Resolução de um Problema de Decisão usando Programação em Lógica com Restrições: C-Note

Diogo Almeida e Pedro Queirós

FEUP-PLOG, Turma 3MIEIC04, Grupo C-Note_3

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

Resumo. Este projeto foi desenvolvido no âmbito da unidade curricular de Programação em Lógica [1] utilizando o Ambiente de Desenvolvimento *Prolog SICStus* [2], com o objetivo de resolver um problema de decisão/otimização utilizando restrições. O problema de decisão escolhido é o puzzle *C-Note* [3], em que se adicionam dígitos antes ou depois do número presente em cada célula da grelha-problema de modo a que a soma de cada linha e de cada coluna desta seja 100. Desta maneira, através da utilização da linguagem de programação *Prolog* e a biblioteca *CLPFD* [4] do *SICStus*, foi possível a resolução e também geração destes puzzles, assunto que será abordado detalhadamente neste artigo.

Keywords: *C-Note*, Problema de Decisão/Otimização, *Prolog*, *SICStus*, FEUP

1 Introdução

O projeto foi desenvolvido no âmbito da unidade curricular de Programação em Lógica do 3º ano do Mestrado Integrado em Engenharia Informática e Computação. O objetivo principal deste é o desenvolvimento de um programa, utilizando Programação em Lógica com Restrições, que forneça a resolução para um problema de decisão ou otimização e que tenha a capacidade de gerar novos problemas, com tamanho e complexidade arbitrárias. O grupo optou por escolher um problema de decisão: o puzzle *C-Note*.

O artigo possui a seguinte estrutura:

- **Descrição do Problema:** descrição detalhada do problema de otimização ou decisão em análise, incluindo todas as restrições envolvidas.
- **Abordagem:** descrição da modelação do problema como um PSR, de acordo com as seguintes subsecções:
  - **Variáveis de Decisão:** descrição das variáveis de decisão e os seus respetivos domínios.
  - **Restrições:** descrição das restrições rígidas e flexíveis do problema e a sua implementação utilizando o *SICStus Prolog*.
- **Visualização da Solução:** explicação dos predicados que permitem visualizar a solução em modo de texto.
- **Experiências e Resultados:**
  - **Análise Dimensional:** demonstração de exemplos de execução em instâncias do problema com diferentes dimensões e análise os resultados obtidos.
  - **Estratégias de Pesquisa:** demonstração das diferentes estratégias de pesquisa (heurísticas de escolha de variável e de valor), comparando os resultados obtidos.
- **Conclusões e Trabalho Futuro:** conclusões retiradas deste projeto, resultados obtidos, vantagens e limitações da solução proposta e aspetos a melhorar no trabalho desenvolvido.
- **Referências**: fontes bibliográficas usadas, incluindo livros, artigos, páginas Web, entre outros.
- **Anexo:** código fonte, ficheiros de dados, resultados detalhados, entre outros.

2 Descrição do Problema

O puzzle *C-Note* desenvolvido pelo Professor de Matemática Erich Friedman, consiste numa grelha de tamanho arbitrário com o mesmo número de linhas e colunas. Inicialmente, cada célula desta grelha contém apenas um número. Para resolver o puzzle é necessário acrescentar ao número de cada célula da grelha fornecida um digito antes ou depois deste, de modo a que a soma de cada linha e de cada coluna da grelha totalize 100.

3 Abordagem

Na resolução deste problema, foi utilizada uma lista de listas para representar a grelha inicial fornecida pelo utilizador, em que cada lista representa uma linha dessa mesma grelha. Assim sendo, o input [[8,8,4],[6,2,5],[3,6,1]] representaria a seguinte grelha:

```
|----|----|----|
|  8 |  8 |  4 |
|----|----|----|
|  6 |  2 |  5 |
|----|----|----|
|  3 |  6 |  1 |
|----|----|----|
```

3.1 Variáveis de Decisão

Após a receção da grelha fornecida pelo utilizador, é inicializada uma lista de variáveis com o tamanho da grelha fornecida, ou seja, se a grelha fornecida tiver uma dimensão de 3x3, a lista irá conter 9 variáveis. Após a aplicação das restrições mencionadas no ponto seguinte e do cálculo dos valores para cada variável, essa lista terá variáveis com os valores atribuídos, constituindo a solução do problema.

