the crane must dislocate from (0,0) to (X,Y) and then back (0,0) simulating a dislocation and expedition time. Therefore the tendecy to pile near the origin (crane) coordinates.

There is a 'ghost' container with ID=1, this container has no Expedition Time, no Weight and a Type of 9999, this ID fills all the rest of the Port, the other IDs from 2 to N, can only be used once, The ghost container Type, helps to be always in top of the others and never up. And his weight and expediton time provides a Time of 0 lifting this container.

The containers Time is not a exact time, it is a relative time, or also a Factor time. The formula for each container is: Time (1/Type) * Weight * Exp * Index. Index is the distance to the top. This factor basically puts the containers that 'create' the smallest disturbance at the top. The bigger the Type, the smaller the time. The Lighter the Container is, the smaller the time. The Smaller the Expedition Time, the smaller the time. The closest to the Top, the smaller the time.

This priorities added to the XY time of the crane, helps to have a close up and efficient pilling system.

Also we tried to used exact times, simulating Expeditions, however it took so much time to compute we were not able to evalute solutions, therefore we used the 'Factor' solution.

3.1 Decision Variables

The decision variables are all the Positions of the Port, and their domain goes from 1 to N, N being the length of the Boat. Although the ID 1 is reserved, this one is a 'ghost' container. Basically fills the all empty positions, that is why his Expedition time is 0, his Weight is 0 and his Type is 9999 (to be always on top of the others).

From 2 to N, there can only be 1 of each ID, this ID's represent existing Containers with Valid Types, Weights and Expedition Times.

3.2 Constraints

The only restrictions used were used according to the Type and Weight when labelling the List, each Position (group of Height containers consecutive) has pilled containers. The top containers must have equal or bigger Type and less or equal Weight.

3.3 Objective Function

From the solution, we know the Efficient solution of pilling the containers in order to have a minimized time for expedition.

The solution is a list of ID's, those ID's are refereed in the boat list as containers. For each 'Height' Containers we have a position and for each Line and Column we have a Matrix of 'Height' Containers as positions. Positions cannot interfere with each position, the first containers of each Position are the top ones.

From that we can know exactly how the Port was generated.

3.4 Search Strategy

The search strategy used was decided by the CLPR library, we only added two options to the labelling: minimize(Time) and $\text{time}_o ut(60000,)$

The first helps labeling with a minimized Time and the other displays the first best solution calculated in the first 60 seconds.

4 Solution Presentation

The predicate defined in block 1 displays the output of the model in a human-readable way.

Block 1: Predicate for displaying the results in a human-readable way

```
        write(Time), nl,
        write(Port), nl.
```

Once the calculations are complete it is displayed the Time for expedition and the acording Port for that Time.

5 Results

This Figure represents a exact solution for a problem. With 2 Lines, 2 Columns and 5 Height.

```
| ?- solve.
_192195
302000
[1,12,14,11,2,1,1,1,1,3,1,1,13,9,7,8,5,10,6,4]
yes
| ?-
```

Figura 1: Exact Result

This figure shows a result with a timeout of 10 seconds, as we can see, the time displayed has doubled, and the Port is different from the above one

```
| ?- solve.
_192195
604000
[1,1,1,1,3,1,1,13,9,7,1,12,14,11,2,8,5,10,6,4]
yes
```

Figura 2: Timeout Result

6 Conclusions

The authors have attempted to show that it is, however, possible to achieve better results if some conditions are met when defining the objective function and the search strategy.

It has also been made clear that the efficiency of the solver depends deeply on the labelling strategy which is deployed. Future work on this project could lie on the definition of custom search strategies capable of leading to a more efficient and precise solutions.

The authors hope that this work serves to show how constraint logic programming techniques can be used to solve optimisation problems efficiently.

For further conclusions read the Source Code documentation.

7 Source Code

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

% ----------------------------------------------------------------
:- include('config.pl').

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

        /*
        Solver function, creates a port which is all the possible position for a
            ↪ container (per line, column and height position)
        from the unique ID's the container has, creates Types, Weigths and
            ↪ Expeditions list. Each index correspondes to the containers ID
        given the Boat length, we know all the domain, and given the Port len we know
            ↪  the cardinality, ID 1 is a special ID represents empty spaces, this
            ↪ container, has Exp 0, Weight 0 and Type 9999 in order to allways be
            ↪ placed on top of the others and in order to be a 'ghost' container
            ↪ for the total time,
        We place the restrictions each 'Heigth' elements, representing the containers
            ↪  in the same coordinates, the restrictions do not extend to other
            ↪ coordinates, for they do not interfere with side containers only top
            ↪ or bottom
        The restrictions are the upper container must be lighter or of equal weight
            ↪ to the down,and the type of the upper must be bigger or equal of the
            ↪ down one
        calculate time according to expression

