

Exercícios Resolvidos em Prolog sobre Sistemas Baseados em Conhecimento: Regras de Produção, Extração de Conhecimento, Procura e Otimização

por **Paulo Cortez**



Unidade de Ensino

Departamento de Sistemas de Informação

Escola de Engenharia

Universidade do Minho

Guimarães, Portugal

Abril, 2018

(documento revisto em março, 2021)

Índice

1 Introdução	1
2 Exercícios sobre Prolog	3
2.1 A Árvore Genealógica da Família Pinheiro	3
2.2 Exercício sobre Listas	6
2.3 <i>Stands</i> de Automóveis	9
3 Regras de Produção	13
3.1 Gemas Preciosas (regras de produção simples)	13
3.2 Compras numa Livraria (regras de produção com incerteza)	16
4 Extração Automática de Conhecimento	19
4.1 Desempenho Escolar	19
5 Procura	39
5.1 Estradas do Norte de Portugal (procura num espaço de soluções)	39
5.2 Blocos Infantis (procura num espaço de soluções)	43
6 Otimização	47
6.1 Soma de <i>Bits</i>	47
6.2 Caixeiro Viajante	52
Referências Bibliográficas	57

1 Introdução

Este texto pedagógico consiste numa versão atualizada (escrita em abril de 2018 e revista em março de 2021) e muito reformulada de um módulo de sebenta “Exercícios Resolvidos em Prolog sobre Sistemas Baseados em Conhecimento”, cuja versão original foi escrita em 2008 e que estava orientado para um conteúdo programático que foi alterado no ano letivo de 2016/17 e que em grande parte se tornou obsoleto. Pretende-se neste texto dar um apoio atualizado à unidade curricular de **Sistemas Baseados em Conhecimento (SBC)**, do terceiro ano e segundo semestre do **Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação (MIEGSI)**, <http://miegsi.dsi.uminho.pt>). O objetivo é complementar a matéria que foi lecionada nas aulas teórico-práticas e práticas-laboratoriais com a apresentação de um conjunto de exercícios e suas resoluções em Prolog.

O texto inicia-se com exercícios gerais sobre a linguagem Prolog (Capítulo 2), seguindo-se os dois grandes temas da unidade curricular: i) sistemas baseados em conhecimento para classificação/diagnóstico/decisão via regras de produção, com aquisição de conhecimento manual (Capítulo 3) e automática (via extração de conhecimento a partir de dados (Capítulo 4); e ii) sistemas baseados em conhecimento para resolução de problemas de procura (Capítulo 5) e otimização (Capítulo 6).

De realçar que:

- todo o código apresentado neste livro foi executado no compilador gratuito SWI-Prolog (**Wielemaker et. al, 2012**), que corre em múltiplas plataformas, tais como o *Windows*, *Linux* ou mesmo *MacOS*: <http://www.swi-prolog.org>;
- o código apresentado neste livro encontra-se disponível em: <http://www3.dsi.uminho.pt/pcortez/sbc-prolog.zip>; e
- cada solução apresentada deve ser entendida como uma possível solução, exemplificativa, e não como a única forma de resolução do exercício proposto.

2 Exercícios sobre Prolog

2.1 A Árvore Genealógica da Família Pinheiro

Enunciado:

Pouco se sabe da história passada da família **Pinheiro**. Existem alguns registos antigos que indicam que o casal José e Maria criou dois filhos, o João e a Ana. Que a Ana teve duas filhas, a Helena e a Joana, também parece ser verdade, segundo os mesmos registos. Além disso, o Mário é filho do João, pois muito se orgulha ele disso. Estranho também, foi constatar que o Carlos nasceu da relação entre a Helena, muito formosa, e o Mário.

- Utilizando o predicado **progenitor(X,Y)** (ou seja, X é progenitor de Y), represente em Prolog todos os progenitores da família Pinheiro.
- Represente em Prolog as relações: **sexo** (masculino ou feminino), **irmã**, **irmão**, **descendente**, **mãe**, **pai**, **avô**, **tio**, **primo**¹.
- Formule em Prolog as seguintes questões:
 - O João é filho do José?
 - Quem são os filhos da Maria?
 - Quem são os primos do Mário?
 - Quantos sobrinhos/sobrinhas com um Tio existem na família Pinheiro?
 - Quem são os ascendentes do Carlos?
 - A Helena tem irmãos? E irmãs?

Código da Resolução:

```
pinheiro.pl
% factos
progenitor(maria,joao).
progenitor(jose,joao).
progenitor(maria,ana).
progenitor(jose,ana).
progenitor(joao,mario).
progenitor(ana,helena).
progenitor(ana,joana).
progenitor(helena,carlos).
progenitor(mario,carlos).

sexo(ana,feminino).
sexo(maria,feminino).
sexo(joana,feminino).
sexo(helena,feminino).

sexo(mario,masculino).
sexo(joao,masculino).
sexo(jose,masculino).
sexo(carlos,masculino).
```

¹ Neste caso, por *primo* entende-se primo ou prima.

```

irma(X,Y):- progenitor(A,X),
            progenitor(A,Y),
            X\==Y,
            sexo(X,feminino).

irmao(X,Y):- progenitor(A,X),
            progenitor(A,Y),
            X\==Y,
            sexo(X,masculino).

descendente(X,Y):- progenitor(X,Y).
descendente(X,Y):- progenitor(X,A),
                    descendente(A,Y).

avo(X,Y):- progenitor(X,A),
            progenitor(A,Y),
            sexo(X,masculino).

mae(X,Y):- progenitor(X,Y),
            sexo(X,feminino).

pai(X,Y):- progenitor(X,Y),
            sexo(X,masculino).

tio(X,Y):- irmao(X,A),
            progenitor(A,Y).

primo(X,Y):-irmao(A,B),
            progenitor(A,X),
            progenitor(B,Y),
            X\==Y.
primo(X,Y):-irma(A,B),
            progenitor(A,X),
            progenitor(B,Y),
            X\==Y.

% questoes:
q1:- progenitor(jose,joao).
q1b:- pai(jose,joao).

q2(X):- mae(maria,X).
q2b(L):-findall(X,mae(maria,X),L).

q3(X):- primo(mario,X).
q3b(L):- findall(X,primo(mario,X),L).
q3c(L):- findall(X,primo(mario,X),LR),list_to_set(LR,L).

q4(X):- tio(_,X).
q4b(L):- findall(X,tio(_,X),LR),list_to_set(LR,L).

q5(X):- descendente(X,carlos).
q5b(L):- findall(X,descendente(X,carlos),L).

q6a(X):- irmao(helena,X).
q6b(X):- irma(helena,X).

```

Explicação da Resolução:

Este exercício envolve **objetos e relações entre objetos**, sendo uma adaptação livre do programa *family* do livro (Brakto, 2012). Dado que o enunciado é livre neste aspeto,

optou-se por utilizar a notação **sexo(Nome, Sexo)** para representar o sexo de cada pessoa. Em algumas das relações pode existir mais do que uma forma de resolver aquilo que é pedido. As questões da alínea c) podem ser ter diferentes interpretações (por exemplo se a questão deve retornar uma ou todas as soluções), sendo que nestes casos, optou-se por apresentar as diversas alternativas (e.g., **q2a**, **q2b**). Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```
?- [pinheiro].
```

```
true.
```

```
?- q1.
```

```
true.
```

```
?- q2b(x).
```

```
x = [joao, ana].
```

(executar as restantes questões q2, q3, q3b, ...).

2.2 Exercício sobre Listas

Enunciado:

Represente em **Prolog** os seguintes predicados genéricos sobre listas (sem utilizar os correspondentes predicados do módulo `lists` do SWI-Prolog):

- 1) **adiciona(X,L1,L2)** – onde L2 é a lista que contém o elemento X e a lista L1. Testar este predicado no interpretador Prolog, executando:
?- adiciona(1,[2,3],L).
?- adiciona(X,[2,3],[1,2,3]).
- 2) **apaga(X,L1,L2)** – onde L2 é a lista L1 sem o elemento X. Testar com:
?- apaga(a,[a,b,a,c],L).
?- apaga(a,L,[b,c]).
- 3) **membro(X,L)** – que é verdadeiro se X pertencer à lista L. Testar com:
?- membro(b,[a,b,c]).
?- membro(X,[a,b,c]). % carregar em ;
?- findall(X,membro(X,[a,b,c]),L).
- 4) **concatena(L1,L2,L3)** – onde L3 é resultado da junção das listas L2 e L1. Testar com:
?- concatena([1,2],[3,4],L).
?- concatena([1,2],L,[1,2,3,4]).
?- concatena(L,[3,4],[1,2,3,4]).
- 5) **comprimento(X,L)** – onde X é o número de elementos da lista L. Testar com:
?- comprimento(X,[a,b,c]).
- 6) **maximo(X,L)** – onde X é o valor máximo da lista L (assumir que L contém somente números). Testar com:
?- maximo(X,[3,2,1,7,4]).
- 7) **media(X,L)** – onde X é o valor médio da lista L (assumir que L contém somente números). Testar com:
?- media(X,[1,2,3,4,5]).
- 8) **nelem(N,L,X)** – onde N é um número e X é o elemento da lista L na posição L. Por exemplo (testar com):
?- nelem(2,[1,2,3],2).
?- nelem(3,[1,2,3],X).
?- nelem(4,[a,b,c,d,e,f,g],X).

Código da Resolução:

listas.pl

```
% 1
adiciona(X,L,[X|L]).

% 2
apaga(X,[X|R],R).
apaga(X,[Y|R1],[Y|R2]):-
    apaga(X,R1,R2).

% 3
membro(X,[X|_] ).
membro(X,[_|R] ) :- membro(X,R ).
```

```

% 4
concatena([],L,L).
concatena([X|L1],L2,[X|L3]):- concatena(L1,L2,L3).

% 5
comprimento(0,[]).
comprimento(N,[_|R]):- comprimento(N1,R),
                        N is 1 + N1.

% 6
max(X,[X]).
max(X,[Y|R]):- max(X,R), X > Y, !.
max(Y,[Y|_]).

% 7
somatorio(0,[]).
somatorio(X,[Y|R]):- somatorio(S,R),
                      X is S+Y.

media(X,L):- comprimento(N,L),
              somatorio(S,L),
              X is S/N.

nelem(N,L,X):-nelem(N,1,L,X).
nelem(N,N,[X|_],X):-!.
nelem(N,I,[_|R],X):- I1 is I+1,
                      nelem(N,I1,R,X).

% testar os predicados:
q1a(L):-adiciona(1,[2,3],L).
q1b(X):-adiciona(X,[2,3],[1,2,3]).

q2a(L):-apaga(a,[a,b,a,c],L).
q2b(L):-apaga(a,L,[b,c]).

q3a:-membro(b,[a,b,c]).
q3b(X):-membro(X,[a,b,c]).
q3c(L):-findall(X,membro(X,[a,b,c]),L).

q4a(L):-concatena([1,2],[3,4],L).
q4b(L):-concatena([1,2],L,[1,2,3,4]).
q4c(L):-concatena(L,[3,4],[1,2,3,4]).

q5(X):-comprimento(X,[a,b,c]).

q6(X):-max(X,[3,2,1,7,4]).

q7(X):-media(X,[1,2,3,4,5]).

q8:-nelem(2,[1,2,3],2).
q8b(X):-nelem(3,[1,2,3],X).
q8c(X):-nelem(4,[a,b,c,d,e,f,g],X).

```

Comentário sobre a Resolução:

Este exercício serve para praticar a manipulação de **listas**, sendo uma adaptação livre do código apresentado no livro (Brakto, 2012). A maior parte destes predicados já se encontra definido no SWI-Prolog em inglês no módulo **lists**. Por exemplo: membro -

member, adiciona - **append**, apaga - **delete**, máximo - **max_list** e nelem - **nth1**. Uma lista mais extensa de predicados encontra-se disponível no manual da ferramenta SWI-Prolog (**Wielemaker et al., 2014**). De notar que a maioria dos predicados que manipulam listas utilizam o mecanismo de recursividade, de modo a poder-se *navegar* ao longo de uma lista. Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```
?- [listas].
```

```
true.
```

```
?- q1a(L).
```

```
L = [1, 2, 3].
```

(executar as restantes questões q1b, q2a, q2b, ...).

2.3 Stands de Automóveis

Enunciado:

Considere a seguinte BD sobre clientes de *stands* de automóveis:

Stand	Nome	Nº Cliente	Idade	Profissão	Compras
Vegas	Rui	2324	23	Médico	Carro Audi A2 por 20000 euros Carro BMW Serie3 por 30000 euros
Vegas	Rita	2325	32	Advogado	Carro Audi A3 por 30000 euros
Vegas	João	2326	26	Professor	Moto Honda GL1800 por 26000 eur.
Vegas	Ana	2327	49	Médico	Carro Audi A4 por 40000 euros Carro BMW Serie3 por 32000 euros Carro Ford Focus por 24000 euros
Miami	Rui	3333	33	Operário	Carro Fiat Panda por 12000 euros
Miami	Paulo	3334	22	Advogado	Carro Audi A4 por 36000 euros
Miami	Pedro	3335	46	Advogado	Carro Honda Accord por 32000 eur. Carro Audi A2 por 20000 euros

1) Registe em Prolog todos os dados relevantes da BD, utilizando factos com a notação: **stand(nome_stand, LC)**.

onde **LC** é uma lista de clientes do tipo:

[**cliente(nome, num, id, prof, C1)**, **cliente(nome2, num2, id2, prof2, C2)**, ...]

onde **C1**, **C2** são listas de compras do tipo:

[**carro(marca1, modelo1, preco1)**, **moto(marca2, modelo2, preco2)**, ...]