3.2 Restrições

Neste problema, a soma de todas as linhas e todas as colunas terá de ser igual a 100, podendo este valor ser definido pelo utilizador. Além disso, os números fornecidos na solução têm de conter os dígitos fornecidos pelo utilizador. Assim sendo, estas são as duas grandes restrições que são impostas para calcular a solução do problema.
Na geração de problemas, apenas é aplicada a primeira restrição mencionada anteriormente, sendo depois escolhido aleatoriamente o dígito a mostrar dos números calculados.

4 Visualização da Solução

Após o programa obter a grelha de um puzzle gerado ou resolvido, existem 5 predicados que permitem visualizá-la em modo de texto, de modo a realizar uma melhor demonstração desta:

1. *print_board(+Board, +Message)*
2. *print_separation(+Counter, +RowLength)*
3. *print_matrix(+Matrix, +Index, +RowLength)*
4. *print_line(+Row)*
5. *print_element(+Element)*

O predicado principal, *print_board(+Board, +Message)*, começa por calcular o número de elementos de uma linha. De seguida, mostra a mensagem que lhe é passada, sendo esta utilizada para fazer a distinção entre uma grelha-problema e uma grelha-solução. É ainda chamado o predicado *print_separation(+Counter, +RowLength)*, que imprime o limite exterior da grelha.

```
print_board(Board, Message):-
    nth1(1,Board,Row),
    length(Row,RowLength),
    nl,
    write(Message),
    nl,
    write('|'),
    print_separation(0,RowLength),
    nl,
    print_matrix(Board,0,RowLength).

print_separation(RowLength,RowLength).
print_separation(Counter,RowLength):-
    write('----|'),
    NewCounter is Counter+1,
    print_separation(NewCounter,RowLength).
```

De seguida, para completar a impressão da matriz é chamado o predicado *print_matrix(+Matrix, +Index, +RowLength)*, que imprime, linha a linha, o número de cada célula através de *print_line(+Row) e print_element(+Element)*.

```
print_matrix([],_,_).
print_matrix([H|T],X, RowLength):-
    X1 is X+1,
    write('|'),
    print_line(H),
    nl,
    write('|'),
    print_separation(0,RowLength),
    nl,
    print_matrix(T,X1,RowLength).
```

```
print_line([]).
print_line([H|T]):-
```

```
    print_element(H),
    print_line(T).

print_element(X):-
    X < 10,
    write('  '),
    write(X),
    write(' |').

print_element(X):-
    X >= 10,
    write(' '),
    write(X),
    write(' |').
```

Quando a execução de todos estes predicados terminar, o resultado será imprimido neste formato:

```
Grid:
|----|----|----|
|  8 |  8 |  4 |
|----|----|----|
|  6 |  2 |  5 |
|----|----|----|
|  3 |  6 |  1 |
|----|----|----|


Solution:
|----|----|----|
|  8 |  8 | 84 |
|----|----|----|
| 56 | 29 | 15 |
|----|----|----|
| 36 | 63 |  1 |
|----|----|----|
```

5 Experiências e Resultados

De modo a retirar conclusões dos resultados obtidos foram medidos o tempo de resolução, o número de retrocessos e o número de restrições criadas.

5.1 Análise Dimensional

Seguem-se as condições de teste e as respetivas conclusões.

- **Fez-se variar o valor da soma das linhas e colunas, mantendo-se a dimensão da grelha/tabuleiro (Tabela 2, Figura 1, Figura 2, Figura 3 em Anexo):** O tempo de resolução do problema e o número de retrocessos variam exponencialmente com o aumento do valor da soma das linhas e colunas. Por outro lado, o número de restrições criadas varia linearmente com o aumento do valor da soma das linhas e colunas. Desta maneira, pode-se assim concluir que o tempo de resolução depende do número de retrocessos e não do número de restrições criadas, quando se aumenta a complexidade do problema ao incrementar o valor da soma das linhas e colunas.
- **Fez-se variar a dimensão da grelha/tabuleiro, mantendo-se constante a soma das linhas e colunas (Tabela 3, Figura 4, Figura 5, Figura 6 em Anexo):** O tempo de resolução do problema, o número de retrocessos e o número de restrições criadas variam exponencialmente com o aumento da dimensão da grelha/tabuleiro.