        Time of Position: 2 * CraneTime + TotalContainers

        CraneTime: sqrt(X*X + Y*Y), starting at (1,1)
        TotalContainers: Sum of all individual containersTime

        containerTime: 1/Type * Exp * Weight

        Then we try to label with the most minimezed time possible in the first 60
            ↪ seconds and display

        */

solve:-
                create_port(Port, Len, Heigth,_, Columnns),
                create_types(Types),
                create_heigths(Weigths),
                create_expeditions(Expeditions),

                boat_length_sum(N, _),
                domain(Port, 1, N),

                create_cardinality(Len, N, Card),
                global_cardinality(Port, Card),

                create_restrictions(Port, Heigth, Types, Weigths),


                timing(Port,Len,Heigth, Columnns, Types, Weigths, Expeditions, Time),
                write(Time), nl,

                labeling([minimize(Time), time_out(60000, _)],Port),

                write(Time), nl,
                write(Port), nl.

%=======================================================
/*
        Creating Port from Heigth, Lines, Columnns
*/
create_port(Port, N, Heigth,Lines, Columnns):-

                port_lines(Lines),
                port_columns(Columnns),
                port_heigth(Heigth),
                N is Lines * Columnns * Heigth,
                length(Port, N).
```

```
%=======================================================
/*
        Retriving boat length (amount of different ID's to label), and sum of all ID'
            ↪ s
*/
boat_length_sum(N, Sum):-

                boat(Boat),
                length(Boat, N),
                get_sum_id(Boat, Sum).

/*
        Get Sum of ID's
*/
get_sum_id([], Sum):-   Sum  = 0.
get_sum_id([[ID | _ ] | Rest], Sum):-

                get_sum_id(Rest, Sum2),
                Sum is ID + Sum2.

%=======================================================
/*
        Create cardinality, first with the ammount of special characters (1) and then
            ↪  the Rest,
        Given that from 2 to N, we can only have one of each, but the Rest with 1's
            ↪ we can create a cardinality to fill all the port
*/
create_cardinality(Len, N, Card):-

                Num is N - 1,
                Zeros is Len - Num,
                create_cardinality_rest(2, N, Rest),
                append([1-Zeros], Rest, Card).
/*
        from 2 to N we can get a individual cardinality for each ID, only being
            ↪ represented 1 time.
*/
create_cardinality_rest(Min, N, Rest):-

                Min == N,
                Rest = [Min-1].

create_cardinality_rest(Min, N, Rest):-

                append([Min-1], Card, Rest),
                NewMin is Min + 1,
                create_cardinality_rest(NewMin, N, Card).

%=======================================================
/*
        Creates List of Types from the Boat, each Index is the Container ID
*/
create_types(Types):-

                boat(Boat),
                create_types_rest(Boat, Types).

create_types_rest([], Types):-  Types = [].
create_types_rest([ [_,Type,_,_] | Rest], Types):-

                create_types_rest(Rest, Types2),
                append([Type], Types2, Types).

%=======================================================
/*
        Creates List of Weights from the Boat, each Index is the Container ID
*/
create_heigths(Heigths):-

                boat(Boat),
                create_heigths_rest(Boat, Heigths).

create_heigths_rest([], Heigths):-      Heigths = [].
create_heigths_rest([ [_,_,Heigth,_] | Rest], Heigths):-

                create_heigths_rest(Rest, Heigths2),
                append([Heigth], Heigths2, Heigths).
```

```
%=======================================================
/*
        Creates List of Expedition Times from the Boat, each Index is the Container
            ↪ ID
*/
create_expeditions(Expeditions):-

                boat(Boat),
                create_expeditions_rest(Boat, Expeditions).

create_expeditions_rest([], Expeditions):-      Expeditions = [].
create_expeditions_rest([ [_,_,_,Exp] | Rest], Expeditions):-

                create_expeditions_rest(Rest, Expeditions2),
                append([Exp], Expeditions2, Expeditions).

%=======================================================
/*
        Create Restrictions in the Port for each Heigth Positions,
        The first container in the sub-list of Heigth containers must have equal or
            ↪ bigger Type or equal or lighter Weight

*/
create_restrictions(Port, Heigth, Types, Weigths):-


                create_restrictions_rest(Port, Port, Heigth, Heigth, Types, Weigths).