2) Defina em Prolog os seguintes predicados:

- 1) **listar_clientes(X, LC)** – devolve a lista LC com o nome de todos clientes do stand X;
- 2) **listar_dados(X, C, D)** – devolve a lista D com todos dados (i.e.: numero, idade e profissão) do cliente com o nome C do stand X;
- 3) **listar_carros(X, LM)** – devolve a lista LM com o nome de todas as marcas de carros vendidos pelo stand X.
- 4) **listar_advogados(LA)**:- devolve a lista LA com o nome de todos os advogados de todos os stands;
- 5) **preco_medio(X, Med)** - devolve o preço médio (Med) de todos os carros vendidos por um stand. Nota: pode re-utilizar o predicado **media(X, L)** do exercício anterior;
- 6) **altera_id(X, C, Id)** – altera a idade do cliente C do stand X para Id. Nota: deve usar os predicados do Prolog **assert** e **retract**.

Utilize os seguintes predicados SWI-Prolog:

flatten(L1, L2) – remove todos os [] extra de L1, devolvendo o resultado em L2;

list_to_set(L1, L2) – remove elementos repetidos de L1, devolvendo L2;

Por exemplo, **flatten([[1],[2,3]],[1,2,3])** e

list_to_set([1,2,2,3],[1,2,3]) dão verdade.

Código da Resolução:

stand.pl

```
:- dynamic(stand/2),
   consult('listas.pl'). % acesso a media de uma lista

% 1: representacao da base de dados
stand(vegas,[
    cliente(rui,2324,23,medico,[ carro(audi,a2,20000),
                                carro(bmw,serie3,30000)]),
    cliente(rita,2325,32,advogado,[carro(audi,a3,30000)]),
    cliente(joao,2326,26,professor,[moto(honda,g11800,26000)]),
    cliente(ana,2327,49,medico,[
        carro(audi,a4,40000),
        carro(bmw,serie3,32000),
        carro(ford,focus,24000)])
]).

stand(miami,[
    cliente(rui,3333,33,operario,[carro(fiat,panda,12000)]),
    cliente(paulo,3334,22,advogado,[carro(audi,a4,36000)]),
    cliente(pedro,3335,46,advogado,[carro(honda,accord,32000),
                                    carro(audi,a2,20000)])
]).

% 2.1: devolve a lista com o nome de todos os clientes de um stand
listar_clientes(X,LC):-
    stand(X,L),
    findall(C,member(cliente(C,_,_,_,_),L),LC).

% 2.2: devolve os dados de cliente (todos excepto o nome):
listar_dados(X,C,D):-
    stand(X,L),
    findall((N,ID,P),member(cliente(C,N,ID,P,_) ,L),D).

% 2.3:
listar_carros(X,LM):-
    stand(X,L),
    findall(C,member(cliente(_,_,_,_,C),L),LC),
    flatten(LC,LCC),
    findall(M,member(carro(M,_,_) ,LCC),LM1),
    list_to_set(LM1,LM).

% 2.4:
listar_advogados(LA):-
    findall(L,stand(_,L),LL),
    flatten(LL,LL2),
    findall(C,member(cliente(C,_,_,advogado,_) ,LL2),LA1),
    list_to_set(LA1,LA).

% 2.5:
preco_medio(X,Med):-
    stand(X,L),
    findall(C,member(cliente(_,_,_,_,C),L),LP),
    flatten(LP,LP2),
    findall(P,member(carro(_,_,P),LP2),LP3),
    media(Med,LP3).

% 2.6:
```

```

altera_id(X,C,Id):-
    retract(stand(X,L)),
    altera_id(L,L2,C,Id),
    assert(stand(X,L2)).

% predicado auxiliar:
altera_id(L,L2,C,NID):- select(cliente(C,N,_,P,V),L,L1),
    append([cliente(C,N,NID,P,V)],L1,L2).

% exemplo de um teste deste programa:
teste:- write('mudar idade da ana\nde:'),
    listar_dados(vegas,ana,D),write(D),
    altera_id(vegas,ana,50),listar_dados(vegas,ana,D1),
    write(' para: '),write(D1).

```

Comentário sobre a Resolução:

Pretende-se aqui praticar a representação de bases de dados em Prolog. Neste caso, o enunciado já explicita qual o formato da representação. A resolução é conseguida à custa dos (poderosos) predicados **findall** e **member**. O facto **stand** tem de ser definido como dinâmico, uma vez que é manipulado via predicados **assert** e **retract**. Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```

?- [stand].
true.
?- teste.
mudar idade da ana
de:[ (2327, 49, medico)] para: [ (2327, 50, medico)]

```


3 Regras de Produção

3.1 Gemas Preciosas (regras de produção simples)

Enunciado:

Existem diversos tipos de gemas preciosas. Para simplificar, somente serão classificadas um conjunto reduzido de gemas, segundo as regras:

- O berilo é caracterizado pelo facto de ser duro e também por ser um mineral;
 - O berilo é uma pedra preciosa, sendo que uma qualquer outra gema que contenha óxido de alumínio também é uma preciosa;
 - Uma esmeralda é uma gema preciosa com um tom verde;
 - Se uma gema for preciosa e tiver cor avermelhada, então é do tipo rubi;
 - Simplificando, podemos admitir que uma safira é uma gema que é preciosa cuja tonalidade não é verde nem avermelhada.
1. Represente este conhecimento através de regras de produção em Prolog.
 2. Admita o seguinte cenário: tem um mineral que contém óxido de alumínio, cuja cor não é verde nem vermelha. Represente esta informação via factos.
 3. Utilize os sistemas de inferência de *backward* (sem e com explicação) e *forward chaining* para classificar este mineral.

Código da Resolução:

gemas.pl

```
% Executado automaticamente quando o Prolog executa este ficheiro:
:-dynamic(fact/1), % definir fact como dinamico
 [backward,forward,proof]. % carregar todos sistemas de inferência

% Base de Conhecimento, alinea 1
if mineral and duro then berilo.
if berilo or oxido_aluminio then precioso.
if precioso and verde then esmeralda.
if precioso and vermelho then rubi.
if oxido_aluminio and nao_verde_vermelho then safira.

% Base de Dados (os factos actuais), alinea 2
fact(mineral).
fact(oxido_aluminio).
% nota: neste caso também se poderia usar: fact(nao_verde_vermelho).
% contudo, a seguinte definição é mais genérica, sendo que
% funciona bem quando por exemplo for fact(verde).
fact(nao_verde_vermelho):- \+ fact(vermelho), \+ fact(verde).

% Classificar o mineral via backward chaining:
backward:- demo(safira). % testa se demo de safira e' verdade
% Classificar o mineral via backward chaining mas com explicacao:
```

```

proof(P):- demo(safira,P).
% Classificar o mineral via forward chaining:
forward:- demo. % gera todos os factos que pode provar

```

backward.pl

```

% backward chaining; adapted from (Brakto, 2012).
:- op( 800, fx, if).
:- op( 700, xfx, then).
:- op( 300, xfy, or).
:- op( 200, xfy, and).

demo( Q) :-
    fact( Q).

demo( Q) :-
    if Condition then Q, % A relevant rule
    demo( Condition). % whose condition is true

demo( Q1 and Q2) :-
    demo( Q1),
    demo( Q2).

demo( Q1 or Q2) :-
    demo( Q1)
    ;
    demo( Q2).

```

forward.pl

```

% forward chaining; adapted from (Brakto, 2012).
demo:-
    new_derived_fact( P),!,% A new fact
    write( 'Derived: '), write( P), nl,
    assert( fact( P)),
    demo. % Continue
demo:- write( 'No more facts'). % All facts derived

new_derived_fact( Concl) :-
    if Cond then Concl, % A rule
    \+ fact( Concl), % Rule's conclusion not yet a fact
    composed_fact( Cond). % Condition true?

composed_fact( Cond) :-
    fact( Cond). % Simple fact

composed_fact( Cond1 and Cond2) :-
    composed_fact( Cond1),
    composed_fact( Cond2). % Both conditions true

composed_fact( Cond1 or Cond2):-
    composed_fact( Cond1);
    composed_fact( Cond2).

```

proof.pl

```

% backward chaining with proof; adapted from (Brakto, 2012).
% demo( P, Proof) Proof is a proof that P is true

```

```

:- op( 800, xfx, <=).

demo( P, P) :-
    fact( P).

demo( P, P <= CondProof) :-
    if Cond then P,
    demo( Cond, CondProof).

demo( P1 and P2, Proof1 and Proof2) :-
    demo( P1, Proof1),
    demo( P2, Proof2).

demo( P1 or P2, Proof) :-
    demo( P1, Proof);
    demo( P2, Proof).

```

Comentário sobre a Resolução:

Pretende-se praticar o uso simples de regras de produção e sistemas de inferência de *backward* (backward.pl, proof.pl) e *forward* (forward.pl) *chaining* lecionados na unidade curricular. É utilizada a negação simples² do SWI-Prolog (operador \+). Mais detalhes são apresentados nos comentários do código. Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```

?- [gemas].
true.
?- backward.
true
?- proof(P).
P = (safira<=oxido_aluminio and nao_verde_vermelho)
?- forward.
Derived: precioso
Derived: safira
No more facts
true.

```

² \+P é *verdadeiro* se não for possível provar P, caso contrario é *falso*.

3.2 Compras numa Livraria (regras de produção com incerteza)

Enunciado:

Com o propósito de construir um SBC que **classifique qual tipo de livro** que irá ser comprado numa Livraria, realizou-se um inquérito sobre consumidores de livros, do qual se tiraram as seguintes conclusões:

- Com uma certeza de 75% vende-se um **livro tecnológico**, desde que o consumidor tenha um computador portátil e seja do sexo masculino;
- As mulheres sem filhos compraram **livros românticos** em 100% dos casos;
- Durante o período de natal, qualquer **livro do tipo policial** com o título “código” ou “vinci” vende-se com 80% de probabilidade;
- Os livros de **poemas** eram adquiridos com uma probabilidade de 20%, desde que fosse a época natalícia ou o comprador fosse uma mulher.
- Os livros românticos ou de poemas podem ser considerados **literários** em 90% dos casos.

- a) **Represente** em Prolog a **base de conhecimento** utilizando o regras de produção com incerteza.
- b) Considere a situação atual: “**Estamos na época natalícia e alguém, que tem um computador portátil e não tem filhos, entrou na Livraria**”. **Represente** em Prolog esta situação.
- c) Indique como poderia saber qual é a probabilidade do consumidor da alínea b) comprar um **livro literário**?

Código da Resolução:

livraria.pl

```
% Executado automaticamente quando o Prolog executa este ficheiro:
:-dynamic(fact/1), % definir fact como dinamico
[certainty]. % carregar todos sistemas de inferencia
```

```
% Base de Conhecimento, alínea a)
if portatil and homem then tecnologico:0.75.
if mulher and sem_filhos then romantico:1.0.
if natal and codigo or vinci then policial:0.8.
if natal or mulher then poemas:0.20.
if romantico or poemas then literario:0.9.
```

```
% Base de Dados (os factos atuais), alínea b)
fact(natal:1).
fact(mulher:0.5).
fact(homem:0.5).
fact(portatil:1.0).
fact(sem_filhos:1.0).
```

certainty.pl

```
% backward with certainties; adapted from (Brakto, 2012).
:- op( 800, fx, if).
:- op( 700, xfx, then).
:- op( 300, xfy, or).
:- op( 200, xfy, and).
```

```

% Rule interpreter with certainties

% democ( Proposition, Certainty)

democ( P, Cert) :-
    fact( P: Cert).

democ( Cond1 and Cond2, Cert) :-
    democ( Cond1, Cert1),
    democ( Cond2, Cert2),
    Cert is min( Cert1, Cert2).

democ( Cond1 or Cond2, Cert) :-
    democ( Cond1, Cert1),
    democ( Cond2, Cert2),
    Cert is max( Cert1, Cert2).

democ( P, Cert) :-
    if Cond then P : C1,
    democ( Cond, C2),
    Cert is C1 * C2.

```

Comentário sobre a Resolução:

Pretende-se praticar a representação de regras de produção com incerteza e respetivo sistema de inferência (`certainty.pl`) lecionado na unidade curricular. Como não se sabe qual o sexo atual da pessoa, assume-se o mais provável, ou seja, 50% de probabilidade para cada tipo de sexo. Mais detalhes são apresentados nos comentários do código. Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```

?- [livraria].
?- democ(literario,C). % alinea c)
C = 0.45

```


4 Extração Automática de Conhecimento

4.1 Desempenho Escolar

Enunciado:

Considere o conjunto de dados *Student Performance* (<http://archive.ics.uci.edu/ml/datasets/Student+Performance>), que contém diversos atributos socioeconómicos e escolares que caracterizam alunos do ensino secundário de Portugal. Em particular, considere o conjunto de dados com avaliações sobre a disciplina da Matemática.

Utilizando uma abordagem de extração automática de conhecimento:

- descubra que regras consegue extrair via um algoritmo de *machine learning* de cobertura em termos de aprovação e reprovação à disciplina de Matemática;
- implemente as regras obtidas anteriormente num sistema baseado em conhecimento recorrendo a regras de produção;
- obtenha uma árvore de decisão que seja automaticamente extraída a partir do mesmo conjunto de dados.

Código da Resolução da alínea a):

preprocess.R (linguagem R)

```
# descarregar o conjunto de dados student performance (zip file)
# do repositório UCI Machine Learning
URL="http://archive.ics.uci.edu/ml/machine-learning-
databases/00320/student.zip"
temp=tempfile() # ficheiro temporário
download.file(URL,temp) # download file to temporary

# função auxiliar que converte um data.frame com factors para
ficheiro em formato Prolog
saveprolog=function(filename,d)
{
  sink(filename) # abrir ficheiro
  # gravar factos do tipo attribute:
  n=names(d)
  n=tolower(n) # prolog compatible
  for(i in 1:length(n))
  {
    l=levels(d[,i])
    str2=paste(l,sep=" ",collapse=" ")
    str=paste("attribute(",n[i]," ", [ " ,str2, "]).\n",sep="")
    cat(str) # fact attribute
  }
  # gravar factos do tipo data:
```

```

for(i in 1:nrow(d))
{
  cat("data(["
  for(j in 1:ncol(d))
    { cat(as.character(d[i,j]))
      if(j<ncol(d)) cat(", ")
    }
  cat(")].\n")
}
sink() # fechar ficheiro
}

