# Colocação de Livros

## Relatório Final


Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

**Grupo Colocacao de Livros_3:**

Francisco Couto - ei12189@fe.up.pt
Joel Dinís - ei12064@fe.up.pt

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

13 de Dezembro de 2015

## Resumo

Como segundo e último trabalho da unidade curricular de Programação em Lógica e de forma a avaliar a matéria relativa à programação em lógica com restrições, foi-nos proposta a resolução de um problema de colocação de livros.

O seu principal objectivo seria, respeitando restrições da prateleira como a largura e a altura máxima, dispôr os livros ordenados por diferentes variáveis, nomeadamente autor, tema, nome e data de publicação.

Dessa forma, a estratégia adotada pretende, existindo uma listagem de tuplos livro e prateleira (book/shelf) cujos atributos como altura, largura são conhecidos, agrupa-los de forma otimizada utilizando uma estratégia de constante avaliação através do *labeling*, ao mesmo tempo que forçávamos o cumprimentos das restrições acima citadas.


## 1. Introdução

O objectivo deste trabalho é implementar a resolução de um problema de otimização, dizendo respeito à distribuição de livros por uma ou mais prateleiras.

Este relatório terá como objectivo explicar mais detalhadamente o problema, bem como a abordagem tomada de forma a obter uma solução válida para este; a explicação da forma usada para representar o estado final do problema; os resultados diferentes obtidos com diferentes variáveis e as suas conclusões.

## 2. Descrição do Problema

Neste problema pretende-se automatizar a colocação e distribuição de livros por diferentes prateleiras. Cada prateleira tem uma largura e uma altura que não devem ser ultrapassadas pelos livros que queremos colocar, mas que ao mesmo tempo devem ser preenchidas o máximo possível.

## 3. Abordagem

Cada livro é definido como um tuplo do tipo *book(id, name, author, theme, day, month, year, width, height)* e cada prateleira por *shelf(id, width, height)*.

## a. Variáveis de decisão

Neste problema as variáveis de decisão passam por uma lista de índices que identificam cada livro. Os seus valores de domínio dependem do numero de livros que queremos ordenar, ou seja, existindo X livros a colocar nas várias prateleiras, sendo as variáveis distintas, o seu valor estará no intervalo de 1 a um valor máximo X.

## b. Restrições

Este problema contém várias restrições. À partida, os livros a colocar nas prateleiras não poderão ultrapassar as larguras das prateleiras nem tão-pouco a sua altura máxima.

Por outro lado, apesar destas restrições a ideia será obter o maior proveito possível dos seus valores deixando o menor espaço possível na prateleira e minimizando a altura média.

O utilizador poderá ainda escolher opções para ordenar os livros colocados nas prateleiras por vários aspectos, nomeadamente autor, nome, tema e data de publicação.

Os predicados responsáveis pelo estabelecimento destas regras são:

1. *widthRestriction(IdShelf, BookVars, WidthSum)*: Obtém os valores de largura para a lista de livros e compara a sua soma (WidthSum) com a largura da prateleira com o IdShelf;

2. *heightRestriction(IdShelf, BookVars, AverageHeight)*: Predicado semelhante ao primeiro, porém com uma restrição extra de não poder exceder a altura máxima definida da prateleira respectiva;

## c. Função de Avaliação

A solução poderá ser avaliada e testada para sabermos a sua qualidade somando as alturas e larguras finais presentes no *output do problema* e verificando que cumprem as restrições e que de facto ocupam o máximo possível de cada estante.

## d. Estratégia de Pesquisa

Para a implementação do *labeling* foram sequenciadas todas as restrições da seguinte maneira:

-Restrição de largura: a etiquetagem encontra a melhor sequência de livros para que minimize o espaço livre numa prateleira

-Restrição de altura: a etiquetagem restringe todos os livros com altura maior que a altura da prateleira e minimiza a variação das alturas dos livros, para que fiquem apenas os livros com altura semelhante que preencham o máximo da largura da prateleira

-Ordenação pela condição escolhida pelo utilizador ordena a lista de livros, que obedecem a todas restrições aplicadas até agora, para que os livros encontrados pela etiquetagem estejam sequenciados pela condição escolhida

# 4. Visualização da Solução

É feita a impressão da solução num modo *user friendly* utilizando o predicado *printShelf* que, para cada prateleira, imprime os livros que esta contém. Caso uma prateleira não contenha qualquer livro é impressa uma mensagem esclarecedora que ajuda o utilizador a perceber isso.