/*
        The only different is receiving the same list twice
*/
create_restrictions_rest([_ | []], _, _, _, _, _).
create_restrictions_rest([_ | Rest], [_ | [_ | _]],1 , Heigth, Types, Weigths):-

                create_restrictions_rest(Rest, Rest, Heigth, Heigth, Types, Weigths).

create_restrictions_rest([ID1 | Rest], [_ | [ID2 | _]], Min, Heigth, Types, Weigths)
    ↪ :-

                element(ID1, Types, Type1),
                element(ID2, Types, Type2),

                element(ID1, Weigths, Weigth1),
                element(ID2, Weigths, Weigth2),

                Type1 #>= Type2,
                Weigth1 #=< Weigth2,

                NewMin is Min - 1,
                create_restrictions_rest(Rest, Rest, NewMin, Heigth, Types, Weigths).

%=======================================================
/*
        Calculating the Total time of the Port
*/
timing(Port,Len, Heigth, Columnns, Types, Weigths, Expeditions, Time):-

                total_timing(Port,Len, Heigth, Columnns, Types, Weigths, Expeditions
                    ↪ ,1, Time).

/*
        Calculates Time for each individual 'Heigth' of containers sub-list from Port
            ↪ .
        This happens because some indexes do not interfere in timing calculations
            ↪ with previous indexes.
        Creates the sub-list, calculates time, calculates next sub-list, calculates
            ↪ time ... and so on...
*/
total_timing(_,Len, _, _, _, _,Index,Time):- Index >= Len, Time is 0.
total_timing(Port, Len, Heigth, Columnns, Types, Weigths, Expeditions, Index, Time):-

                get_list(Port, Heigth, Index, Vars), % write(Vars), nl,
                calculate_vars_time(Vars, Heigth, Columnns, Types, Weigths,
                    ↪ Expeditions,Index, Time1),

                NewIndex is Index +  Heigth,
```

```
                total_timing(Port, Len, Heigth, Columnns, Types, Weigths, Expeditions
                    ↪ , NewIndex, Time2),
                Time #= Time1 + Time2.

/*
        Create sub-list of Heigth Positions
*/
get_list(_, 0, _, Vars):-       Vars = [].
get_list(Port, Heigth, Index, Vars):-

                nth1(Index, Port, Var),
                NewIndex is Index + 1,
                NewHeigth is Heigth - 1,
                get_list(Port, NewHeigth, NewIndex, Vars2),
                append([Var], Vars2, Vars).
/*
        First the XY positions correspondingly then calculates sub-list time.
        The XY position affects very much, it is multiplied to the Time of the Sub-
            ↪ list,
        so if to distant in XY the bigger the time will be.
        So it tends to be near (0,0) for smaller time.
*/
calculate_vars_time(Vars, Heigth, Columnns, Types, Weigths, Expeditions,Index,  Time)
    ↪ :-

                calculate_x_y_time(Index, Heigth, Columnns, TimeXY),
                get_vars_time(Vars, TimeXY, Types, Weigths, Expeditions, 1, Time1),
                Time #= TimeXY * Time1.
/*
        Calculates XY coordinates from only one INDEX and from the Heigth and Columns
            ↪  of the Port
*/
calculate_x_y_time(Index, Heigth, Columnns, TimeXY):-


                I is Index - 1,
                I1 is I // Heigth,

                Y is I1 // Columnns,
                X is I1 rem Columnns,

                NewY is Y + 1,
                NewX is X + 1,

                Diff2 is sqrt(NewX * NewX + NewY * NewY),
                Diff is integer(Diff2),

                TimeXY is 2* Diff * 100.
/*
        Calculates each container Time by the formula:


        Time: (1/Type) * Exp * Weight * Index,

        Time is not exact, it is more of a Factor Value.

        If the Expedition Time is small and is at the bottom, then it will tend to go
            ↪  up because a small Index and a small Exp gives smaller Time,
        If the Expedition Time is big and is at the top, it tends to be there unless
            ↪ there is containers with smaller Expedition Times, providing more
            ↪ optimization

        If the Type is smaller it tends to go up as well for 1/Type provides a good
            ↪ factor, unless other types bigger go up first for better time

        If lighter, it tends to go up for it is easier to move lighter objects
*/
get_vars_time([], _, _, _, _, _, Time):-        Time #= 0.
get_vars_time([ID | Rest], TimeXY, Types, Weigths, Expeditions, Index, Time):-

                element(ID,Types,Type),
                element(ID,Weigths,Weigth),
                element(ID,Expeditions,Exp),

                Time1 #= 1 / Type * Exp * Weigth * Index,

                I is Index + 1,
```

```
get_vars_time(Rest, TimeXY, Types, Weigths, Expeditions, I, Time2),
Time #= Time2 + Time1.
```

8 Unused Code

8.1 File *port.pl*

```
/*
        Bellow we show a previous way we tried to implement. But unfortunatelly the
            ↪ huge calculation times prevented us from implementing it
        This one is actually based on real time, and calculate the exact time of each
            ↪  postion, how many containers should be lifted for one to be expeled,
            ↪  or if there was nothing to be lifted.
*/
/*
:- include('config.pl').