5.2 Estratégias de Pesquisa

Foram realizados diversos testes às várias opções de pesquisa para a resolução deste problema. Para ser possível obter uma conclusão com estes testes teve de ser usada uma soma constante de 700 e a mesma grelha [[8,7,8],[3,1,2],[2,7,2]], representada da seguinte forma:

```
|----|----|----|
|  8 |  7 |  8 |
|----|----|----|
|  3 |  1 |  2 |
|----|----|----|
|  2 |  7 |  2 |
|----|----|----|
```

A tabela 1 em Anexo apresenta os dados destes mesmos testes. Observando os resultados podemos então concluir que a melhor estratégia de pesquisa é a utilização das opções de *labeling* [5] ***bissect occurences***, contrastando com a pior estratégia que utiliza as opções ***middle most_constrained***.

6 Conclusões e Trabalho Futuro

Este projeto teve como principal objetivo o desenvolvimento de um programa que aplicasse os conhecimentos adquiridos nas aulas teóricas e práticas acerca de Programação em Lógica com Restrições, de modo a resolver problemas de otimização/decisão no ambiente de desenvolvimento *SICStus* em *Prolog*.

Ao longo do desenvolvimento do projeto, surgiram algumas dificuldades relativas ao uso das restrições para o nosso problema. No entanto, após uma análise extensa da biblioteca CLPFD e do material das aulas teóricas [6], estas dificuldades foram ultrapassadas com sucesso.

Relativamente aos aspetos a melhorar, poderia ter sido escolhido um método mais eficiente e otimizado de modo a acelerar a resolução de grelhas de maior dimensão.

Em suma, o projeto foi concluído com sucesso, visto que, além de solucionar corretamente o problema proposto, possui também a capacidade de gerar novos problemas. O desenvolvimento deste programa contribuiu significativamente para uma melhor compreensão do funcionamento do *labeling*, restrições e ainda variáveis de decisão.

7 Referências

[1] sigarra.up.pt. (2020). FEUP - Programação em Lógica. [online] Available at: https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=459482 [Accessed 1 Jan. 2021].
[2] sicstus.sics.se. (2020). SICStus Prolog Homepage. [online] Available at: https://sicstus.sics.se/ [Accessed 1 Jan. 2021].
[3] Friedman, E. (2011). C-Note Puzzles. [online] Available at: https://erich-friedman.github.io/puzzle/100/ [Accessed 1 Jan. 2021].
[4] sicstus.sics.se. (2020). SICStus Prolog: lib-clpfd. [online] Available at: https://sicstus.sics.se/sicstus/docs/4.1.0/html/sicstus/lib_002dclpfd.html [Accessed 1 Jan. 2021].
[5] sicstus.sics.se. (2020). SICStus Prolog: Enumeration Predicates. [online] Available at: https://sicstus.sics.se/sicstus/docs/4.1.3/html/sicstus/Enumeration-Predicates.html [Accessed 3 Jan. 2021].
[6] moodle.up.pt (2020). Moodle UC: Programação em Lógica. [online] Available at: https://moodle.up.pt/course/view.php?id=1476 [Accessed 3 Jan. 2021].

8 Anexos

8.1 Tabelas e Gráficos

Tabela 1. Teste de Estratégia de Pesquisa

	leftmost	min	max	first_fail	anti_first_fail	occurrence	most_constrained	max_regret
step	1,26	8,63	2,14	17,44	5,96	1,32	19,18	1,28
enum	2,52	13,7	3,77	37,29	2,67	2,52	41,77	2,66
bisect	1,29	9,91	9,89	10,8	32,22	1,23	11,79	1,32
middle	2,72	14,1	4,05	40,58	44,56	2,94	45,11	2,86
median	2,77	14,17	4,12	40,87	50,9	2,69	44,28	2,99

Tabela 2. Variação da duração, retrocessos e restrições criadas em função da soma das linhas e colunas

Soma	Tempo/s	Retrocessos	Restrições Criadas	Input
100	0,04	410	505	[[8,7,8],[3,1,2],[2,7,2]]
150	0,05	768	604	[[8,7,8],[3,1,2],[2,7,2]]
200	0,07	1550	781	[[8,7,8],[3,1,2],[2,7,2]]
250	0,09	3253	1179	[[8,7,8],[3,1,2],[2,7,2]]
300	0,16	5285	1343	[[8,7,8],[3,1,2],[2,7,2]]
400	0,31	10199	1744	[[8,7,8],[3,1,2],[2,7,2]]
500	0,6	17996	1863	[[8,7,8],[3,1,2],[2,7,2]]
700	1,33	32428	2129	[[8,7,8],[3,1,2],[2,7,2]]
900	2,42	52733	2423	[[8,7,8],[3,1,2],[2,7,2]]
1200	5,06	96505	3209	[[8,7,8],[3,1,2],[2,7,2]]
1500	9,32	156319	3891	[[8,7,8],[3,1,2],[2,7,2]]