Para que isto aconteça é lhe passado uma lista de listas que contém os id's de cada livro.

```
| ?- main(1).

Time: 1.02s

Resumptions: 112409
Entailments: 40586
Prunings: 112239
Backtracks: 27972
Constraints created: 258

 -- Sorted by the name of the book

1 -- Shelf - 10x50cm

  -> A Viagem do Elefante,          Saramago          Romance          28-2-1995  2x15cm
  -> Memorial do Covento,           Saramago          Romance          19-7-1995  2x15cm
  -> Mensagem,                       Pessoa           Poetry           23-7-1995  3x20cm
  -> O Crime do Padre Amaro,           Eca             Romance          22-7-1995  3x20cm

-------------------------------------------------------------------------------------------------------------

2 -- Shelf - 15x20cm

  -> A Cidade e as Serras,             Eca             Classic          24-7-1995  3x20cm
  -> Lusiadas,                      Camoes             Classic          18-7-1995  3x15cm
  -> Maias,                            Eca             Classic          18-7-1995  3x15cm

-------------------------------------------------------------------------------------------------------------

3 -- Shelf - 20x15cm

  -> No books in this shelf

-------------------------------------------------------------------------------------------------------------
```

## 5. Resultados

Nos testes efetuados com vários livros e prateleiras obtivemos sempre tempos inferiores a 10 segundos para a resolução eficaz do problema. É de esperar, no entanto, que este valor aumente à medida que também se aumentam o número de livros e prateleiras. Ficam alguns resultados demonstrativos para três prateleiras:

| Nº de Livros | Sem ordenação | Ordenados p/ Autor |
|---|---|---|
| 5 | 0.0s | 0.08s |
| 6 | 0.07 | 0.21 |
| 7 | 0.81s | 1.03s |
| 8 | 10.43s | 11.06s |

Pelos resultados obtidos, bem como pela análise do enunciado é a opinião do grupo que o trabalho atingiu os objetivos propostos utilizando os métodos que eram pedidos.

## 6. Conclusões e Trabalho Futuro

Este trabalho é demonstrativo do uso benéfico da programação em lógica com restrições para resolver este tipo de problemas sendo uma linguagem funcional, porém apresenta algumas limitações quando se tratam de problemas de maior complexidade.

A solução implementada não aparenta ter quaisquer restrições ao nível do *input* fornecido.

## Bibliografia

https://sicstus.sics.se/
http://www.swi-prolog.org/
https://stackoverflow.com/

# Anexo

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

%shelf(id, width, height)
shelf(1, 10,    50).
shelf(2, 10,    20).
shelf(3, 20, 15).
%shelf(4, 50, 20).
%shelf(5, 20, 25).

%book(id, name, author, theme, day, month, year, width, height)
book(1, 'Memorial do Covento', 'Saramago', 'Romance', 19, 7, 1995, 2, 15).
book(2, 'Lusiadas',Camoes',         'Classic',       18, 7, 1995,    3,  15).
book(3, 'Maias', 'Eca',  'Classic', 18, 7, 1995,  3,  15).
book(4, 'A Viagem do Elefante', 'Saramago', 'Romance',  28, 2, 1995, 2,  15).
book(5, 'O Crime do Padre Amaro', 'Eca',  'Romance', 22, 7, 1995,  3,          20).
book(6, 'Mensagem', 'Pessoa',  'Poetry',  23, 7, 1995, 3, 20).
book(7, 'A Cidade e as Serras', Eca', 'Classic',    24, 7, 1995, 3, 20).
%book(8, 'Inferno',Dan Brown','Fiction',24, 7, 1995, 2,15).
%book(9, 'Conspiracao','Dan Brown','Romance',        18, 7, 1995,    2,       15).

%-----------------------------------------------

%returns a list of all the shelves ids
listShelves(List) :-
        findall(Id,
                        shelf(Id, _, _),
                        List).