# descompactar o ficheiro zip para um data.frame:
d=read.table(unz(temp,"student-mat.csv"),sep=";",header=TRUE,
stringsAsFactors=TRUE)

# seleccionar atributos de entrada e saida (demo):
inputs=1:30 # todos excepto as avaliacoes do primeiro e segundo
periodos G1 e G2
output=33 # avaliacao final, do terceiro periodo G3
d=d[,c(inputs,output)]

# alterar nomes dos niveis para minusculas (devido ao Prolog):
levels(d[, "school"])=list(gp="GP",ms="MS")
levels(d[, "sex"])=list(f="F",m="M")
levels(d[, "address"])=list(r="R",u="U")
levels(d[, "famsize"])=list(gt3="GT3",le3="LE3")
levels(d[, "Pstatus"])=list(a="A",t="T")

# conversao numerico para categorico
## entradas:
d[, "age"]=cut(d[, "age"],c(14,15,16,17,18,23),
              c("a15","a16","a17","a18","maior18"))
d[, "Medu"]=cut(d[, "Medu"],c(-1,0,1,2,3,4),
              c("e0","e1","e2","e3","e4"))
d[, "Fedu"]=cut(d[, "Fedu"],c(-1,0,1,2,3,4),
              c("e0","e1","e2","e3","e4"))
d[, "traveltime"]=cut(d[, "traveltime"],c(0,1,2,3,4),
                    c("t1","t2","t3","t4"))
d[, "studytime"]=cut(d[, "studytime"],c(0,1,2,3,4),
                    c("t1","t2","t3","t4"))
d[, "failures"]=cut(d[, "failures"],c(-1,0,1,2,3),
                  c("f0","f1","f2","f3"))
d[, "famrel"]=cut(d[, "famrel"],c(0,1,2,3,4,5),
                c("fr1","fr2","fr3","fr4","fr5"))
d[, "freetime"]=cut(d[, "freetime"],c(0,1,2,3,4,5),
                  c("ft1","ft2","ft3","ft4","ft5"))
d[, "goout"]=cut(d[, "goout"],c(0,1,2,3,4,5),
                c("g1","g2","g3","g4","g5"))
d[, "Dalc"]=cut(d[, "Dalc"],c(0,1,2,3,4,5),
               c("da1","da2","da3","da4","da5"))
d[, "Walc"]=cut(d[, "Walc"],c(0,1,2,3,4,5),
               c("wa1","wa2","wa3","wa4","wa5"))
d[, "health"]=cut(d[, "health"],c(0,1,2,3,4,5),
                 c("h1","h2","h3","h4","h5"))
d[, "absences"]=cut(d[, "absences"],c(-1,0,4,10,20,76),
                   c("a0","a1","a2","a3","a4"))
## saida: reprovado/aprovado
d[, "G3"]=cut(d[, "G3"],c(-1,9,20),c("fail","pass"))

# mostrar um sumario dos dados preprocessados:
print(summary(d))

```



```
# gravar para ficheiro csv:
write.table(d,file="math.csv",sep=",",row.names=FALSE,col.names=TRUE)

# gravar para formato Prolog:
saveprolog("math.pl",d)
```

math1.pl

```
:-[preprocess], % consult preprocess
   dynamic(example/2),dynamic(data/1), % set dynamic factors
   [math]. % consult math

% lets preprocess the data:
% this creates a new math2.pl file with the attributes and
examples
createfile:-
    preprocess, % build all examples
    tell('math2.pl'),
        listing(attribute/2),
        listing(example/2),
    told.

:- createfile. % create the math2.pl file
```

preprocess.pl

```
% Paulo Cortez 2017@
% transforms simpler data facts into the required format
% of Ivan Bratko: example/2.
preprocess:-
    retract(data(L)),
    buildExample(L,Output,Inputs),
    assert(example(Output,Inputs)),
    preprocess.
preprocess:- !. % end

% build example from list L:
buildExample(L,Inputs,Output):-
    findall(A,attribute(A,_),LA),
    buildExample(LA,1,L,Inputs,Output).

buildExample(LA,I,[X,Output],Output,[X1]):-
    buildInput(LA,I,X,X1).
buildExample(LA,I,[X|R],Output,[X1|R1]):-
    buildInput(LA,I,X,X1),
    I1 is I+1,
    buildExample(LA,I1,R,Output,R1).

% build input from I-th attribute and value X:
buildInput(LA,I,X,A=X):- nth1(I,LA,A).
```

math_rules.pl

```
:-[satisfy,induce_ifthen,math2].

% the goal is to classify playTennis
learn_rules:-
    learn(fail), % negative examples
    learn(pass), % positive examples
    % save new rules:
    tell('math_ifthen.pl'),
```

```
listing(<==),
told.
```

satisfy.pl

```
% satisfy rule used for extraction rules and decision trees
% adapted from (Bratko, 2012):
```

```
satisfy( Object, Conj) :-
  \+ ( member( Att = Val, Conj),
      member( Att = ValX, Object),
      ValX \== Val).
```

induce_ifthen.pl

```
% Induction of if-then rules by a Covering algorithm;
% adapted from (Bratko, 2012)
% read the book (Bratko, 2012) for the algorithm details
:- op( 300, xfx, <==).
```

```
% learn( Class): collect learning examples into a list, construct and
% output a description for Class, and assert the corresponding
% rule about Class
```

```
learn( Class) :-
  bagof( example( ClassX, Obj), example( ClassX, Obj), Examples),
  learn( Examples, Class, Description),
  nl, write( Class), write( ' <== '), nl,
  writelist( Description),
  assert( Class <== Description).
```

```
% learn( Examples, Class, Description):
%   Description covers exactly the examples of class Class
%   in list Examples
```

```
learn( Examples, Class, []) :-
  \+ member( example( Class, _ ), Examples).
```

```
learn( Examples, Class, [Conj | Conjs]) :-
  learn_conj( Examples, Class, Conj),
  remove( Examples, Conj, RestExamples),
  learn( RestExamples, Class, Conjs).
```

```
% learn_conj( Examples, Class, Conj):
%   Conj is a list of attribute values satisfied by some examples
%   of class Class and
%   no other class
```

```
learn_conj( Examples, Class, []) :-
  \+ ( member( example( ClassX, _ ), Examples),
      ClassX \== Class), !.
```

```
learn_conj( Examples, Class, [Cond | Conds]) :-
  choose_cond( Examples, Class, Cond),
  filter( Examples, [ Cond], Examples1),
  learn_conj( Examples1, Class, Conds).
```

```
choose_cond( Examples, Class, AttVal) :-
  findall( AV/Score, score( Examples, Class, AV, Score), AVs),
  best( AVs, AttVal).
```

```
best( [ AttVal/_], AttVal).
```

```
best( [ AV0/S0, AV1/S1 | AVSlist], AttVal) :-
  S1 > S0, !,
```

```

best( [AV1/S1 | AVSlist], AttVal)
;
best( [AV0/S0 | AVSlist], AttVal).

% filter( Examples, Condition, Examples1):
%   Examples1 contains elements of Examples that satisfy Condition
filter( Examples, Cond, Examples1) :-
    findall( example( Class, Obj),
             ( member( example( Class, Obj), Examples), satisfy(
Obj, Cond))),
           Examples1).

% remove( Examples, Conj, Examples1):
%   removing from Examples those examples that are covered by
%   Conj gives Examples1
remove( [], _, []).

%remove( [example( Class, Obj) | Es], Conj, Es1) :-
remove( [example( _, Obj) | Es], Conj, Es1) :-
    satisfy( Obj, Conj), !,
    remove( Es, Conj, Es1).
remove( [E | Es], Conj, [E | Es1]) :-
    remove( Es, Conj, Es1).

score( Examples, Class, AttVal, Score) :-
    candidate( Examples, Class, AttVal),
    filter( Examples, [ AttVal], Examples1),
    length( Examples1, N1),
    count_pos( Examples1, Class, NPos1),
    NPos1 > 0,
    Score is (NPos1 + 1) / (N1 + 2).

candidate( Examples, Class, Att = Val) :-
    attribute( Att, Values),
    member( Val, Values),
    suitable( Att = Val, Examples, Class).

suitable( AttVal, Examples, Class) :-
    % At least one negative example must not match AttVal
    member( example( ClassX, ObjX), Examples),
    ClassX \== Class,
    \+ satisfy( ObjX, [ AttVal]), !.

% count_pos( Examples, Class, N):
%   N is the number of positive examples of Class
count_pos( [], _, 0).
count_pos( [example( ClassX, _ ) | Examples], Class, N) :-
    count_pos( Examples, Class, N1),
    ( ClassX = Class, !, N is N1 + 1; N = N1).

writelist( []).
writelist( [X | L]) :-
    tab( 2), write( X), nl,
    writelist( L).

% classify an object (get the class):
classify(Object,Class):-
    Class <== Description,
    member(Conj, Description),
    satisfy(Object, Conj).

```

Comentário sobre a Resolução:

O código fornecido no livro do Ivan Brakto (2012) para extração automática de regras via um algoritmo cobertura assume que só podem ser trabalhados dados categóricos, devendo os mesmos estar representados de acordo com os factos attribute (atributo de dados e seus valores possíveis) e example (instância, com um exemplo completo de pares atributo-valor).

Para facilitar o pré-processamento de dados, optou-se por criar um código na linguagem estatística R, via o ficheiro **preprocess.R**. Este ficheiro deve ser executado no ambiente R (que pode ser descarregado gratuitamente a partir de <http://www.r-projet.org>), conforme comando `source` executado na diretoria atual de trabalho (onde estão os ficheiros `.pl` e `.R`) e descrito abaixo:

```
> source("preprocess.R")
trying URL 'http://archive.ics.uci.edu/ml/machine-learning-databases/00320/student.zip'
Content type 'application/zip' length 20478 bytes (19 KB)
=====
downloaded 19 KB

school sex age address famsize Pstatus Medu Fedu
gp:349 f:208 a15 : 82 r: 88 gt3:281 a: 41 e0: 3 e0: 2
ms: 46 m:187 a16 :104 u:307 le3:114 t:354 e1: 59 e1: 82
a17 : 98 e2:103 e2:115
a18 : 82 e3: 99 e3:100
maior18: 29 e4:131 e4: 96

Mjob Fjob reason guardian traveltime studytime
at_home : 59 at_home : 20 course :145 father: 90 t1:257 t1:105
health : 34 health : 18 home :109 mother:273 t2:107 t2:198
other :141 other :217 other : 36 other : 32 t3: 23 t3: 65
services:103 services:111 reputation:105 t4: 8 t4: 27
teacher : 58 teacher : 29

failures schoolsup famsup paid activities nursery higher internet
f0:312 no :344 no :153 no :214 no :194 no : 81 no : 20 no : 66
f1: 50 yes: 51 yes:242 yes:181 yes:201 yes:314 yes:375 yes:329
f2: 17
f3: 16

romantic famrel freetime goout Dalc Walc health absences G3
no :263 fr1: 8 ft1: 19 g1: 23 da1:276 wa1:151 h1: 47 a0:115 fail:130
yes:132 fr2: 18 ft2: 64 g2:103 da2: 75 wa2: 85 h2: 45 a1:129 pass:265
fr3: 68 ft3:157 g3:130 da3: 26 wa3: 80 h3: 91 a2: 85
fr4:195 ft4:115 g4: 86 da4: 9 wa4: 51 h4: 66 a3: 51
fr5:106 ft5: 40 g5: 53 da5: 9 wa5: 28 h5:146 a4: 15
```

O pré-processamento consiste: na obtenção do conjunto de dados via um pedido `http`; descompactação de um ficheiro `csv` a partir do ficheiro original `zip`; na seleção de alguns atributos de entrada (todos exceto `G1` e `G2`) e o atributo de saída (`G3`);

transformação de nomes de valores de atributos, de modo a ter somente letras minúsculas; conversão dos atributos numéricos para categóricos (usando conhecimento do domínio e até um máximo de 5 níveis por atributo). O ficheiro R mostra ainda um sumário dos dados pré-processados, gravando o conjunto de dados resultante em dois tipos de ficheiros: `math.csv` (para posterior leitura por um programa de *machine learning/data mining*, como o R ou Weka), e `math.pl`, para posterior tratamento via Prolog. Para mais detalhes sobre o conjunto de dados analisado, consultar (Cortez & Silva, 2008). O ficheiro `math.pl` contém um elevado conjunto de factos `attribute` e `example`, dois quais se exemplificam somente alguns:

math.pl (ficheiro incompleto)

```
attribute(school, [ gp, ms]).
attribute(sex, [ f, m]).
attribute(age, [ a15, a16, a17, a18, maior18]).
% ...
data([gp, f, a18, u, gt3, a, e4, e4, at_home, teacher, course,
mother, t2, t2, f0, yes, no, no, no, yes, yes, no, no, fr4, ft3, g4,
dal, wal, h3, a2, fail]).
data([gp, f, a17, u, gt3, t, e1, e1, at_home, other, course, father,
t1, t2, f0, no, yes, no, no, no, yes, yes, no, fr5, ft3, g3, dal,
wal, h3, a1, fail]).
data([gp, f, a15, u, le3, t, e1, e1, at_home, other, other, mother,
t1, t2, f3, yes, no, yes, no, yes, yes, yes, no, fr4, ft3, g2, da2,
wa3, h3, a2, pass]).
% ...
```

Todo o restante código Prolog é executado via programa SWI-Prolog. A sequência de ficheiros a executar deverá ser: `math1.pl` e depois `math_rules.pl`. O ficheiro `math1.pl` cria o conjunto de dados `math2.pl`, que está já no formato exigido pelo algoritmo de cobertura para extração automática de regras. A transformação de formatos é obtida via o predicado `preprocess` (do ficheiro `preprocess.pl`). De seguida, exemplifica-se uma porção do código `math2.pl`:

math2.pl (ficheiro incompleto)

```
attribute(school, [ gp, ms]).
attribute(sex, [ f, m]).
attribute(age, [ a15, a16, a17, a18, maior18]).
% ...
:- dynamic example/2.

example(fail, [school=gp, sex=f, age=a18, address=u, famsize=gt3,
pstatus=a, medu=e4, fedu=e4, mjob=at_home, fjob=teacher,
reason=course, guardian=mother, traveltime=t2, studytime=t2,
failures=f0, schoolsup=yes, famsup=no, paid=no, activities=no,
nursery=yes, higher=yes, internet=no, romantic=no, famrel=fr4,
freetime=ft3, goout=g4, dalc=dal, walc=wal, health=h3, absences=a2]).
example(fail, [school=gp, sex=f, age=a17, address=u, famsize=gt3,
pstatus=t, medu=e1, fedu=e1, mjob=at_home, fjob=other, reason=course,
```

```

guardian=father, traveltime=t1, studytime=t2, failures=f0,
schoolsup=no, famsup=yes, paid=no, activities=no, nursery=no,
higher=yes, internet=yes, romantic=no, famrel=fr5, freetime=ft3,
goout=g3, dalc=da1, walc=wa1, health=h3, absences=a1]).
example(pass, [school=gp, sex=f, age=a15, address=u, famsize=le3,
pstatus=t, medu=e1, fedu=e1, mjob=at_home, fjob=other, reason=other,
guardian=mother, traveltime=t1, studytime=t2, failures=f3,
schoolsup=yes, famsup=no, paid=yes, activities=no, nursery=yes,
higher=yes, internet=yes, romantic=no, famrel=fr4, freetime=ft3,
goout=g2, dalc=da2, walc=wa3, health=h3, absences=a2])).
% ...

```

Realça-se que a linguagem de programação Prolog não é muito eficiente para um processamento intenso de dados, pelo que a execução do código `math_rules`, e seu predicado principal `learn_rules`, demora algum tempo. O resultado que se obtém (da execução no SWI-Prolog) é:

```

?- [math_rules].
true.