:- use_module(library(lists)).
:- use_module(library(clpfd)).
solve:-
                create_port(Port, Len, Heigth,_, Columnns),
                create_types(Types),
                create_heigths(Weigths),
                create_expeditions(Expeditions),

                boat_length_sum(N, _),
                domain(Port, 1, N),

                create_cardinality(Len, N, Card),
                global_cardinality(Port, Card),

                create_restrictions(Port, Heigth, Types, Weigths),

                %write(Port), nl,
                %timing(Port,Len,Heigth, Columnns, Types, Weigths, Expeditions, Time)
                    ↪ ,
                %write(Time), nl,

                timing(Port,Len,Heigth, Columnns, Types, Weigths, Expeditions, Time),
                write(Time), nl,

                minimize(labeling([],Port), Time),

                write(Time), nl,
                write(Port), nl.

%=======================================================
create_port(Port, N, Heigth,Lines, Columnns):-

                port_lines(Lines),
                port_columns(Columnns),
                port_heigth(Heigth),
                N is Lines * Columnns * Heigth,
                length(Port, N).
%=======================================================
boat_length_sum(N, Sum):-

                boat(Boat),
                length(Boat, N),
                get_sum_id(Boat, Sum).


get_sum_id([], Sum):-   Sum  = 0.
get_sum_id([[ID | _ ] | Rest], Sum):-

                get_sum_id(Rest, Sum2),
                Sum is ID + Sum2.
%=======================================================
create_cardinality(Len, N, Card):-

                Num is N - 1,
                Zeros is Len - Num,
                create_cardinality_rest(2, N, Rest),
                append([1-Zeros], Rest, Card).

create_cardinality_rest(Min, N, Rest):-

                Min == N,
                Rest = [Min-1].
```

```
create_cardinality_rest(Min, N, Rest):-

                append([Min-1], Card, Rest),
                NewMin is Min + 1,
                create_cardinality_rest(NewMin, N, Card).
%=======================================================
create_types(Types):-

                boat(Boat),
                create_types_rest(Boat, Types).

create_types_rest([], Types):-  Types = [].
create_types_rest([ [_,Type,_,_] | Rest], Types):-

                create_types_rest(Rest, Types2),
                append([Type], Types2, Types).

%=======================================================
create_heigths(Heigths):-

                boat(Boat),
                create_heigths_rest(Boat, Heigths).

create_heigths_rest([], Heigths):-      Heigths = [].
create_heigths_rest([ [_,_,Heigth,_] | Rest], Heigths):-

                create_heigths_rest(Rest, Heigths2),
                append([Heigth], Heigths2, Heigths).

%=======================================================
create_expeditions(Expeditions):-

                boat(Boat),
                create_expeditions_rest(Boat, Expeditions).

create_expeditions_rest([], Expeditions):-      Expeditions = [].
create_expeditions_rest([ [_,_,_,Exp] | Rest], Expeditions):-

                create_expeditions_rest(Rest, Expeditions2),
                append([Exp], Expeditions2, Expeditions).

%=======================================================
create_restrictions(Port, Heigth, Types, Weigths):-

create_restrictions_rest(Port, Port, Heigth, Heigth, Types, Weigths).

create_restrictions_rest([_ | []], _, _, _, _, _).
create_restrictions_rest([_ | Rest], [_ | [_ | _]],1 , Heigth, Types, Weigths):-

create_restrictions_rest(Rest, Rest, Heigth, Heigth, Types, Weigths).

create_restrictions_rest([ID1 | Rest], [_ | [ID2 | _]], Min, Heigth, Types, Weigths)
    ↪ :-

                element(ID1, Types, Type1),
                element(ID2, Types, Type2),

                element(ID1, Weigths, Weigth1),
                element(ID2, Weigths, Weigth2),

                Type1 #>= Type2,
                Weigth1 #=< Weigth2,

                NewMin is Min - 1,
                create_restrictions_rest(Rest, Rest, NewMin, Heigth, Types, Weigths).