%returns a list of all the books ids
listBooks(List) :-
        findall(Id,
                        book(Id, _, _, _, _, _, _, _, _),
                        List).

%creates a list with empty vars
createVarList(0, []).
createVarList(N, [V|MoreVars]) :-
```

```
        N > 0,
        M is N - 1,
        createVarList(M, MoreVars).

%returns a sublist of the original list and a list of the vars not listed
sugestBooks([], [], []).
sugestBooks([First|Rest], [First|Sub], NotUSed) :- sugestBooks(Rest, Sub,
NotUSed).
sugestBooks([First|Rest], Sub, [First|More]) :- sugestBooks(Rest, Sub, More).

%-----------------------------------------------

%removes duplicates of an element in a list
removeDuplicates([], []).
removeDuplicates([First|Rest], Result) :-
        removeDuplicates(Rest, TempResult),
        (
                member(First, TempResult) ->
                (
                        Result = TempResult
                );
                (
                        append([First], TempResult, Result)
                )
        ).

%returns a list of positions of elements in a list
getMap([], _, []).
getMap([First|Rest], DescList, [ThisResult|MoreResults]) :-
        getMap(Rest, DescList, MoreResults),
        nth1(ThisResult, DescList, First).

%returns a list of all the appearances of an element in a list
getAllPositionsOfElement([], _, _, []).
getAllPositionsOfElement([First|Rest], First, Index, [Index|RestOfIndex]) :-
        NextIndex is Index + 1,
        getAllPositionsOfElement(Rest, First, NextIndex, RestOfIndex).
getAllPositionsOfElement([_|Rest], Elem, Index, Indexes) :-
        NextIndex is Index + 1,
        getAllPositionsOfElement(Rest, Elem, NextIndex, Indexes).

%returns all the positions of an element
```

```
allNth1(Indexes, List, Elem) :- getAllPositionsOfElement(List, Elem, 1, Indexes).

%checks if a position is occupied
auxFindValidPosition(1, [Index|_], _, Index).
auxFindValidPosition(N, Indexes, Positions, Position) :-
	nth1(N, Indexes, Pos),
	(
		member(Pos, Positions) ->
		(
			M is N - 1,
			auxFindValidPosition(M, Indexes, Positions, Position)
		);
		(
			Position = Pos
		)
	).

%searches for a valid position in a list
findValidPosition(Indexes, Positions, FirstPosition) :-
	length(Indexes, Num),
	auxFindValidPosition(Num, Indexes, Positions, FirstPosition).

%searches for positions for every element of a list
findPositions([], _, []).
findPositions([First|Rest], Order, [FirstPosition|MorePositions]) :-
	findPositions(Rest, Order, MorePositions),
	allNth1(Indexes, Order, First),
	findValidPosition(Indexes, MorePositions, FirstPosition).

%gets the elements of a list below a certain index
getFirstElements(1, _, []).
getFirstElements(N, [H|T], [H|R]) :-
	M is N - 1,
	getFirstElements(M, T, R).

%gets the last elements of a list after a certain index
getLastElements(N, N, _, []).
getLastElements(1, _, [_|Rest], Rest).
getLastElements(N, Max, [_|Rest], Result) :-
	M is N - 1,
	getLastElements(M, Max, Rest, Result).
```

```
%replaces an element in a list depending of an index
insertElement(Pos, Elem, List, Result) :-
        getFirstElements(Pos, List, HeadList),
        length(List, MaxLength),
        getLastElements(Pos, MaxLength, List, TailList),
        append(HeadList, [Elem], TempResult),
        append(TempResult, TailList, Result).

%reorders a list depending on a list of positions
auxRearrangeOrder([], [], Result, Result).
auxRearrangeOrder([Var|MoreVars], [FirstPosition|MorePositions], List, Result)
:-
        insertElement(FirstPosition, Var, List, TempResult),
        auxRearrangeOrder(MoreVars, MorePositions, TempResult, Result).

%reorders a list
rearrangeOrder(Vars, Positions, Result) :-
        length(Vars, Num),
        createVarList(Num, List),
        auxRearrangeOrder(Vars, Positions, List, Result).