Tabela 3. Variação da duração, retrocessos e restrições criadas em função da dimensão da grelha/tabuleiro

Dimensão do Tabuleiro	Tempo/s	Retrocessos	Restrições Criadas	Input
4	0,02	0	8	[[5,5],[5,5]]
9	0,04	410	505	[[8,7,8],[3,1,2],[2,7,2]]
16	0,41	40246	6695	[[3,2,8,4],[2,6,5,4],[1,6,4,4],[3,4,1,2]]
25	313,72	36084851	587692	[[9,1,1,1,1],[1,6,3,2,1],[1,2,1,7,7],[1,3,4,7,1],[1,9,1,6,3]]

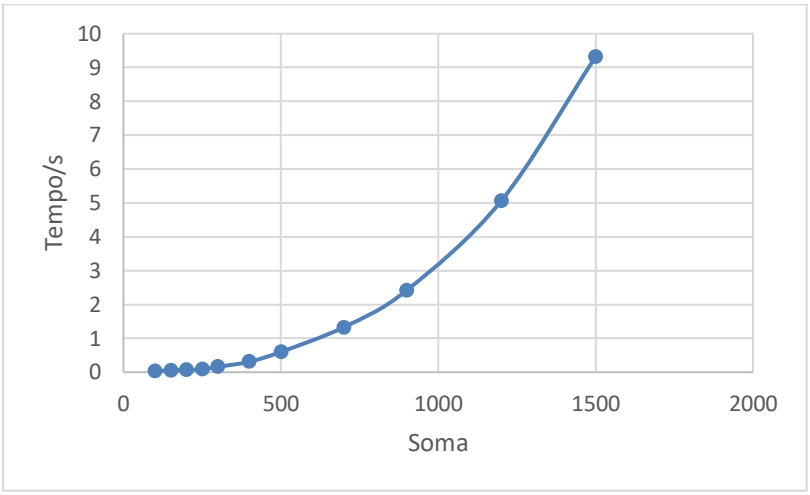

Fig. 1. Variação da duração de resolução em função da soma das linhas e colunas

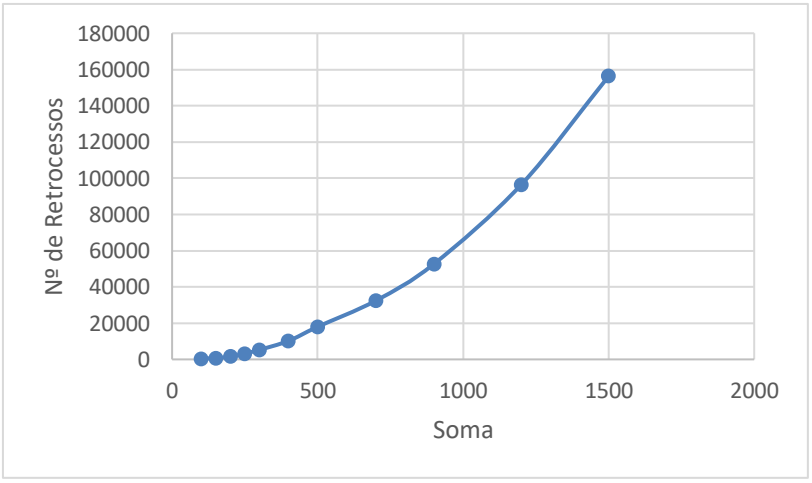