%=======================================================
timing(Port,Len, Heigth, Columnns, Types, Weigths, Expeditions, Time):-

                max_member(MaxExp, Expeditions),
                total_timing(Port,Len, Heigth, Columnns, Types, Weigths, Expeditions
                    ↪ ,1,1, MaxExp, Time).

total_timing(_,Len, _, _, _, _,Index, _, _, Time):-  Index >= Len, Time is 0.
total_timing(Port, Len, Heigth, Columnns, Types, Weigths, Expeditions, Index, MinExp,
    ↪  MaxExp, Time):-
```

```
                get_list(Port, Heigth, Index, Vars), % write(Vars), nl,
                calculate_vars_time(Vars, Heigth, Columnns, Types, Weigths,
                    ↪ Expeditions,Index, MinExp, MaxExp, Time1),

                NewIndex is Index +  Heigth,

                total_timing(Port, Len, Heigth, Columnns, Types, Weigths, Expeditions
                    ↪ , NewIndex, MinExp, MaxExp, Time2),
                Time #= Time1 + Time2.


get_list(_, 0, _, Vars):-       Vars = [].
get_list(Port, Heigth, Index, Vars):-

                nth1(Index, Port, Var),
                NewIndex is Index + 1,
                NewHeigth is Heigth - 1,
                get_list(Port, NewHeigth, NewIndex, Vars2),
                append([Var], Vars2, Vars).

calculate_vars_time(Vars, Heigth, Columnns, Types, Weigths, Expeditions,Index, MinExp
    ↪ , MaxExp, Time):-

                calculate_x_y_time(Index, Heigth, Columnns, TimeXY),
                get_vars_time(Vars, TimeXY, Types, Weigths, Expeditions, MinExp,
                    ↪ MaxExp, Time).

calculate_x_y_time(Index, Heigth, Columnns, TimeXY):-


                I is Index - 1,
                I1 is I // Heigth,

                Y is I1 // Columnns,
                X is I1 rem Columnns,

                NewY is Y + 1,
                NewX is X + 1,

                Diff2 is sqrt(NewX * NewX + NewY * NewY),
                Diff is integer(Diff2),

                TimeXY is 2* Diff * 100.

get_vars_time(_, _, _, _, _, MinExp, MaxExp, Time):-    MinExp > MaxExp, Time is 0.
get_vars_time(Vars, TimeXY, Types, Weigths, Expeditions, MinExp, MaxExp, Time):-

                get_vars_time_each(Vars, Types, Weigths, Expeditions, MinExp, Time1),

                Time1 #= 0 #<=> B,
                eval_time(Time1, TimeXY, B, NewTime1),

                NewMin is MinExp + 1,
                get_vars_time(Vars, TimeXY, Types, Weigths, Expeditions, NewMin,
                    ↪ MaxExp, Time2),

                Time #= NewTime1 + Time2.

eval_time(_, _, B, NewTime1):-  B == 1, NewTime1 is 0.
eval_time(Time1, TimeXY, B, NewTime1):- NewTime1 #= Time1 + TimeXY.

get_vars_time_each([], _, _, _, _, Time):-      Time is 0.
get_vars_time_each([ID | Rest], Types, Weigths, Expeditions, MinExp, Time):-

                get_vars_time_each(Rest, Types, Weigths, Expeditions, MinExp, Time2),

                Time2 #= 0 #<=> B,
                eval_time_bellow(ID, Types, Weigths, Expeditions, MinExp, B, Time1),

                Time #= Time1 + Time2.

eval_time_bellow(ID, Types, Weigths, Expeditions, MinExp, B, Time):-

                B == 1,

                element(ID, Expeditions, Exp),
```

```
                Exp #= MinExp #<=> B2,
                eval_time_bellow_exp(ID, Types, Weigths, B2, Time).


eval_time_bellow(ID, Types, Weigths, _, _, B, Time):-

                element(ID, Types, Type),
                element(ID, Weigths, Weigth),

                get_time_exp(Type, Weigth, Time).

eval_time_bellow_exp(_, _, _, B, Time):-        B == 0, Time is 0.
eval_time_bellow_exp(ID, Types, Weigths, B, Time):-

                element(ID, Types, Type),
                element(ID, Weigths, Weigth),

                get_time_exp(Type, Weigth, Time).

get_time_exp(Type, Weigth, Time):-

                Factor #= 1 / Type,
                Time #= Factor * 60 * Weigth.
*/
```

9 Data files

9.1 File *config.pl*

```
port_lines(1).
port_columns(4).
port_heigth(5).

boat([[1,9999,0,0],[2,1,54,2],[3,3,206,5],[4,3,297,2],[5,3,219,1],
[6,3,283,3],[7,2,193,4],[8,3,168,1], [9,2,106,5],[10,3,280,2],[11,1,46,4],
[12,1,12,3],[13,3,40,3],[14,1,18,3]]).
```