%---------------------------------------------

%bubblesort sorting algorithm
bubblesort(Rel, List, SortedList) :-
        swap(Rel, List, NewList), !,
        bubblesort(Rel, NewList, SortedList).
bubblesort(_, SortedList, SortedList).

swap(Rel, [A, B|List], [B, A|List]) :-
        check(Rel, B, A).
swap(Rel, [A|List], [A|NewList]) :-
        swap(Rel, List, NewList).

check(Rel, A, B) :-
        Goal =.. [Rel, A, B],
        call(Goal).

%---------------------------------------------

%converts a date to an integer value
dateToInt(Day, Month, Year, Result) :-
```

```
        VYear is Year * 365,
        VMonth is Month * 31,
        VYM is VYear + VMonth,
        Result is Day + VYM.

%-----------------------------------------------

%gets the width of a list of books
getBooksWidthList([], []).
getBooksWidthList([BId|RestOfBooks], [Width|RestOfWidths]) :-
        book(BId, _, _, _, _, _, _, Width, _),
        getBooksWidthList(RestOfBooks, RestOfWidths).

%gets the height of a list of books
getBooksHeightList([], []).
getBooksHeightList([BId|RestOfBooks], [Height|RestOfHeights]) :-
        book(BId, _, _, _, _, _, _, _, Height),
        getBooksHeightList(RestOfBooks, RestOfHeights).

%gets the authors of a list of books
getBooksAuthorList([], []).
getBooksAuthorList([BId|RestOfBooks], [Author|RestOfAuthors]) :-
        book(BId, _, Author, _, _, _, _, _, _),
        getBooksAuthorList(RestOfBooks, RestOfAuthors).

%gets the themes of a list of boooks
getBooksThemeList([], []).
getBooksThemeList([BId|RestOfBooks], [Theme|RestOfThemes]) :-
        book(BId, _, _, Theme, _, _, _, _, _),
        getBooksThemeList(RestOfBooks, RestOfThemes).

%gets the names of a list of boooks
getBooksNameList([], []).
getBooksNameList([BId|RestOfBooks], [Name|RestOfNames]) :-
        book(BId, Name, _, _, _, _, _, _, _),
        getBooksNameList(RestOfBooks, RestOfNames).

%gets the dates of a list of boooks
getBookDates([], []).
getBookDates([BId|RestOfBooks], [Date|RestOfDates]) :-
        book(BId, _, _, _, Day, Month, Year, _, _),
        dateToInt(Day, Month, Year, Date),
```

```
        getBookDates(RestOfBooks, RestOfDates).

%gets all existing authors
getAllAuthors(AllAuthors) :-
        findall(Author,
                        book(_, _, Author, _, _, _, _, _, _),
                        TempList),
        removeDuplicates(TempList, AllAuthors).

%gets all existing names
getAllNames(AllNames) :-
        findall(Name,
                        book(_, Name, _, _, _, _, _, _, _),
                        TempList),
        removeDuplicates(TempList, AllNames).

%gets all existing authors
getAllThemes(AllThemes) :-
        findall(Theme,
                        book(_, _, _, Theme, _, _, _, _, _),
                        TempList),
        removeDuplicates(TempList, AllThemes).

getRestrictionList(Mode, RestrictionList) :-
        Mode =:= 1 ->
        (
                getAllNames(TempList),
                bubblesort(@<, TempList, RestrictionList)
        );

        Mode =:= 2 ->
        (
                getAllAuthors(TempList),
                bubblesort(@<, TempList, RestrictionList)
        );

        Mode =:= 3 ->
        (
                getAllThemes(TempList),
                bubblesort(@<, TempList, RestrictionList)
        );
        (
```

```
            RestrictionList = []
        ).

%---------------------------------------------

%checks if all values of a list are below a certain max value
checkBelowMaxHeight([], _).
checkBelowMaxHeight([Height|MoreHeights], MaxHeight) :-
        Height #=< MaxHeight,
        checkBelowMaxHeight(MoreHeights, MaxHeight).