Fig. 2. Variação do número de retrocessos em função da soma das linhas e colunas

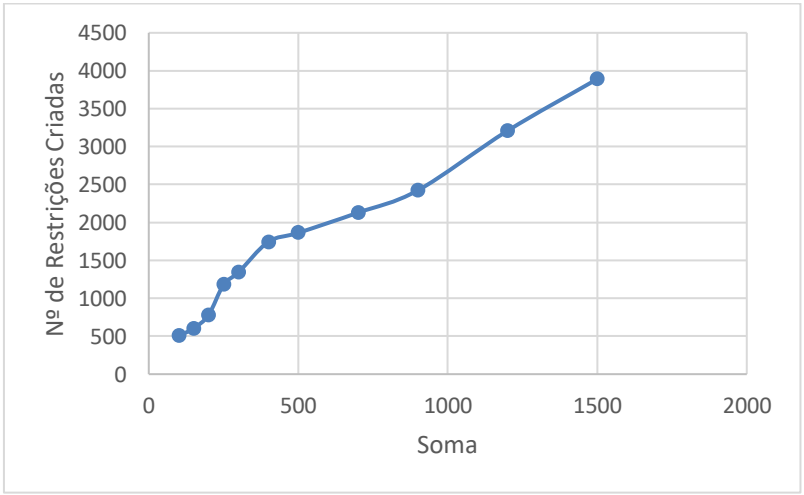

Fig. 3. Variação do número de restrições criadas em função da soma das linhas e colunas

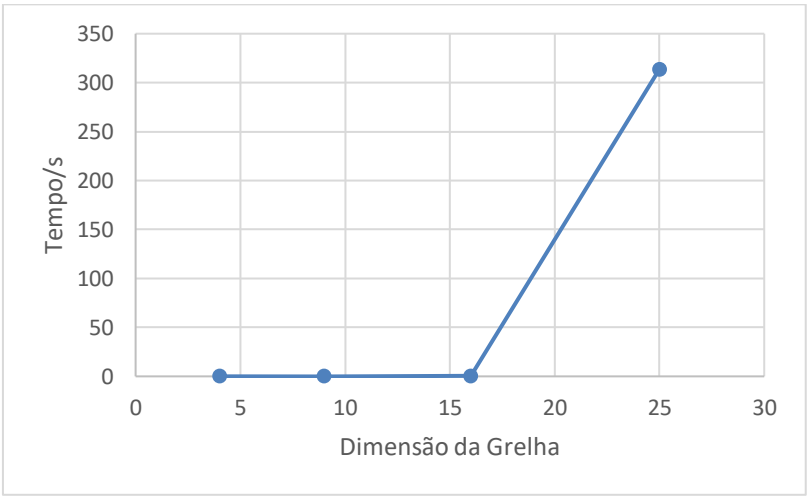

Fig. 4. Variação da duração de resolução em função da dimensão da grelha/tabuleiro

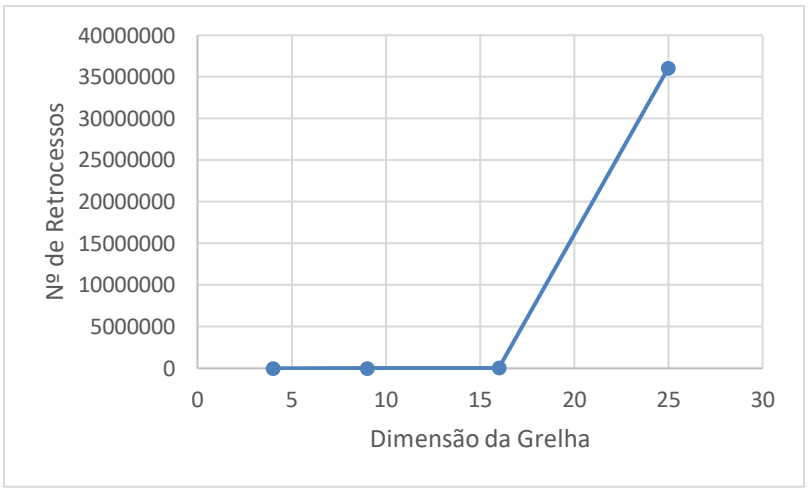

Fig. 5. Variação do número de retrocessos em função da dimensão da grelha/tabuleiro

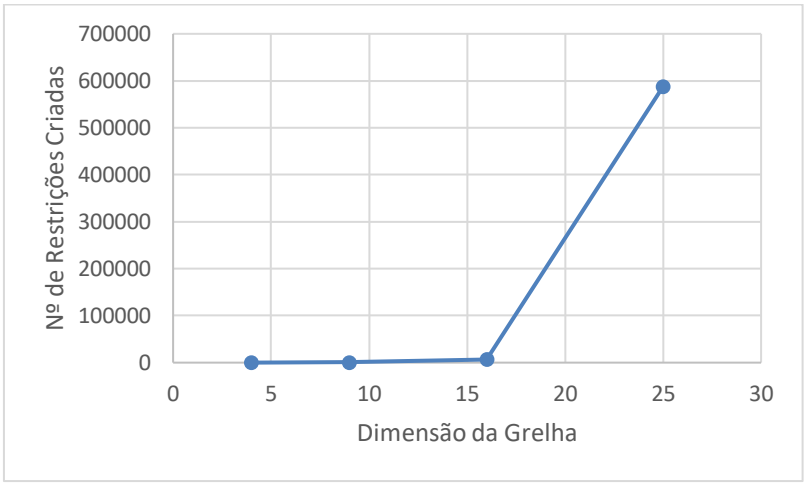

Fig. 6. Variação do número de restrições criadas em função da dimensão da grelha/tabuleiro