?- learn_rules.

fail <==
[failures=f2,studytime=t2]
[failures=f3,fjob=services]
[failures=f2,studytime=t1]
[failures=f3,age=maior18]
[higher=no,absences=a0]
[failures=f1,traveltime=t3]
[dalc=da4,traveltime=t2]
[absences=a4,sex=m]
[failures=f1,freetime=ft3,sex=m]
[schoolsup=yes,mjob=teacher]
[schoolsup=yes,fjob=at_home]
[absences=a4,age=a15]
[freetime=ft1,pstatus=a]
[failures=f1,goout=g1]
[failures=f3,walc=wa2]
[failures=f1,health=h4,age=a18]
[traveltime=t4,mjob=at_home]
[failures=f2,fedu=e4]
[age=maior18,absences=a2]
[goout=g4,guardian=other,absences=a0]
[schoolsup=yes,mjob=health]
[goout=g5,guardian=other]
[mjob=at_home,fjob=teacher]
[goout=g5,freetime=ft1]
[absences=a4,mjob=at_home]
[absences=a0,goout=g5,freetime=ft4]
[goout=g4,internet=no,walc=wa1]
[fedu=e1,mjob=health]
[schoolsup=yes,reason=other,medu=e4]

```

```

[famrel=fr3,goout=g5,paid=no]
[sex=f,absences=a0,age=maior18]
[sex=f,absences=a0,guardian=other]
[sex=f,medu=e3,mjob=teacher]
[sex=f,medu=e3,health=h4,schoolsup=no]
[medu=e2,walc=wa4,age=a18]
[absences=a4,famrel=fr5,famsize=le3]
[schoolsup=yes,studytime=t4,reason=course]
[mjob=at_home,goout=g5,age=a17]
[famrel=fr3,fjob=health,sex=f]
[fjob=at_home,traveltime=t3]
[health=h3,walc=wa4,goout=g3]
[medu=e1,absences=a0,activities=yes]
[famrel=fr3,goout=g1,activities=yes]
[schoolsup=yes,fjob=health,medu=e2]
[famrel=fr2,walc=wa2]
[fjob=at_home,age=a15,sex=f]
[health=h3,mjob=teacher,age=a18]
[mjob=other,absences=a3,medu=e1]
[absences=a2,school=ms,fedu=e1]
[goout=g4,studytime=t3,fedu=e3]
[schoolsup=yes,freetime=ft2,famsize=le3]
[age=a17,mjob=teacher,fedu=e3]
[health=h3,pstatus=a,mjob=health]
[health=h2,internet=no,famsize=le3]
[health=h3,mjob=teacher,dalc=da2]
[age=a17,absences=a3,goout=g2]
[goout=g4,studytime=t3,mjob=teacher]
[absences=a0,health=h2,paid=yes]
[medu=e1,schoolsup=yes,fedu=e2]
[fedu=e1,health=h1,age=a18]
[goout=g4,walc=wa1,health=h5]
[mjob=at_home,absences=a0,health=h3]
[nursery=no,goout=g4,fedu=e3]
[fedu=e1,famrel=fr3,absences=a0]
[medu=e1,famrel=fr5,age=a17,sex=f]

```

pass <==

```

[absences=a1,famrel=fr5,nursery=yes]
[mjob=health,age=a16]
[health=h1,goout=g2]
[fedu=e0]
[absences=a1,romantic=yes,failures=f0]
[goout=g1,sex=m]
[studytime=t3,famsize=le3]
[absences=a2,walc=wa5]
[mjob=health,freetime=ft2]
[studytime=t3,health=h2]
[absences=a1,fedu=e4,schoolsup=no]
[goout=g2,freetime=ft5]
[absences=a2,fjob=health]
[failures=f0,mjob=services,walc=wa3]
[absences=a2,freetime=ft5]

```

[age=a16,reason=reputation,famsize=gt3]
 [studytime=t3,famsup=no,nursery=yes]
 [failures=f0,freetime=ft2,walc=wa2,activities=no]
 [mjob=health,school=ms]
 [failures=f0,medu=e0]
 [failures=f0,famrel=fr2,age=a17]
 [age=a15,guardian=other]
 [fjob=teacher,paid=yes]
 [health=h1,dalc=da3]
 [failures=f0,guardian=father,fedu=e4]
 [absences=a1,pstatus=a]
 [absences=a1,famrel=fr2]
 [absences=a1,fedu=e2,famsize=gt3]
 [health=h1,reason=home,pstatus=t]
 [goout=g2,walc=wa4]
 [famsize=le3,reason=other]
 [studytime=t4,reason=reputation,famsize=le3]
 [age=a16,absences=a4]
 [famsup=no,fjob=health]
 [pstatus=a,freetime=ft4]
 [dalc=da5,age=a18]
 [fedu=e3,walc=wa5]
 [age=a16,dalc=da4]
 [failures=f0,guardian=father,walc=wa2]
 [famsup=no,age=a15,absences=a0]
 [walc=wa3,mjob=at_home,sex=f]
 [age=a16,famrel=fr5,goout=g2]
 [goout=g1,schoolsup=yes]
 [mjob=health,studytime=t3]
 [absences=a2,medu=e2,reason=home]
 [goout=g3,freetime=ft5,school=gp]
 [walc=wa3,goout=g4,famsize=le3]
 [traveltime=t3,medu=e2]
 [health=h3,fjob=at_home,age=a18]
 [health=h3,walc=wa5,age=a17]
 [fedu=e3,medu=e2,age=a16]
 [mjob=health,goout=g5]
 [absences=a1,age=a18,sex=f]
 [pstatus=a,medu=e1]
 [reason=home,famrel=fr2]
 [absences=a1,walc=wa3,age=a17]
 [fjob=health,address=r,medu=e3]
 [fjob=teacher,absences=a4]
 [studytime=t4,medu=e4,sex=f]
 [absences=a2,health=h1,sex=m]
 [mjob=other,reason=other,age=a16]
 [fjob=at_home,age=a18,school=gp]
 [health=h3,goout=g4,fedu=e3]
 [reason=home,age=a15,fedu=e3]
 [age=a16,freetime=ft2,sex=m]
 [health=h3,age=a17,school=ms]
 [fedu=e2,studytime=t3,age=a17]
 [nursery=no,absences=a2,fedu=e2]


```

[age=a16,famrel=fr5,medu=e3]
[dalc=da2,health=h4,famrel=fr4]
[age=a16,goout=g4,reason=reputation]
[age=a16,goout=g4,health=h3]
true .

```

O algoritmo de cobertura conseguiu extrair duas regras principais (algo extensas), uma para o valor reprovação (fail) e outra para a aprovação (pass). Para além de mostrar na consola estas regras, o programa Prolog também cria o ficheiro `math_ifthen.pl` que será utilizado na alínea seguinte (b) e que se lista de seguida:

```
:- dynamic (<==)/2.
```

```

fail<==[[failures=f2, studytime=t2], [failures=f3, fjob=services], [failures=f2,
studytime=t1], [failures=f3, age=maior18], [higher=no, absences=a0], [failures=f1,
traveltime=t3], [dalc=da4, traveltime=t2], [absences=a4, sex=m], [failures=f1,
freetime=ft3, sex=m], [schoolsup=yes, mjob=teacher], [schoolsup=yes, fjob=at_home],
[absences=a4, age=a15], [freetime=ft1, pstatus=a], [failures=f1, goout=g1],
[failures=f3, walc=wa2], [failures=f1, health=h4, age=a18], [traveltime=t4,
mjob=at_home], [failures=f2, fedu=e4], [age=maior18, absences=a2], [goout=g4,
guardian=other, absences=a0], [schoolsup=yes, mjob=health], [goout=g5, guardian=other],
[mjob=at_home, fjob=teacher], [goout=g5, freetime=ft1], [absences=a4, mjob=at_home],
[absences=a0, goout=g5, freetime=ft4], [goout=g4, internet=no, walc=wa1], [fedu=e1,
mjob=health], [schoolsup=yes, reason=other, medu=e4], [famrel=fr3, goout=g5, paid=no],
[sex=f, absences=a0, age=maior18], [sex=f, absences=a0, guardian=other], [sex=f,
medu=e3, mjob=teacher], [sex=f, medu=e3, health=h4, schoolsup=no], [medu=e2, walc=wa4,
age=a18], [absences=a4, famrel=fr5, famsize=le3], [schoolsup=yes, studytime=t4,
reason=course], [mjob=at_home, goout=g5, age=a17], [famrel=fr3, fjob=health, sex=f],
[fjob=at_home, traveltime=t3], [health=h3, walc=wa4, goout=g3], [medu=e1, absences=a0,
activities=yes], [famrel=fr3, goout=g1, activities=yes], [schoolsup=yes, fjob=health,
medu=e2], [famrel=fr2, walc=wa2], [fjob=at_home, age=a15, sex=f], [health=h3,
mjob=teacher, age=a18], [mjob=other, absences=a3, medu=e1], [absences=a2, school=ms,
fedu=e1], [goout=g4, studytime=t3, fedu=e3], [schoolsup=yes, freetime=ft2, famsize=le3],
[age=a17, mjob=teacher, fedu=e3], [health=h3, pstatus=a, mjob=health], [health=h2,
internet=no, famsize=le3], [health=h3, mjob=teacher, dalc=da2], [age=a17, absences=a3,
goout=g2], [goout=g4, studytime=t3, mjob=teacher], [absences=a0, health=h2, paid=yes],
[medu=e1, schoolsup=yes, fedu=e2], [fedu=e1, health=h1, age=a18], [goout=g4, walc=wa1,
health=h5], [mjob=at_home, absences=a0, health=h3], [nursery=no, goout=g4, fedu=e3],
[fedu=e1, famrel=fr3, absences=a0], [medu=e1, famrel=fr5, age=a17, sex=f]].
pass<==[[absences=a1, famrel=fr5, nursery=yes], [mjob=health, age=a16], [health=h1,
goout=g2], [fedu=e0], [absences=a1, romantic=yes, failures=f0], [goout=g1, sex=m],
[studytime=t3, famsize=le3], [absences=a2, walc=wa5], [mjob=health, freetime=ft2],
[studytime=t3, health=h2], [absences=a1, fedu=e4, schoolsup=no], [goout=g2,
freetime=ft5], [absences=a2, fjob=health], [failures=f0, mjob=services, walc=wa3],
[absences=a2, freetime=ft5], [age=a16, reason=reputation, famsize=gt3], [studytime=t3,
famsup=no, nursery=yes], [failures=f0, freetime=ft2, walc=wa2, activities=no],
[mjob=health, school=ms], [failures=f0, medu=e0], [failures=f0, famrel=fr2, age=a17],
[age=a15, guardian=other], [fjob=teacher, paid=yes], [health=h1, dalc=da3],
[failures=f0, guardian=father, fedu=e4], [absences=a1, pstatus=a], [absences=a1,
famrel=fr2], [absences=a1, fedu=e2, famsize=gt3], [health=h1, reason=home, pstatus=t],
[goout=g2, walc=wa4], [famsize=le3, reason=other], [studytime=t4, reason=reputation,
famsize=le3], [age=a16, absences=a4], [famsup=no, fjob=health], [pstatus=a,

```

```
freetime=ft4], [dalc=da5, age=a18], [fedu=e3, walc=wa5], [age=a16, dalc=da4],
[failures=f0, guardian=father, walc=wa2], [famsup=no, age=a15, absences=a0], [walc=wa3,
mjob=at_home, sex=f], [age=a16, famrel=fr5, goout=g2], [goout=g1, schoolsup=yes],
[mjob=health, studytime=t3], [absences=a2, medu=e2, reason=home], [goout=g3,
freetime=ft5, school=gp], [walc=wa3, goout=g4, famsize=le3], [traveltime=t3, medu=e2],
[health=h3, fjob=at_home, age=a18], [health=h3, walc=wa5, age=a17], [fedu=e3, medu=e2,
age=a16], [mjob=health, goout=g5], [absences=a1, age=a18, sex=f], [pstatus=a, medu=e1],
[reason=home, famrel=fr2], [absences=a1, walc=wa3, age=a17], [fjob=health, address=r,
medu=e3], [fjob=teacher, absences=a4], [studytime=t4, medu=e4, sex=f], [absences=a2,
health=h1, sex=m], [mjob=other, reason=other, age=a16], [fjob=at_home, age=a18,
school=gp], [health=h3, goout=g4, fedu=e3], [reason=home, age=a15, fedu=e3], [age=a16,
freetime=ft2, sex=m], [health=h3, age=a17, school=ms], [fedu=e2, studytime=t3, age=a17],
[nursery=no, absences=a2, fedu=e2], [age=a16, famrel=fr5, medu=e3], [dalc=da2,
health=h4, famrel=fr4], [age=a16, goout=g4, reason=reputation], [age=a16, goout=g4,
health=h3]].
```

Código da Resolução da alínea b):

build_es.pl

```
% Paulo Cortez @2018
:- op( 300, xfx, <==).
:-[math_ifthen,backward].