%implements the max width restriction
widthRestriction(IdShelf, BookVars, WidthSum) :-
        %get shelf size
        shelf(IdShelf, MaxWidth, _),

        %get all width values of the books
        getBooksWidthList(BookVars, WidthVals),

        %imposes the restriction
        sum(WidthVals, #=, WidthSum),
        WidthSum #=< MaxWidth.

%implements the max height restriction and minimizes height variations
heightRestriction(IdShelf, BookVars, AverageHeight) :-
        %get shelf height
        shelf(IdShelf, _, MaxHeight),

        %get the number of vars
        length(BookVars, Size),

        %get all height values of the books
        getBooksHeightList(BookVars, HeightVals),

        %imposes the restriction
        checkBelowMaxHeight(HeightVals, MaxHeight),

        %minimizes variatons of height
        sum(HeightVals, #=, HeightSum),
        AverageHeight #= ( HeightSum / Size ).

%group books by their authors
```

```
orderByAuthorRestriction(AllAuthors, BookVars) :-
        %get books authors
        getBooksAuthorList(BookVars, Authors),

        %association between authors and an Id
        getMap(Authors, AllAuthors, MapAuthors),

        %sorts books by author
        length(MapAuthors, Num),
        createVarList(Num, Null),
        createVarList(Num, Ordered),
        sorting(MapAuthors, Null, Ordered),
        findPositions(MapAuthors, Ordered, Positions),

        %rearranges the vars order
        rearrangeOrder(BookVars, Positions, BookVars).

%group books by their name
orderByNameRestriction(AllNames, BookVars) :-
        %get books themes
        getBooksNameList(BookVars, Names),

        %association between themes and an Id
        getMap(Names, AllNames, MapNames),

        %sorts books by theme
        length(MapNames, Num),
        createVarList(Num, Null),
        createVarList(Num, Ordered),
        sorting(MapNames, Null, Ordered),
        findPositions(MapNames, Ordered, Positions),

        %rearranges the vars order
        rearrangeOrder(BookVars, Positions, BookVars).

%group books by their theme
orderByThemeRestriction(AllThemes, BookVars) :-
        %get books themes
        getBooksThemeList(BookVars, Themes),

        %association between themes and an Id
        getMap(Themes, AllThemes, MapThemes),
```

```
        %sorts books by theme
        length(MapThemes, Num),
        createVarList(Num, Null),
        createVarList(Num, Ordered),
        sorting(MapThemes, Null, Ordered),
        findPositions(MapThemes, Ordered, Positions),

        %rearranges the vars order
        rearrangeOrder(BookVars, Positions, BookVars).

%order books by their date of publication
orderByDateRestriction(BookVars) :-
        %get date values
        getBookDates(BookVars, DateVals),

        %sorts books by date of publication
        length(DateVals, Num),
        createVarList(Num, Null),
        createVarList(Num, Ordered),
        sorting(DateVals, Null, Ordered),
        findPositions(DateVals, Ordered, Positions),

        %rearranges the vars order
        rearrangeOrder(BookVars, Positions, BookVars).

%-----------------------------------------------

%algorithm to fill a shelf correctly
fillShelf(Mode, RestrictionList, IdShelf, BookVars, BookVarsNotUsed, Options,
Ordered) :-
        %sugest books to fill the shelf
        sugestBooks(BookVars, Ordered, BookVarsNotUsed),

        (
                Ordered == [] ->
                (
                        Options = []
                );
                (
                        %always process restrictions with <ordered> which is the list
with a variable number of variables
```

```
                widthRestriction(IdShelf, Ordered, WidthSum),
                heightRestriction(IdShelf, Ordered, AverageHeight),
                (
                        Mode =:= 1 ->
                        (
                                orderByNameRestriction(RestrictionList,
Ordered)
                        );

                        Mode =:= 2 ->
                        (
                                orderByAuthorRestriction(RestrictionList,
Ordered)
                        );

                        Mode =:= 3 ->
                        (
                                orderByThemeRestriction(RestrictionList,
Ordered)
                        );

                        Mode =:= 4 ->
                        (
                                orderByDateRestriction(Ordered)
                        );
                        (
                                true
                        )
                ),

                %defines all options to label
                Options = [maximize(WidthSum), minimize(AverageHeight)]
        )
    ).