8.2 Código Fonte

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

%predi-
cates given in the moodle section to have time of execution
reset_timer :- statistics(walltime,_).
print_time :-
    statistics(walltime,[_,T]),
    TS is ((T//10)*10)/1000,
    nl, write('Time: '), write(TS), write('s'), nl, nl.

%predicates for the implementation of call_nth in sicstus
:- use_module(library(structs),
         [new/2,
          dispose/1,
          get_contents/3,
          put_contents/3]).

:- meta_predicate(call_nth(0, ?)).
:- meta_predicate(call_nth1(0, +, ?)).

call_nth(Goal_0, Nth) :-
   new(unsigned_32, Counter),
   call_cleanup(call_nth1(Goal_0, Counter, Nth),
           dispose(Counter)).

call_nth1(Goal_0, Counter, Nth) :-
    nonvar(Nth),
    !,
    Nth \== 0,
    \+arg(Nth,s(1),2), % produces all expected errors
    call(Goal_0),
    get_contents(Counter, contents, Count0),
    Count1 is Count0+1,
    (  Nth == Count1
    -> !
    ;  put_contents(Counter, contents, Count1),
       fail
    ).

call_nth1(Goal_0, Counter, Nth) :-
    call(Goal_0),
    get_contents(Counter, contents, Count0),
    Count1 is Count0+1,
    put_contents(Counter, contents, Count1),
```

```
    Nth = Count1.

%predicate to convert a list of lists to a flatten list
flatten([], []) :- !.
flatten([L|Ls], FlatL) :-
    !,
    flatten(L, NewL),
    flatten(Ls, NewLs),
    append(NewL, NewLs, FlatL).
flatten(L, [L]).

%predi-
cate to apply the restrictions for the sum of a line be equ
al to Sum
sumLine([],_).
sumLine([H|T],Sum):-
    sum(H,#=,Sum),
    sumLine(T,Sum).

%predi-
cate to apply the restriction of a number including a digit
 given by the user
includesDigit(Var, Number):-
    Var mod 10  #= Number.
includesDigit(Var, Number):-
    Var #> 0,
    Rest #= Var // 10,
    includesDigit(Rest, Number).

%predi-
cate to iterate the list of vars to apply the restrictions
in the includesDigit predicate
numberRestrictions([], _, _).
numberRestrictions([Var|T], InputList, Index):-
    nth1(Index, InputList, Num),
    includesDigit(Var, Num),
    NewIn-
dex is Index + 1, numberRestrictions(T, InputList, NewIndex
).

%predicates to convert a list to a list of lists
makeRow([],_,_,_,[]).

mak-
eRow([H|T],RowLength,RowLength,CurrentRow,[NewCurrentRow|Re
st]):-
    append(CurrentRow,[H],NewCurrentRow),
    makeRow(T,1,RowLength,[],Rest).
```

```
mak-
eRow([H|T],CurrentRowLength,RowLength,CurrentRow,FinalRow):
-
    append(CurrentRow,[H],NewCurrentRow),
    NewCurrentRowLength is CurrentRowLength + 1,
    mak-
eRow(T,NewCurrentRowLength,RowLength,NewCurrentRow,FinalRow
).


%predi-
cate to solve the problem given in InputList with a sum of
lines and rows equal to Sum and give the solution in Output
List
solver(InputList, Sum, OutputList):-
    flatten(InputList,FlattenList),
    nth1(1,InputList,Row),
    length(Row,RowLength),
    length(FlattenList,Length),
    length(OutputList,Length),
    domain(OutputList,1,Sum),
    makeRow(OutputList,1,RowLength,[],Rows),
    transpose(Rows,Cols),
    sumLine(Rows,Sum),
    sumLine(Cols,Sum),
    numberRestrictions(OutputList,FlattenList,1),
    labeling([],OutputList).