% transform into if then rules:
ruleconv:-
    Concl <== List, % one extracted rule
    ruleconv(List,Concl), % convert to if then rules
    retractall(Concl <== List), % remove all <== rules
    ruleconv. % try another extracted rule
ruleconv:- !. % end

ruleconv([],_). % end
ruleconv(L,C):-
    select(X,L,L2), % select X
    process_cond(X,Cond), % convert X to Cond
    assert(if Cond then C), % assert rule
    ruleconv(L2,C). % process L2

% add one if Cond then Concl rule:
process_cond([Att=Val],C):- % end
    atomic_list_concat([Att,"_",Val],C). % nice format
process_cond([Att=Val|R],C and CR):-
    atomic_list_concat([Att,"_",Val],C), % nice format
    process_cond(R,CR).

% save Expert System
:- ruleconv,
    tell('es1.pl'),
    listing(if),
    told.
```

demo_es1.pl

```
:-[backward,es1].

fact(age_a15).
```

```
fact(guardian_other).
```

```
%demo(pass).
```

Comentário sobre a Resolução:

O ficheiro `build_es.pl` cria automaticamente as regras de produção, no formato **if Cond then Concl** do capítulo anterior, a partir das regras geradas pelo algoritmo de cobertura e gravadas no ficheiro `math_ifthen.pl`. O resultado da sua execução é:

```
?- [build_es].  
true.
```

Para o efeito, utiliza o útil predicado `atomic_list_concat` do SWI-Prolog e que agrupa num único elemento atómico os valores de uma lista. As regras de produção são gravadas no ficheiro `es1.pl` e que é utilizado pelo ficheiro de demonstração `demo_es1.pl`. Este ficheiro usa o sistema de inferência `backward.pl` apresentado no capítulo anterior. De seguida, mostra-se o conteúdo do ficheiro `es1.pl` que contém um conjunto elevado de regras de produção que foram geradas de modo automático:

```
:- dynamic (if)/1.  
  
if failures_f2 and studytime_t2 then fail.  
if failures_f3 and fjob_services then fail.  
if failures_f2 and studytime_t1 then fail.  
if failures_f3 and age_maior18 then fail.  
if higher_no and absences_a0 then fail.  
if failures_f1 and traveltime_t3 then fail.  
if dalc_da4 and traveltime_t2 then fail.  
if absences_a4 and sex_m then fail.  
if failures_f1 and freetime_ft3 and sex_m then fail.  
if schoolsup_yes and mjob_teacher then fail.  
if schoolsup_yes and fjob_at_home then fail.  
if absences_a4 and age_a15 then fail.  
if freetime_ft1 and pstatus_a then fail.  
if failures_f1 and goout_g1 then fail.  
if failures_f3 and walc_wa2 then fail.  
if failures_f1 and health_h4 and age_a18 then fail.  
if traveltime_t4 and mjob_at_home then fail.  
if failures_f2 and fedu_e4 then fail.  
if age_maior18 and absences_a2 then fail.  
if goout_g4 and guardian_other and absences_a0 then fail.  
if schoolsup_yes and mjob_health then fail.  
if goout_g5 and guardian_other then fail.  
if mjob_at_home and fjob_teacher then fail.  
if goout_g5 and freetime_ft1 then fail.  
if absences_a4 and mjob_at_home then fail.  
if absences_a0 and goout_g5 and freetime_ft4 then fail.  
if goout_g4 and internet_no and walc_wa1 then fail.  
if fedu_e1 and mjob_health then fail.  
if schoolsup_yes and reason_other and medu_e4 then fail.  
if famrel_fr3 and goout_g5 and paid_no then fail.  
if sex_f and absences_a0 and age_maior18 then fail.  
if sex_f and absences_a0 and guardian_other then fail.  
if sex_f and medu_e3 and mjob_teacher then fail.  
if sex_f and medu_e3 and health_h4 and schoolsup_no then fail.  
if medu_e2 and walc_wa4 and age_a18 then fail.  
if absences_a4 and famrel_fr5 and famsize_le3 then fail.  
if schoolsup_yes and studytime_t4 and reason_course then fail.  
if mjob_at_home and goout_g5 and age_a17 then fail.  
if famrel_fr3 and fjob_health and sex_f then fail.  
if fjob_at_home and traveltime_t3 then fail.  
if health_h3 and walc_wa4 and goout_g3 then fail.  
if medu_e1 and absences_a0 and activities_yes then fail.  
if famrel_fr3 and goout_g1 and activities_yes then fail.  
if schoolsup_yes and fjob_health and medu_e2 then fail.  
if famrel_fr2 and walc_wa2 then fail.  
if fjob_at_home and age_a15 and sex_f then fail.  
if health_h3 and mjob_teacher and age_a18 then fail.
```

if mjob_other and absences_a3 and medu_e1 then fail.
 if absences_a2 and school_ms and fedu_e1 then fail.
 if goout_g4 and studytime_t3 and fedu_e3 then fail.
 if schoolsup_yes and freetime_ft2 and famsize_le3 then fail.
 if age_a17 and mjob_teacher and fedu_e3 then fail.
 if health_h3 and pstatus_a and mjob_health then fail.
 if health_h2 and internet_no and famsize_le3 then fail.
 if health_h3 and mjob_teacher and dalc_da2 then fail.
 if age_a17 and absences_a3 and goout_g2 then fail.
 if goout_g4 and studytime_t3 and mjob_teacher then fail.
 if absences_a0 and health_h2 and paid_yes then fail.
 if medu_e1 and schoolsup_yes and fedu_e2 then fail.
 if fedu_e1 and health_h1 and age_a18 then fail.
 if goout_g4 and walc_wal and health_h5 then fail.
 if mjob_at_home and absences_a0 and health_h3 then fail.
 if nursery_no and goout_g4 and fedu_e3 then fail.
 if fedu_e1 and famrel_fr3 and absences_a0 then fail.
 if medu_e1 and famrel_fr5 and age_a17 and sex_f then fail.
 if absences_a1 and famrel_fr5 and nursery_yes then pass.
 if mjob_health and age_a16 then pass.
 if health_h1 and goout_g2 then pass.
 if fedu_e0 then pass.
 if absences_a1 and romantic_yes and failures_f0 then pass.
 if goout_g1 and sex_m then pass.
 if studytime_t3 and famsize_le3 then pass.
 if absences_a2 and walc_wa5 then pass.
 if mjob_health and freetime_ft2 then pass.
 if studytime_t3 and health_h2 then pass.
 if absences_a1 and fedu_e4 and schoolsup_no then pass.
 if goout_g2 and freetime_ft5 then pass.
 if absences_a2 and fjob_health then pass.
 if failures_f0 and mjob_services and walc_wa3 then pass.
 if absences_a2 and freetime_ft5 then pass.
 if age_a16 and reason_reputation and famsize_gt3 then pass.
 if studytime_t3 and famsup_no and nursery_yes then pass.
 if failures_f0 and freetime_ft2 and walc_wa2 and activities_no then pass.
 if mjob_health and school_ms then pass.
 if failures_f0 and medu_e0 then pass.
 if failures_f0 and famrel_fr2 and age_a17 then pass.
 if age_a15 and guardian_other then pass.
 if fjob_teacher and paid_yes then pass.
 if health_h1 and dalc_da3 then pass.
 if failures_f0 and guardian_father and fedu_e4 then pass.
 if absences_a1 and pstatus_a then pass.
 if absences_a1 and famrel_fr2 then pass.
 if absences_a1 and fedu_e2 and famsize_gt3 then pass.
 if health_h1 and reason_home and pstatus_t then pass.
 if goout_g2 and walc_wa4 then pass.
 if famsize_le3 and reason_other then pass.
 if studytime_t4 and reason_reputation and famsize_le3 then pass.
 if age_a16 and absences_a4 then pass.
 if famsup_no and fjob_health then pass.
 if pstatus_a and freetime_ft4 then pass.
 if dalc_da5 and age_a18 then pass.
 if fedu_e3 and walc_wa5 then pass.
 if age_a16 and dalc_da4 then pass.
 if failures_f0 and guardian_father and walc_wa2 then pass.
 if famsup_no and age_a15 and absences_a0 then pass.
 if walc_wa3 and mjob_at_home and sex_f then pass.
 if age_a16 and famrel_fr5 and goout_g2 then pass.
 if goout_g1 and schoolsup_yes then pass.
 if mjob_health and studytime_t3 then pass.
 if absences_a2 and medu_e2 and reason_home then pass.
 if goout_g3 and freetime_ft5 and school_gp then pass.
 if walc_wa3 and goout_g4 and famsize_le3 then pass.
 if traveltime_t3 and medu_e2 then pass.
 if health_h3 and fjob_at_home and age_a18 then pass.
 if health_h3 and walc_wa5 and age_a17 then pass.
 if fedu_e3 and medu_e2 and age_a16 then pass.
 if mjob_health and goout_g5 then pass.
 if absences_a1 and age_a18 and sex_f then pass.
 if pstatus_a and medu_e1 then pass.
 if reason_home and famrel_fr2 then pass.
 if absences_a1 and walc_wa3 and age_a17 then pass.
 if fjob_health and address_r and medu_e3 then pass.
 if fjob_teacher and absences_a4 then pass.
 if studytime_t4 and medu_e4 and sex_f then pass.
 if absences_a2 and health_h1 and sex_m then pass.
 if mjob_other and reason_other and age_a16 then pass.

```

if fjob_at_home and age_a18 and school_gp then pass.
if health_h3 and goout_g4 and fedu_e3 then pass.
if reason_home and age_a15 and fedu_e3 then pass.
if age_a16 and freetime_ft2 and sex_m then pass.
if health_h3 and age_a17 and school_ms then pass.
if fedu_e2 and studytime_t3 and age_a17 then pass.
if nursery_no and absences_a2 and fedu_e2 then pass.
if age_a16 and famrel_fr5 and medu_e3 then pass.
if dalc_da2 and health_h4 and famrel_fr4 then pass.
if age_a16 and goout_g4 and reason_reputation then pass.
if age_a16 and goout_g4 and health_h3 then pass.

```

O resultado da execução SWI-Prolog do ficheiro de demonstração é:

```

?- [demo_es1].
true.

```

```

?- demo(pass).
true .

```

Código da Resolução da alínea c):

math_dt.pl

```

:-[satisfy,induce_tree,math2],dynamic(tree/2).

% note: the minimum code should be: induce_tree(T),show(T).
learn_tree:-
    induce_tree(T), % get decision tree
    show(T), % show decision tree
    assert(T), % save into memory
    tell('math_tree.pl'), % save into file
    listing(tree/2),
    told.

```

induce_tree.pl

```

% decision tree code, uses information gain
% adapted from (Bratko 2012) and from the code by
% Vít Novotný: https://gitlab.fi.muni.cz/xnovot32/pb016-priklady.git

% usage: induce_tree(T), show(T).

% induce the decision tree, IDT - induced decision tree:
induce_tree( Tree ) :-
    findall( example( Class, Obj), example( Class, Obj), Examples),
    findall( Att, attribute( Att, _ ), Attributes),
    induce_tree( Attributes, Examples, Tree).

% induce_tree( Attributes , Examples, Tree)
induce_tree(_,[], null ) :- !.
induce_tree(_,[example( Class,_) | Examples], leaf( Class)) :-
    \+ ((member(example( ClassX,_), Examples), ClassX \== Class)), !.
% No other example of different class
induce_tree(Attributes , Examples, tree( Attribute , SubTrees)) :-
    choose_attribute( Attributes , Examples, Attribute/_), ! ,
    del( Attribute , Attributes , RestAtts), % delete attribute
    attribute( Attribute , Values),
    induce_trees( Attribute , Values, RestAtts, Examples, SubTrees).
induce_tree(_, Examples, leaf( ExClasses)) :- % no useful attribute
    findall(Class, member( example( Class, _), Examples), ExClasses).

% induce_trees( Att, Values, RestAtts, Examples, SubTrees):
% -> induces decision SubTrees for subsets of Examples according to
Values of Attribute:

```

```

induce_trees( _, [ ], _, [ ] ). % No attributes, no subtrees
induce_trees( Att , [Vall | Vals ], RestAtts, Exs, [Vall : Tree1 |
Trees] ) :-
    attval_subset( Att = Vall, Exs, ExampleSubset),
    induce_tree( RestAtts, ExampleSubset, Tree1),
    induce_trees( Att , Vals, RestAtts, Exs, Trees).

% attval_subset(Attribute = Value, Examples, Subset):
% is true if Subset is the subset of examples in Examples that
% satisfy the condition Attribute = Value
attval_subset( AttributeValue, Examples, ExampleSubset) :-
    findall(example(Class, Obj),
        (member( example( Class, Obj), Examples), satisfy( Obj, [
AttributeValue ])),
        ExampleSubset).

