Índice

1. Pesquisa 2
1.1 Estratégias de Pesquisa 2
1.2 Pesquisas ignorantes ou cegas 2
1.2.1 Pesquisa em largura primeiro (Breadth-first search) 2
1.2.2 Pesquisa de custo uniforme (Uniform cost search) 2
1.2.3 Pesquisa em profundidade primeiro (Depth-first search) 3
1.2.4 Pesquisa em profundidade limitada (Depth-limited search) 3
1.2.5 Pesquisa por aprofundamento progressivo (Iterative deepening search) 3
1.2.6 Pesquisa bidireccional (grafos) 3
1.3 Pesquisas informadas 4
1.3.1 Pesquisa do melhor primeiro (Best-first search) 4
1.3.2 Pesquisa sôfrega (Greedy search) 4
1.3.3 Pesquisa A* 4
1.4 Heurísticas 4
1.5 Pesquisas informadas com memória limitada 5
1.5.1 A* por aprofundamento progressivo (Iteractive deepening A* - IDA*) 5
1.5.2 A* de memória limitada simplificado (Simplified memory-bounded A* - SMA*) 5
1.6 Algoritmos de melhoramento iterativo 5
1.6.1 Trepa-colinas (Hill-climbing) 5
1.6.2 Trepa-colinas com recomeços aleatórios (Random-restart hill-climbing) 5
1.6.3 Recristalização simulada (Simulated annealing) 6
1.7 Algoritmos genéticos 6
1.8 Problemas de satisfação de restrições (Constraint restriction problem CSP) 6
2. Jogos 6
2.1 Minimax 6
2.2 Corte α - β 7
2.3 Algoritmo para jogos não deterministas 7
3. Representação de conhecimento 7
3.1 Exemplos 7
3.1.1 Nqueens (1ª versão) 7
3.1.2 Nqueens (2ª versão) 8
3.1.3 Nqueens (3ª versão) 8
3.1.4 Estabelecer ordens lineares 8
4. Planeamento 10
4.1 Trabalhos 10
4.1.1 Sapatos 10
4.1.2 Viagem 10
4.1.3 Blocos (Sussman) 10
4.1.4 Compras 10
4.1.5 Robot 11
5. Redes de Bayes 12
5.1 Probabilidade condicional 12
5.2 Regra de Bayes 12
6. Exames 13
6.1 Exame 1ª chamada 2003 13
Grupo 1 (Pesquisa) 13
Grupo 2 (Planeamento) 14
Grupo 3 (Redes de Bayes) 15
Grupo 4 (Representação do conhecimento) 17
6.2 Exame 2ª chamada 2003 18
Grupo 1 (Jogos de 2 oponentes) 18
Grupo 3 (Redes de Bayes) 18
6.3 Exame recurso 2003 20
Grupo 1 (Pesquisa) 20
Grupo 2 (Redes de Bayes) 21

1. Pesquisa

1.1 Estratégias de Pesquisa

Uma estratégia é definida pela ordem de expansão dos nós. As estratégias são avaliadas de acordo com as seguintes dimensões:

- Completude: encontra uma solução caso existam soluções?
- Complexidade temporal: numero de nós gerados/expandidos
- Complexidade espacial: numero máximo de nós em memória
- Optimalidade: encontra sempre uma solução de custo mínimo?

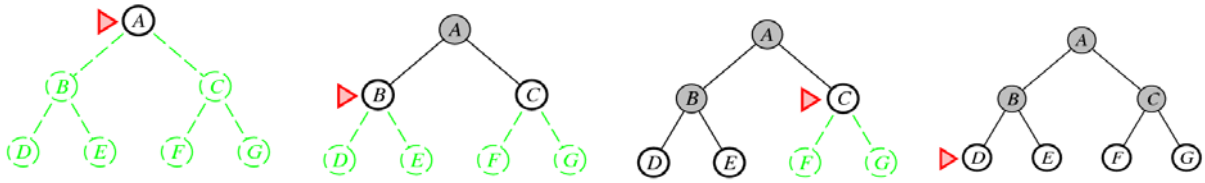
A complexidade temporal e espacial são avaliadas em termos de:

- b: factor de ramificação máximo da árvore de pesquisa (branching)
- d: profundidade da solução de custo mínimo (depth)
- m: profundidade máxima do espaço de estados
- L: limite forçado de profundidade

1.2 Pesquisas ignorantes ou cegas

1.2.1 Pesquisa em largura primeiro (Breadth-first search)

A raiz é expandida primeiro, depois todos os nós gerados pela raiz são expandidos e depois os seus sucessores. A implementação pode ser feita com uma fila de espera em que os novos sucessores são colocados no fim.



- **Completa:** Sim. Se houver um solução é garantido que esta pesquisa a encontra.
- **Complexidade temporal**: $O(b^d)$
- **Complexidade espacial**: $O(b^d)$ (mantém todos os nós em memória)
- **Optimalidade:** Sim, se o custo de um passo for constante.

O espaço é o problema essencial.