%predicates to print the boards for the user
print_board(Board, Message):-
    nth1(1,Board,Row),
    length(Row,RowLength),
    nl,
    write(Message),
    nl,
    write('|'),
    print_separation(0,RowLength),
    nl,
    print_matrix(Board,0,RowLength).

print_matrix([],_,_).

print_matrix([H|T],X, RowLength):-
    X1 is X+1,
    write('|'),
    print_line(H),
    nl,
```

```
    write('|'),
    print_separation(0,RowLength),
    nl,
    print_matrix(T,X1,RowLength).

%Predicates that print every row of the board
print_line([]).
print_line([H|T]):-
    print_element(H),
    print_line(T).

%Prints the separator between rows
print_separation(RowLength,RowLength).
print_separation(Counter,RowLength):-
    %nl.
    write('----|'),
    NewCounter is Counter+1,
    print_separation(NewCounter,RowLength).

%Prints an element
print_element(X):-
    X < 10,
    write('  '),
    write(X),
    write(' |').

print_element(X):-
    X >= 10,
    write(' '),
    write(X),
    write(' |').

%predicates to generate random labeling values
randomLabelingValues(Var, _Rest, BB, BB1) :-
    fd_set(Var, Set),
    select_best_value(Set, Value),
    (
        first_bound(BB, BB1), Var #= Value
        ;
        later_bound(BB, BB1), Var #\= Value
    ).

select_best_value(Set, BestValue):-
    fdset_to_list(Set, Lista),
    length(Lista, Len),
    random(0, Len, RandomIndex),
    nth0(RandomIndex, Lista, BestValue).
```

```
%predicates to choose a random element of a list
choose([], []).
choose(List, Elt) :-
    length(List, Length),
    random(0, Length, Index),
    nth0(Index, List, Elt).

%predicate to choose a random digit to be shown
chooseRandomDigit(Number, Digit):-
    number_codes(Number,List),
    choose(List,Element),
    Digit is Element-48.

%predicate to verify if a predicate succeeds more than once
more_than_once(Goal) :-
    \+ \+ call_nth(Goal,2).

%predicate to generate a random problem
generateRandomPuzzle(RowLength,Sum,OutputList):-
    BoardSize is RowLength*RowLength,
    length(Aux,BoardSize),
    domain(Aux,1,Sum),
    makeRow(Aux,1,RowLength,[],Rows),
    transpose(Rows,Cols),
    sumLine(Rows,Sum),
    sumLine(Cols,Sum),
    labeling([value(randomLabelingValues)], Aux),
    maplist(chooseRandomDigit,Aux,OutputList).

%predi-
cate to generate a random problem with unique solution
generateRandomPuzzleWithUniqueSolu-
tion(RowLength,Sum,OutputList):-
    BoardSize is RowLength*RowLength,
    length(Aux,BoardSize),
    domain(Aux,1,Sum),
    makeRow(Aux,1,RowLength,[],Rows),
    transpose(Rows,Cols),
    sumLine(Rows,Sum),
    sumLine(Cols,Sum),
    labeling([value(randomLabelingValues)], Aux),
    maplist(chooseRandomDigit,Aux,OutputList),
    once(makeRow(OutputList,1,RowLength,[],Matrix)),
    \+more_than_once(solver(Matrix,Sum,_)).
```

```
%predicate to generate a random puzzle with
a unique solution and display it to the user
cNoteGenerateUnique(RowLength,Sum, Matrix):-
    generateRandomPuzzleWithUniqueSolu-
tion(RowLength,Sum,OutputList),
    makeRow(OutputList,1,RowLength,[],Matrix),
    print_board(Matrix,'Problem generated:'),!.

%predi-
cate to generate a random puzzle and display it to the user
cNoteGenerate(RowLength,Sum, Matrix):-
    generateRandomPuzzle(RowLength,Sum,OutputList),
    makeRow(OutputList,1,RowLength,[],Matrix),
    print_board(Matrix,'Problem generated:'),!.

%predi-
cate to solve a given problem and display it to the user
cNote(InputList,Sum):-
    nth1(1,InputList,Row),
    length(Row,RowLength),
    once(print_board(InputList,'Grid:')),
    reset_timer,
    solver(InputList,Sum,OutputList),
    makeRow(OutputList,1,RowLength,[],Matrix),
    print_board(Matrix,'Solution:'),
    print_time, nl,
    fd_statistics.
```