% choose_attribute( +Atts, +Examples, -BestAtt/BestGain)
choose_attribute( [ ], _, 0/0).
choose_attribute( [Att], Examples, Att/Gain):-!, gain(Examples, Att,
Gain).
choose_attribute( [Att|Atts], Examples, BestAtt/BestGain):-
    choose_attribute(Atts,Examples,BestAtt1/BestGain1),
    gain(Examples, Att, Gain),
    (Gain>BestGain1,!,BestAtt=Att,BestGain=Gain;
    BestAtt=BestAtt1,BestGain=BestGain1).

% gain( +Examples, +Attribute, -Gain) : gain of Attribute
gain( Exs, Att , Gain) :- attribute( Att , AttVals ),
    length(Exs, Total),
    setof(Class, X^example(Class,X), Classes),          % The set of
all Class, see 'help(setof)'
    findall(Nc, (member(C,Classes), cntclass(C,Exs,Nc)), CCnts),
    info(CCnts,Total,I),
    rem(Att, AttVals,Exs,Classes,Total,Rem),
    Gain is I-Rem.

% info(+ValueCounts, +Total, -I) - rate information
info( [ ], _, 0).
info( [VC|ValueCounts], Total, I) :-
    info(ValueCounts,Total,I1),
    (VC = 0, !, I is I1;
    Pvi is VC / Total,
    log2(Pvi, LogPvi), I is - Pvi * LogPvi + I1).

% rem( +Att, +AttVals, +Exs, +Classes, +Total, -Rem) - "Residual
information," after a test on all attribute values Att
rem( _, [ ], _, _, _, 0).
rem( Att, [V | Vs], Exs, Classes, Total, Rem) :-
    findall(1, (member(example( _, AVs),Exs), member(Att = V, AVs)),
L1), length(L1, Nv),
    findall(Ni, (member(C, Classes), cntclassattv(Att,V,C,Exs,Ni)),
VCnts),
    Pv is Nv / Total, % P(v)
    info(VCnts,Nv,I),
    rem(Att,Vs,Exs,Classes,Total,Rem1),
    Rem is Pv * I + Rem1.

% cntclass( +Class, +Exs, -Cnt) - number of examples of Class
cntclass( Class, Exs, Cnt) :-
    findall(1, member(example(Class, _),Exs), L), length(L, Cnt).

% cntclass( +Att, +Val, +Class, +Exs, -Cnt) - Number of instances for
the Class attribute value Val Att

```

```

cntclassattv( Att, Val, Class, Exs, Cnt) :-
    findall(1, (member(example(Class,AVs),Exs), member(Att = Val,
AVs)), L), length(L, Cnt).

% log2(+X, -Y)
log2(X, Y) :- Y is log(X) / log(2).

% =====
% show(+X,+L,-L1)
del(A,[A|T],T).
del(A,[H|T1],[H|T2]) :- del(A,T1,T2).

% show(+Tree)
show(Tree) :-
    show(Tree, 0).

% show(+Tree, +Ind)
show(leaf(Class), Ind) :-
    tab(Ind), write(Class), nl.
show(tree(A, SubTrees), Ind) :-
    tab(Ind), write(A), write('?'), nl,
    NI is Ind+2, show(SubTrees, NI).
show([], _).
show([_ : null | SubTrees], Ind) :- !, show(SubTrees, Ind).
show([V1 : ST1 | SubTrees], Ind) :-
    tab(Ind), write('= '), write(V1), nl,
    NI is Ind+2, show(ST1, NI),
    show(SubTrees, Ind).

```

Comentário sobre a Resolução:

O ficheiro **math_dt.pl** usa o mesmo conjunto de dados **math2.pl** da alínea a), executando agora um algoritmo de árvore de decisão, que se baseia no critério de Information Gain (IG) para selecionar os atributos mais informativos, conforme descrito no livro (Brakto, 2012). A execução do algoritmo de árvore de decisão é bem mais rápida do que o algoritmo de cobertura da alínea a). A árvore de decisão obtida, apresentada na consola e no ficheiro **math_tree.pl** tem uma elevada dimensão, uma vez que não é realizada nenhum tipo de “poda”. Como tal, é somente aqui apresentado uma porção inicial (e incompleta) desta árvore, conforme o que é mostrado na consola do SWI-Prolog:

```

?- [math_dt].
true.

?- learn_tree.
failures?
  = f0
    absences?
      = a0
        goout?
          = g1
            famrel?
              = fr3
                fail
              = fr4
                pass
              = fr5
                pass
          = g2
            famrel?
              = fr2
                fail
              = fr3
                walc?
                  = wa1

```

```

      fail
    = wa2
      pass
    = wa3
      fail
= fr4
health?
  = h1
    pass
  = h2
    fail
  = h3
    pass
  = h5
    pass
= fr5
  pass

```

A título de exemplo, refere-se que a primeira regra da árvore corresponde à expressão: *if failures=f0 and absences=a0 and goout=g1 and famrel=fr3 then fail.*

Como complemento apresenta-se o código `math_dt.R` da linguagem R, mais indicada para tratamento computacional de grandes volumes de dados e que obtém uma árvore de decisão mais simples, conforme demonstrado na imagem do ficheiro `math_dt.pdf`. Para mais detalhes sobre a ferramenta R e pacote `rminer`, consultar (Cortez, 2015). A execução faz-se na ferramenta R via comando `source`:

```
> source("math_dt.R").
```

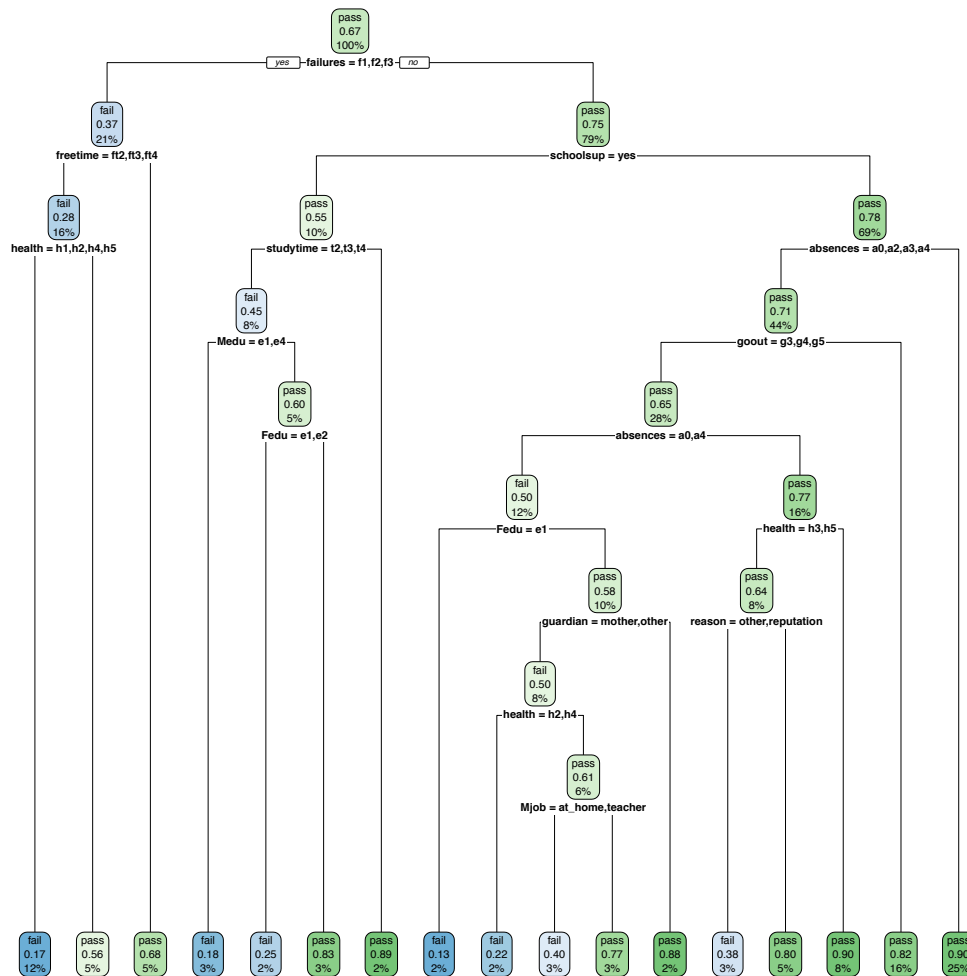
math_dt.R (ficheiro R)

```

# install.packages("rminer") # uncomment this line if not installed
# install.packages("rpart.plot") # uncomment this line if not installed
# read csv file into object
d=read.table("math.csv",header=TRUE,sep="," ,stringsAsFactors=TRUE)
library(rminer)
library(rpart.plot)
# fit a decision tree using default minsplit:
M=fit(G3~.,data=d,model="rpart")
# plot the decision tree:
pdf("math_dt.pdf")
rpart.plot(M@object)
dev.off()

```

O conteúdo do ficheiro math_dt.pdf que foi gerado é:



Uma regra para reprovação é: *if failures=f1,f2,f3 and freetime=ft2,ft3,ft4 and health=h1,h2,h4,h5 then fail.*

5 Procura

5.1 Estradas do Norte de Portugal (procura num espaço de soluções)

Enunciado:

Considere as seguintes cidades do Norte de Portugal: Porto, Póvoa de Varzim, Barcelos, Vila Nova de Famalicão, Braga e Guimarães. Determine a rota e distância a percorrer por uma viatura automóvel entre as cidades de Porto e Guimarães.

Código da Resolução:

norte.pl

```
:-[search].

% -- database:
%   simple state representation: S, where S is a city
initial(porto). % initial city
goal(quimaraes). % destination city

% --- knowledge base:
% road(Origin, Destination, Distance-in-km)
% data collected from: https://pt.distance.to/
road(porto,povoavarzim,37).
road(povoavarzim,barcelos,24).
road(povoavarzim,famalicao,30).
road(barcelos,braga,25).
road(barcelos,famalicao,23).
road(famalicao,porto,36).
road(famalicao,braga,24).
road(braga,quimaraes,25).
road(famalicao,quimaraes,29).

% travel(City1, City2, distance):
travel(X,Y,KM):- (road(X,Y,KM);road(Y,X,KM)). % if road or symmetrical

% state transition rule s/2: s(City1, City2)
s(N1,N2):- travel(N1,N2,_). % link s(O,D,Dist) with s(O,D)

% evaluation function: (sum of distances for all pairs)
eval([_],0).
eval([City1, City2|R],DS):-
    travel(City1, City2, D),
    eval([City2|R],DR),
    DS is D+DR.

% execute and show a search method result:
run(Method):- search(Method,Par,S),
    write('method:'),write(Method),writepar(Par),nl,
    write('solution:'),write(S),nl,
    length(S,N),N1 is N-1,write('solution
steps:'),write(N1),nl,
    eval(S,D),write('distance:'),write(D).
% write parameter (if any):
writepar(X):- integer(X),write(' par:'),write(X). % write X
writepar(_). % do not write X

% execute 3 example searches:
q1:- run(depthfirst).
```

```
q2:- run(iterativedeepening).
q3:- run(breadthfirst).
```

search.pl

```
% Paulo Cortez 2021@
:-[depthfirst,breadthfirst]. % consult all search types

% search method: generic for different search methods
% Method - search method, one of: depthfirst, depthfirst2,
iterativedeepening, breadthfirst
% Par - any parameters associated with the search method. The current
parameters are:
%     Maxdepth for depthfirst2 or iterativedeepening
% Solution - list with the path of states from initial to goal.
% Moves - list with transitions from initial to goal.

% first variant, transition with the Move: s(S1,Move,S2) is defined
(arity of /3) and shows
% the final Solution and Moves, executing a reverse on the original
solution.
% use this method if you define transition:
search(Method,Par,Solution,Moves):-
    initial(S0),
    execute(Method,Par,S0,Solution1),
    ifreverseneeded(Method,Solution1,Solution),
    get_moves(Solution,Moves).

% second variant, no Move transition is defined,
% use this method if only s(S1,S2) is defined:
search(Method,Par,Solution):-
    initial(S0),
    execute(Method,Par,S0,Solution1),
    ifreverseneeded(Method,Solution1,Solution).

% third option, no Move transition is defined, no reverse is
executed,
%search3(Method,Par,Solution):-
%     initial(S0),
%     execute(Method,Par,S0,Solution).

% depthfirst2 and iterativedeepening do not need a reverse:
ifreverseneeded(depthfirst2,S,S):-!. % cut is needed because
of backtracking
ifreverseneeded(iterativedeepening,S,S):-!. % cut is needed because
of backtracking
ifreverseneeded(_,S,S1):- reverse(S,S1). % other methods

% get all moves for a particular solution:
% assuming that a particular Solution was found,
% returns Moves - the list of actions/moves to result in this
Solution:
get_moves([_],[ ]).
get_moves([N1,N2|RN],[Move|RM]):-
    s(N1,Move,N2),
    get_moves([N2|RN],RM).

% links execute with the type of search method used:
execute(depthfirst,_,Node,Solution):-
    depthfirst([],Node,Solution).
execute(depthfirst2,Maxdepth,Node,Solution):-
    depthfirst2(Node,Solution,Maxdepth).
% returns N, which is found by iterativedeepening
execute(iterativedeepening,N,Node,Solution):-
```

```

        iterativedeepening(Node,Solution,1,N).
execute(breadthfirst,_,Start,Solution):-
    breadthfirst([ [Start] | Z] - Z,Solution).

```

depthfirst.pl

```

% 3 depth-first methods; adapted from (Bratko, 2012)

% depthfirst( Path, Node, Solution):
%   extending the path [Node | Path] to a goal gives Solution
depthfirst( Path, Node, [Node | Path]) :-
    goal(Node).

depthfirst( Path, Node, Sol) :-
    s(Node, Node1),
    \+ member( Node1, Path),                % Prevent a cycle
    depthfirst( [Node | Path], Node1, Sol).

% Maxdepth depth first:
% depthfirst2( Node, Solution, Maxdepth):
%   Solution is a path, not longer than Maxdepth, from Node to a goal
depthfirst2( Node, [Node], _) :-
    goal( Node).

depthfirst2( Node, [Node | Sol], Maxdepth) :-
    Maxdepth > 0,
    s( Node, Node1),
    Max1 is Maxdepth - 1,
    depthfirst2( Node1, Sol, Max1).