%-----------------------------------------------

%fill all shelves with books depending on the restrictions
orderShelf([], _, _, [], [], []). %if there are no more shelves to process, it must not
exist any more books too
orderShelf([], _, _, _, _, _) :- noSolution.
```

```
orderShelf([IdShelf|MoreShelves], Mode, RestrictionList, BookVars, Options,
[Ordered|MoreOrdered]) :-
        %fill every shelf with books that obeys the restrictions
        fillShelf(Mode, RestrictionList, IdShelf, BookVars, BookVarsNotUsed,
Option, Ordered),
        orderShelf(MoreShelves, Mode, RestrictionList, BookVarsNotUsed,
MoreOptions, MoreOrdered),
        append(Option, MoreOptions, Options).

orderShelves(Mode, RestrictionList, LShelves, NBooks, ResultList) :-

        %create a list of vars for every book
        createVarList(NBooks, BookVars),

        %domain is every if of books
        domain(BookVars, 1, NBooks),
        all_distinct(BookVars),

        %process shelves and books
        orderShelf(LShelves, Mode, RestrictionList, BookVars, Options, ResultList),

        %joins more options to labeling
        MoreOptions = [best],
        append(Options, MoreOptions, AllOptions),

        once(labeling(AllOptions, BookVars)).

%-----------------------------------------------

%statistics functions to inform user
reset_timer :- statistics(walltime,_).
print_time :-
        statistics(walltime,[_,T]),
        TS is ((T//10)*10)/1000,
        nl, write('Time: '), write(TS), write('s'), nl, nl.

%-----------------------------------------------

%stops program execution when there are no solutions to the problem
noSolution :- throw('No solution found').

%-----------------------------------------------
```

```
%prints the ordering mode
printMode(Mode) :-
	Mode =:= 0 ->
	(
		write(' -- No sorting'),nl
	);

	Mode =:= 1 ->
	(
		write(' -- Sorted by the name of the book'),nl
	);

	Mode =:= 2 ->
	(
		write(' -- Sorted by the name of the books author'),nl
	);

	Mode =:= 3 ->
	(
		write(' -- Sorted by the books theme'),nl
	);

	Mode =:= 4 ->
	(
		write(' -- Sorted by the books date of publication'),nl
	).

%prints the solution in an easy view
printShelf([]).
printShelf([H|T]) :-
	book(H, Name, Author, Theme, Day, Month, Year, Width, Height),
	format('  -> ~s,~t~35+~t~s~t~25+~t~s~t~25+  ~d-~d-~d  ~dx~dcm~n',
[Name, Author, Theme, Day, Month, Year, Width, Height]),
	printShelf(T).

printResult([], _).
printResult([H|T], N) :-
	M is N + 1,
	shelf(N, Width, Height),
	write(N),write(' -- Shelf -
'),write(Width),write('x'),write(Height),write('cm'),nl,nl,
```

```
        (
                H == [] ->
                (
                        write('  -> No books in this shelf'),nl
                );
                (
                        printShelf(H)
                )
        ),
        nl,

write('-------------------------------------------------------------------------------------------
-------------'),nl,nl,
        printResult(T, M).

%----------------------------------------------

main :-
        nl,
        write('-> Invalid Input!'),nl,nl,
        write('Input format:'),nl,
        write(' main(< order-num >).'),nl,nl,
        write('< order-num > has to be one of the following:'),nl,
        write(' 0 - No order'),nl,
        write(' 1 - Order by the name of the book'),nl,
        write(' 2 - Order by the name of the books author'),nl,
        write(' 3 - Order by the books theme'),nl,
        write(' 4 - Order by the books date of publication'),nl,nl.

%entry for the program
main(Mode) :-
        (
                ( Mode < 0; Mode > 4 ) ->
                (
                        main
                );
                (
                        getRestrictionList(Mode, RestrictionList),
                        listShelves(LShelves),
                        listBooks(LBooks),
                        length(LBooks, NBooks),
                        reset_timer,
```

```
                orderShelves(Mode, RestrictionList, LShelves, NBooks,
ResultList),
                print_time,
                fd_statistics,
                nl,printMode(Mode),
                nl,printResult(ResultList, 1)
            )
        ).
```