% Iterative Deepening: Node, Solution, StartN, FinalN
iterativedeepening(Node,Solution,N,N):-
    depthfirst2(Node,Solution,N).
iterativedeepening(Node,Solution,N,NR):-
    %N < Maxdepth, % assure this condition when backtracking
    N1 is N+1,
    % write('n:'),write(N),nl, % verbose if needed
    iterativedeepening(Node,Solution,N1,NR).

```

breadthfirst.pl

```

% breadth first; adapted (Bratko,2012)

breadthfirst( [ [Node | Path] | _] - _, [Node | Path] ) :-
    goal( Node).

breadthfirst( [Path | Paths] - Z, Solution) :-
    extend( Path, NewPaths),
    append( NewPaths, Z1, Z),             % Add NewPaths at end
    Paths \== Z1,                          % Set of candidates not empty
    breadthfirst( Paths - Z1, Solution).

extend( [Node | Path], NewPaths) :-
    findall( [NewNode, Node | Path],
            ( s(Node,NewNode), \+ member(NewNode, [Node | Path])),
            NewPaths).

```

Comentário sobre a Resolução:

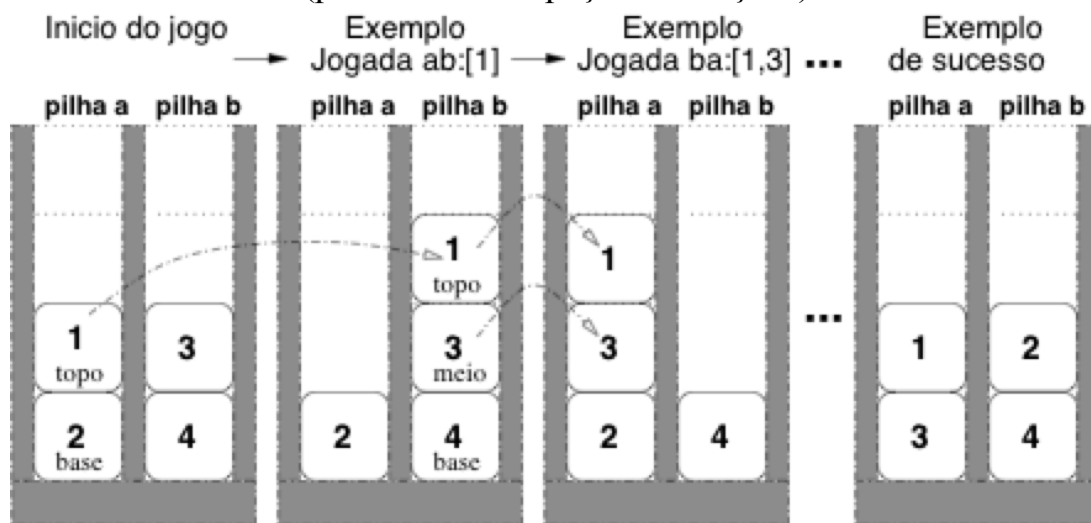
Numa procura via transição de estados é necessário: definir o estado inicial (*initial*) e o objetivo final (*goal*); as transições via predicado *s* (neste caso será usado o modo mais simples *s/2* com os argumentos de origem e destino); e utilizar o

sistema de inferência de acordo com o que está definido em `search.pl`. As transições dependem de um deslocamento possível entre duas cidades (medido pelo predicado `travel`). Como cada transição tem dois argumentos (`s/2`), utiliza-se o predicado `search` com três argumentos (`search/3: Method, Par, Solution`). Na resolução, foram definidos três tipos de procura: em profundidade (`q1`), *iterative deepening* (`q2`), e em largura (`q3`). Cada procura é executada via o predicado `run`, que mostra diversos elementos úteis na consola. Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```
?- [norte].
true.
?- q1.
method:depthfirst
solution:[porto,povoavarzim,barcelos,braga,guimaraes]
solution steps:4
distance:111
true .
?- q2.
method:iterativedeepening par:2
solution:[porto,famalicao,guimaraes]
solution steps:2
distance:65
true .
?- q3.
method:breadthfirst
solution:[porto,famalicao,guimaraes]
solution steps:2
distance:65
true .
```

Todas as três procuras obtêm uma rota válida. As procuras `q2` e `q3` obtêm uma solução que é ótima para o problema, exigindo somente 2 movimentos e com uma distância percorrida de 65 km.

5.2 Blocos Infantis (procura num espaço de soluções)



Enunciado:

Considere o seguinte jogo infantil de blocos de cubos:

Existem duas pilhas (**a** e **b**) e quatro cubos com números (**1,2,3,4**). O jogo inicia-se com **1** no topo e **2** na base da pilha **a** e **3** no topo e **4** na base da pilha **b**. Em cada jogada, a criança pode mover de 1 a 3 cubos da pilha **a** para a pilha **b** (ou vice-versa, de **b** para **a**). Por exemplo, no início pode mover-se o cubo 1 para a pilha **b** (jogada **ab:[1]**). Outra opção seria mover **1,2** para a pilha **b** (jogada **ab:[1,2]**), sendo que neste caso a pilha **a** ficaria vazia e a pilha **b** com os cubos **1,2,3,4** (desde o topo até à base). Não é possível trocar a ordem vertical dos cubos. Ou seja, no início não é possível fazer a jogada **ab:[2,1]**, onde a intenção era ter os cubos **2,1,3,4** na pilha **b**. O jogo **termina** quando existirem **cubos pares** numa pilha qualquer e **cubos ímpares** noutra (ver exemplo da figura).

- Para a técnica da procura de estados via transições, **defina** em Prolog o **estado inicial** (**initial**) e **objetivos** (**goal**) deste problema. Utilize a seguinte notação para o estado: **b(LA,LB)** onde **LA** é a lista dos blocos que existem na pilha **a** e **LB** a lista dos blocos que existem na pilha **b**. Por exemplo, se **LA** for **[1,3,2]** tal significa que na pilha **a** estão os cubos 1 no topo, 3 no meio e 2 na base.
- Codifique** em Prolog todas as **transições** **s(S1,Move,S2)** necessárias para resolver este jogo. Para cada jogada, utilize a notação: **ab:[L]** para mover os blocos da lista **L** da pilha **a** para a pilha **b**; ou **ba:[L]** para mover da pilha **b** para a pilha **a**. A figura mostra exemplos de 2 jogadas consecutivas efectuadas a partir do início: **ab:[1]** e **ba:[1,3]** (estes exemplos são demonstrativos, não quer dizer que tenham mesmo que ser seguidos para a obtenção da solução final).
- Execute** em Prolog diversos tipos de procura (e.g., em profundidade) para resolver este problema.

Código da Resolução:

cubes.pl

```
:-[search].

% state representation: b(L1,L2), where L1 and L2 are lists with
numbers
% initial state:
initial(b([1,2],[3,4])).
% goal all even or all odd numbers in L1:
goal(b([1,3],[_,_])).
goal(b([3,1],[_,_])).
goal(b([2,4],[_,_])).
goal(b([4,2],[_,_])).

% links s/2 with s/3:
s(S1,S2):- s(S1,_,S2).

% all state transitions: s(S1,Move,S2).
s(b(L,[A|R]),ba:[A],b([A|L],R)).
s(b(L,[A,B|R]),ba:[A,B],b([A,B|L],R)).
s(b(L,[A,B,C|R]),ba:[A,B,C],b([A,B,C|L],R)).
s(b([A|R],L),ab:[A],b(R,[A|L])).
s(b([A,B|R],L),ab:[A,B],b(R,[A,B|L])).
s(b([A,B,C|R],L),ab:[A,B,C],b(R,[A,B,C|L])).

% execute and show a search method result:
run(Method,Par):- search(Method,Par,S,Moves),
                  write('method:'),write(Method),writepar(Par),nl,
                  write('solution:'),write(S),nl,
                  length(Moves,N),write('solution steps:'),write(N),nl,
                  write('moves:'),write(Moves).

% write parameter (if any):
writepar(X):- integer(X),write(' par:'),write(X). % write X
writepar(_). % do not write X

% four different search execution examples:
q1:- run(depthfirst,_).
q2:- run(depthfirst2,3).
q3:- run(iterativedeepening,_).
q4:- run(breadthfirst,_).
```

Comentário sobre a Resolução:

Utilizando a notação definida no enunciado, é necessário: definir o estado inicial e o objetivo final (`goal`); as transições, via predicado `s` que tem três argumentos (estado origem, nome do movimento, estado destino); e utilizar o sistema de inferência de acordo com o que está definido em `search.pl`. Existe um total de 6 transições, três para cada pilha (a ou b), conforme o número de cubos a movimentar (1, 2 ou 3). Como cada transição tem três argumentos (`s/3`), utiliza-se o predicado `search` com quatro argumentos (`search/4: Method, Par, Solution, Moves`). Na resolução, foram definidos quatro tipos de procura: em profundidade (`q1`), em profundidade com um

Maxdepth fixo (q2, aqui com uma profundidade máxima de `par=3`), *iterative deepening* (q3), e em largura (q4). Cada procura é executada via o predicado `run`, que mostra diversos elementos úteis na consola. Para correr o programa no SWI-Prolog basta executar os seguintes comandos:

```
?- [cubes].
```

```
true.
```

```
?- q1.
```

```
method:depthfirst
```

```
solution:[b([1,2],[3,4]),b([3,1,2],[4]),b([4,3,1,2],[ ]),b([1,2],[4,3]
),b([4,1,2],[3]),b([3,4,1,2],[ ]),b([2],[3,4,1]),b([3,2],[4,1]),b([4,3
,2],[1]),b([1,4,3,2],[ ]),b([3,2],[1,4]),b([1,3,2],[4]),b([4,1,3,2],[ ]
),b([2],[4,1,3]),b([4,2],[1,3])]
```

```
solution steps:14
```

```
moves:[ba:[3],ba:[4],ab:[4,3],ba:[4],ba:[3],ab:[3,4,1],ba:[3],ba:[4],
ba:[1],ab:[1,4],ba:[1],ba:[4],ab:[4,1,3],ba:[4]]
```

```
true .
```

```
?- q2.
```

```
method:depthfirst2 par:3
```

```
solution:[b([1,2],[3,4]),b([3,1,2],[4]),b([],[3,1,2,4]),b([3,1],[2,4]
)]
```

```
solution steps:3
```

```
moves:[ba:[3],ab:[3,1,2],ba:[3,1]]
```

```
true .
```

```
?- q3.
```

```
method:iterativedeepening par:3
```

```
solution:[b([1,2],[3,4]),b([3,1,2],[4]),b([],[3,1,2,4]),b([3,1],[2,4]
)]
```

```
solution steps:3
```

```
moves:[ba:[3],ab:[3,1,2],ba:[3,1]]
```

```
true .
```

```
?- q4.
```

```
method:breadthfirst
```

```
solution:[b([1,2],[3,4]),b([3,1,2],[4]),b([],[3,1,2,4]),b([3,1],[2,4]
)]
```

```
solution steps:3
```

```
moves:[ba:[3],ab:[3,1,2],ba:[3,1]]
```

```
true .
```

Realça-se que a primeira procura não é muito eficiente, uma vez que exige um total de 14 movimentos. Em contraste, as procuras seguintes (q2, q3 e q4) obtêm a solução ótima para o problema, exigindo somente 3 movimentos.

6 Otimização (hill climbing)

6.1 Soma de *Bits*

Enunciado:

Considere o problema clássico (e acadêmico) de otimização: otimizar um conjunto de *bits* (0 ou 1), tal que a soma dos *bits* tenha o valor máximo. Resolva este problema de otimização via três métodos: procura cega, *hill climbing* e *stochastic hill climbing* (probabilidade de 20%), comparando os tempos de processamento e qualidade das soluções finais obtidas.

Código da Resolução:

sumbits.pl

```
:-[auxiliar,hill,search].
:- dynamic(dimension/1). % number of bits to optimize

%%% hill climbing approach: -----
% evaluate a solution:
eval(S,Bits):- sum_list(S,Bits).

% random flip one bit of list S1, return S2:
change(S1,S2):- binary_change(S1,S2).

% D is the dimension, number of bits to optimize % S is the solution
hrun1(D,S):- % standard hill climbing
    rep(D,0,S0), % initial solution S0
    % 100 iterations, report every 10 iterations:
    time(hill_climbing(S0,[100,10,0,max],S)).
hrun2(D,S):- % stochastic hill climbing Prob=0.2
    rep(D,0,S0), % initial solution S0
    % 100 iterations, report every 10 iterations
    time(hill_climbing(S0,[100,10,0.2,max],S)).

%%% blind search approach: -----
initial([]). % empty binary list
goal(S):- dimension(D),length(S,D),eval(S,D).

% transition s/2 s(S1,S2) - add a binary number 0 or 1:
s(S1,[0|S1]).
s(S1,[1|S1]).

brun(D,S):-
    retractall(dimension(_)),
    assert(dimension(D)), % update D
    time(blindsearch(D,S)).

blindsearch(D,S):-
    search(depthfirst2,D,S1),
    last(S1,S),
    eval(S,E1),
    show(final,1,S1,E1,_,_).

q1:-brun(20,S),write(S).
q2:-hrun1(20,S),write(S).
q3:-hrun2(20,S),write(S).
```

hill.pl

```
% hill climbing (standard and stochastic versions)
% Paulo Cortez @2021
% assumes that eval(Solution,Result) and change(S1,S2) are defined
% best solution achieved so far is stored in a special dynamic fact:
:- dynamic(hbest_sofar/2).

% internal auxiliary rules, used to update hbest_sofar:
update_hbest(S,E,Opt):- hbest_sofar(SB,EB),
                        best_opt(0,Opt,S,E,SB,EB,SR,ER),

retract(hbest_sofar(SB,EB)),assert(hbest_sofar(SR,ER)).

% return SR,ER the best value of S1 and S2:
best(Prob,Opt,S1,E1,S2,E2,SR,ER):-
    eval(S2,E2),
    update_hbest(S2,E2,Opt), % update hbest_sofar if needed
    best_opt(Prob,Opt,S1,E1,S2,E2,SR,ER).

best_opt(Prob,_,_,_,S2,E2,S2,E2):-
    random(X), % random from 0 to 1,
    X< Prob. % accept new solution

best_opt(_,Opt,S1,E1,S2,E2,SR,ER):- % else, select the best one
    ((Opt=max,max_list([E1,E2],ER));(Opt=min,min_list([E1,E2],ER))),
    ((ER==E1,SR=S1); (ER==E2,SR=S2)).

% show evolution:
show(final,Verbose,S1,E1,_,_):-
    Verbose>0,
    write('final:'),write(' S:'),write(S1),write('
E:'),write(E1),nl,!.
show(final2,Verbose,S1,E1,SB,EB):-
    Verbose>0,
    write('final:'),write(' S:'),write(S1),write('
E:'),write(E1),nl,
    write('best:'),write(' S:'),write(SB),write('
E:'),write(EB),nl,!.
show(0,Verbose,S1,E1,_,_):-
    Verbose>0,
    write('init:'),write(' S0:'),write(S1),write(' E0:'),
    write(E1),nl,!.
show(I,Verbose,S1,E1,S2,E2):-
    0 is I mod Verbose,
    write('iter:'),write(I),write(' S1:'),write(S1),
    write(' E1:'),write(E1),write(' S2:'),write(S2),
    write(' E2:'),write(E2),nl,!.
show(____,____).

% hill climbing
% Prob=0 is pure hill climbing, Prob>0 means Stochastic Hill Climbing
% S0 is the initial solution,
% Control is a list with the number of iterations,
% verbose in console, probability and type of optimization.
% ---
% more detail about Control:
% Iter -- the maximum number of iterations, the algorithm stops
%         after Iter iterations.
% Verbose -- used to show the algorithm progress, every Verbose
%            iterations it shows current solution and evaluation.
```

```

% Prob -- numeric value from 0.0 to 1.0;
%     if 0.0, then a pure hill climbing is performed,
%     if > 0.0, then Stochastic Hill Climbing is executed.
% Opt -- max or min. If max then a maximization task is assumed,
%     if min then a minimization task is executed.

% /3 arity, main function:
hill_climbing(S0,[Iter,Verbose,Prob,Opt],SB):-
    eval(S0,E0),
    retractall(hbest_sofar(_,_)), % remove any previous
hbest_sofar
    assert(hbest_sofar(S0,E0)), % update hbest_sofar
    show(0,Verbose,S0,E0,_,_),
    hill_climbing(S0,E0,0,Iter,Verbose,Prob,Opt,S1),

    hbest_sofar(SB,EB),
    eval(S1,E1),
    show(final2,Verbose,S1,E1,SB,EB).
% /8 arity, auxiliary function:
hill_climbing(S,_ ,Iter,Iter,_ ,_ ,S):-!.
hill_climbing(S1,E1,I,Iter,Verbose,Prob,Opt,SFinal):-
    change(S1,SNew),
    best(Prob,Opt,S1,E1,SNew,_ ,S2,E2),
    I1 is I+1,
    show(I1,Verbose,S1,E1,S2,E2),
    hill_climbing(S2,E2,I1,Iter,Verbose,Prob,Opt,SFinal),!.

```

auxiliar.pl

```

% auxiliar functions; Paulo Cortez @2018
% reverse bit:
flip(0,1).
flip(1,0).

% stochastic change of solution S1 made of bits:
% random flip one bit of S1, return S2:
binary_change(S1,S2):-
    length(S1,L),
    random_between(1,L,X),
    nth1(X,S1,Point),
    flip(Point,Point2),
    replace_list(S1,X,Point2,S2).

% repeat X in a list N times:
rep(N,X,L):-rep(1,N,X,L).
rep(N,N,X,[X]).
rep(I,N,X,[X|R]):-
    I1 is I+1,
    rep(I1,N,X,R).

% efficient replace of element in list (code retrieved from):
% http://stackoverflow.com/questions/8519203/prolog-replace-an-
% element-in-a-list-at-a-specified-index
% replace_list(List,Position,NewElement,Result)
:- use_module(library(clpfd)).
replace_list(Es, N, X, Xs) :-
    list_index0_index_item_replaced(Es,1,N,X,Xs).
list_index0_index_item_replaced([_|Es], I ,I, X, [X|Es]).
list_index0_index_item_replaced([E|Es], I0,I, X, [E|Xs]) :-
    I0 #< I,
    I1 #= I0+1,
    list_index0_index_item_replaced(Es, I1,I, X, Xs).

```


porque o espaço de procura é muito superior: 2^D para *depth search* e 100 (iterações) para o *hill climbing*.

6.2 Caixeiro Viajante

Enunciado:

Considere o problema clássico do caixeiro viajante, onde o caixeiro tem de viajar entre N cidades, pretendendo-se minimizar a distância percorrida. Resolva este problema de otimização via métodos de *hill climbing* e *stochastic hill climbing* (probabilidade de 20%), comparando a qualidade das soluções finais obtidas.

Código da Resolução:

traveling.pl

```
:-[auxiliar,hill].
% north cost region of Portugal:
% driving distance (in km), from: https://pt.distance.to
dist(porto,varzim,36.99).
dist(porto,famalicao,35.8).
dist(porto,barcelos,59.61).
dist(porto,braga,55.74).
dist(porto,viana,75.48).
dist(porto,valenca,114.88).
dist(varzim,famalicao,31.39).
dist(varzim,barcelos,21.34).
dist(varzim,braga,38.88).
dist(varzim,viana,41.44).
dist(varzim,valenca,85.35).
dist(famalicao,barcelos,21.45).
dist(famalicao,braga,24.76).
dist(famalicao,viana,69.89).
dist(famalicao,valenca,83.91).
dist(barcelos,braga,25.79).
dist(barcelos,viana,39.42).
dist(barcelos,valenca,68.48).
dist(braga,viana,62.46).
dist(braga,valenca,67.19).
dist(viana,valenca,52.26).
travel(X,Y,D):-dist(Y,X,D);dist(X,Y,D). % true if dist or symmetrical

% representation: S is a list of cities
% (first and last cities need to be the same)

% evaluation function: (sum of distances for all pairs)
eval([_],0).
eval([City1,City2|R],DS):-
    travel(City1,City2,D),
    eval([City2|R],DR),
    DS is D+DR.

% change: switch 2 adjacent cities at random position:
change(S1,S2):-
    length(S1,L),
    random_between(1,L,P1),
    change(S1,P1,L,S2).
% change travelling path, auxiliary function:
```



```

change(S1,P1,L,S2):- % general case, normal switch
    P1>2, P1<L, % normal switch
    nth1(P1,S1,City1),
    P2 is P1-1,
    nth1(P2,S1,City2),
    replace_list(S1,P1,City2,S11),
    replace_list(S11,P2,City1,S2).
change(S1,1,L,S2):- % P1 is 1
    P2 is L-1, % last non repeated city
    nth1(1,S1,City1),
    nth1(P2,S1,City2),
    replace_list(S1,1,City2,S11),
    replace_list(S11,P2,City1,S12),
    replace_list(S12,L,City2,S2). % last
change(S1,2,L,S2):- % P1 is 2
    P2 is 1, % first element
    nth1(2,S1,City1),
    nth1(P2,S1,City2),
    replace_list(S1,2,City2,S11),
    replace_list(S11,P2,City1,S12),
    replace_list(S12,L,City1,S2). % last
change(S1,L,L,S2):- % P1 is L
    P2 is L-1, % last non repeated city
    nth1(L,S1,City1),
    nth1(P2,S1,City2),
    replace_list(S1,L,City2,S11),
    replace_list(S11,P2,City1,S12),
    replace_list(S12,1,City2,S2). % last

% initial solution: lets start with a bad solution:
initial([porto,viana,famalicao,valenca,varzim,braga,barcelos,porto]).

:- set_random(seed(12345)). % set initial random seed

% S is the solution
q1:- % standard hill climbing
    initial(S0), % initial solution
    % 10000 iterations, report every 2000 iterations
    time(hill_climbing(S0,[10000,2000,0,min],S)),
    eval(S,Dist),
    write('sol:'),write(S),nl,write('dist:'),write(Dist),nl.

q2:- % stochastic hill climbing Prob=0.2
    initial(S0), % initial solution
    % 10000 iterations, report every 2000 iterations
    time(hill_climbing(S0,[10000,2000,0.2,min],S)),
    eval(S,Dist),
    write('sol:'),write(S),nl,write('dist:'),write(Dist),nl.

```

Comentário sobre a Resolução:

Este é um problema já muito estudado em otimização, tendo uma complexidade NP, sendo por isso de difícil resolução quando o número de cidades é elevado (Cortez, 2014). Nesta resolução, e uma vez que o enunciado é livre, optou-se por escolher uma instância simples do problema e que corresponde a oito cidades do litoral norte de Portugal. Todas distâncias entre cidades estão descritas no predicado `dist`.

Esta resolução assume uma lista para representar o caminho total percorrido pelo caixeiro viajante, sendo que a lista inicia numa cidade, depois segue com um percurso por todas cidades, terminando na cidade inicial. A função `eval` calcula a distância D_S para cada solução s . Por sua vez, a função `change` escolhe de modo aleatório uma posição da lista e depois opera uma troca da posição das cidades adjacentes a essa posição. Uma vez que a cidade inicial é igual à cidade final, é necessário garantir essa igualdade quando a posição aleatória tem o valor de 1, 2 ou L (comprimento da lista).

Quanto à otimização, escolheu-se uma solução inicial “má”, correspondendo a um percurso longo de 438.9 km, justamente para avaliar como se comportam os métodos de *hill climbing*. Para efeitos de demonstração, e para garantir que se obtém a mesma execução em computadores diferentes, foi definida uma semente inicial de geração de números aleatórios, através do predicado SWI-Prolog `set_random`. Quanto aos métodos de *hill climbing*, compara-se o método clássico com o estocástico (probabilidade de 20%), sendo executadas 10000 iterações de ambos métodos. De seguida, apresenta-se o resultado de uma execução:

```
?- [traveling].
true.

?- q1.
init: S0:[porto,viana,famalicao,valenca,varzim,braga,barcelos,porto]
E0:438.90999999999997
iter:2000
S1:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E1:295.22
S2:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E2:295.22
iter:4000
S1:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E1:295.22
S2:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E2:295.22
iter:6000
S1:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E1:295.22
S2:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E2:295.22
iter:8000
S1:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E1:295.22
```

```

S2:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E2:295.22
iter:10000
S1:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E1:295.22
S2:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E2:295.22
final: S:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E:295.22
best: S:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
E:295.22
% 974,858 inferences, 0.371 CPU in 0.499 seconds (74% CPU, 2625244
Lips)
sol:[viana,varzim,barcelos,famalicao,porto,braga,valenca,viana]
dist:295.22
true .
?- q2.
init: S0:[porto,viana,famalicao,valenca,varzim,braga,barcelos,porto]
E0:438.90999999999997
iter:2000
S1:[valenca,viana,varzim,braga,porto,famalicao,barcelos,valenca]
E1:314.04999999999995
S2:[valenca,viana,varzim,braga,porto,famalicao,barcelos,valenca]
E2:314.04999999999995
iter:4000
S1:[porto,famalicao,barcelos,valenca,viana,varzim,braga,porto]
E1:314.05
S2:[braga,famalicao,barcelos,valenca,viana,varzim,porto,braga]
E2:301.12
iter:6000
S1:[porto,varzim,viana,barcelos,famalicao,braga,valenca,porto]
E1:346.13
S2:[porto,varzim,viana,barcelos,famalicao,braga,valenca,porto]
E2:346.13
iter:8000
S1:[barcelos,valenca,viana,porto,braga,famalicao,varzim,barcelos]
E1:329.45000000000005
S2:[barcelos,valenca,viana,porto,famalicao,braga,varzim,barcelos]
E2:317.0
iter:10000
S1:[valenca,viana,varzim,porto,braga,famalicao,barcelos,valenca]
E1:301.12

```

```
S2:[valenca,viana,varzim,porto,braga,barcelos,famalicao,valenca]
E2:317.58000000000004
final:
S:[valenca,viana,varzim,porto,braga,barcelos,famalicao,valenca]
E:317.58000000000004
best: S:[porto,varzim,barcelos,viana,valenca,braga,famalicao,porto]
E:277.76
% 950,189 inferences, 0.400 CPU in 0.598 seconds (67% CPU, 2375538
Lips)
sol:[porto,varzim,barcelos,viana,valenca,braga,famalicao,porto]
dist:277.76
true .
```

Nesta execução em particular, o método estocástico obteve um melhor resultado (277,76 km) quando comparado com o método puro de *hill climbing* (295,22 km). Realça-se que a comparação de métodos de otimização requiere a realização de um elevado número de execuções, pelo que esta diferença entre o *hill climbing* puro e estocástico não é significativa em termos estatísticos. Ver o livro (Cortez, 2014) para mais detalhes sobre como comparar métodos de otimização.

Referências Bibliográficas

Cortez, P., & Silva, A. (2008). Using data mining to predict secondary school student performance.

Cortez, P. (2014). *Modern optimization with R*. Springer.

Cortez, P. (2015). A tutorial on the rminer R package for data mining tasks. *Teaching Report*, University of Minho, Guimarães, Portugal, <http://hdl.handle.net/1822/36210>.

Bratko, I. (2012). *Programming in Prolog for Artificial intelligence*. Pearson Education, 4th edition, Harlow, England.

Wielemaker, J., Schrijvers, T., Triska, M., & Lager, T. (2012). Swi-prolog. *Theory and Practice of Logic Programming*, 12(1-2), 67-96.

Wielemaker, J., De Koninck, L., Fruehwirth, T., Triska, M., & Uneson, M. (2014). *SWI Prolog Reference Manual 7.1*.