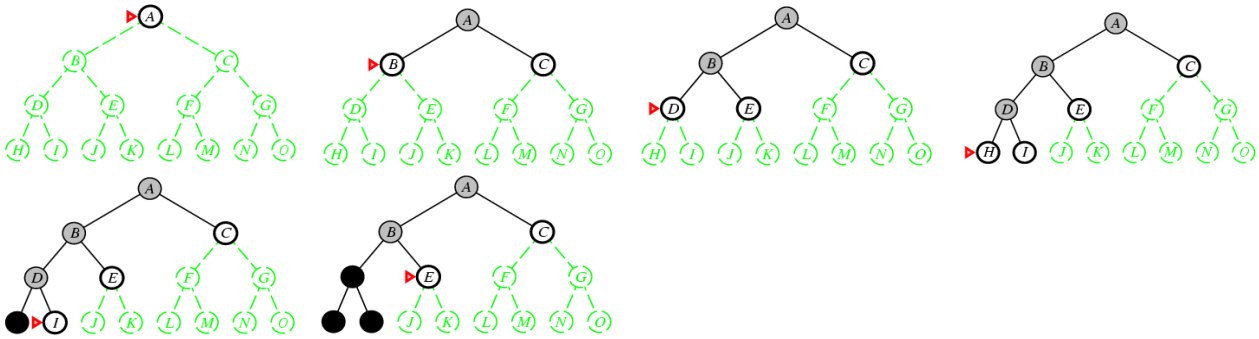
1.2.2 Pesquisa de custo uniforme (Uniform cost search)

É uma modificação da pesquisa em largura, em que se expande o nó de menor custo *g(n)*.

- **Completa:** Sim
- **Complexidade temporal**: número de nós com custo menor que o custo da solução óptima. $O(b^d)$
- **Complexidade espacial**: $O(b^d)$
- **Optimalidade:** Sim.

1.2.3 Pesquisa em profundidade primeiro (Depth-first search)

Expande sempre os nós com profundidade maior. A implementação pode ser feita com uma fila de espera em que os novos sucessores são colocados no início.



- **Completa:** Falha em espaços de profundidade infinita e em espaços com ciclos. Modificando para evitar estados repetidos no mesmo caminho é completo para espaços finitos.
- **Complexidade temporal**: $O(b^m)$ muito mau se m é muito maior que d
- **Complexidade espacial**: O(bm) linear
- **Optimalidade:** Não.

Para problemas que têm muitas soluções este algoritmo pode ser mais rápido que a pesquisa em largura porque tem boas hipóteses de encontrar uma solução depois de explorar uma pequena parte do espaço.
A desvantagem é que o algoritmo de pesquisa em profundidade pode ficar preso ao tomar um caminho errado e neste caso poderá nunca retornar ou retornar uma solução que não é a melhor.

1.2.4 Pesquisa em profundidade limitada (Depth-limited search)

Evita os problemas da pesquisa em profundidade ao impor uma restrição (L) na profundidade máxima de pesquisa.

- **Completa:** Sim. Encontra uma solução se esta existir (i.e. se L>=d).
- **Complexidade temporal**: $O(b^L)$
- **Complexidade espacial**: O(bL) linear
- **Optimalidade:** Não existem garantias que se encontre a melhor solução. Se o limite máximo de pesquisa for demasiado pequeno pode nem ser completo.

1.2.5 Pesquisa por aprofundamento progressivo (Iterative deepening search)

Evita o problema de escolher a profundidade correcta na pesquisa em profundidade limitada ao tentar todas as profundidades possíveis.

- **Completa:** Sim.
- **Complexidade temporal**: $O(b^d)$
- **Complexidade espacial**: O(bd) linear
- **Optimalidade:** Sim.

1.2.6 Pesquisa bidireccional (grafos)

Pesquisa simultaneamente para a frente desde o estado inicial e para trás desde o objectivo. Só é aplicável se for possível/viável definir o predecessor de um nó. Pode reduzir muito a complexidade temporal mas nem sempre é aplicável.

- **Completa:** Sim.
- **Complexidade temporal**: $O(b^{d/2})$

- **Complexidade espacial**: $O(b^{d/2})$
- **Optimalidade:** Sim.

1.3 Pesquisas informadas

1.3.1 Pesquisa do melhor primeiro (Best-first search)

A ideia é aplicar uma função de avaliação *h(n)* (heurística) a cada nó que indica se o nó é promissor ou não e expandir o nó mais promissor. A implementação pode ser feita com uma fila de espera em que os nós mais promissores são colocados no início.

1.3.2 Pesquisa sôfrega (Greedy search)

Usa h(n) = estimativa do custo do menor caminho até ao objectivo. Por exemplo $h_{SLD}(n)$ = distância em linha recta até Bucareste

- **Completa:** Não. Pode ficar em círculos.
- **Complexidade temporal**: $O(b^d)$ Uma boa heurística pode ter melhorias espectaculares.
- **Complexidade espacial**: $O(b^d)$
- **Optimalidade:** Não.

1.3.3 Pesquisa A*

Combina a pesquisa sôfrega com a pesquisa de custo uniforme, combinando as duas funções de avaliação: *f(n) = g(n) + h(n)* onde:

- g(n) = custo actual para atingir o nó *n.*
- h(n) = custo estimado para atingir o objectivo através de *n*
- f(n) = custo estimado do caminho até ao objectivo passando por *n*.

A pesquisa A* utiliza uma heurística admissível (i.e. h(n)<=h*(n), com h*(n)=custo verdadeiro a partir de n)

- **Completa:** Sim, a não ser que haja um numero infinito de nós com f<=f(G).
- **Complexidade temporal**: Na maior parte dos casos tem um crescimento exponencial a não ser que o erro na função heurística não cresça mais depressa que o custo actual, ou seja, **|h(n) - h*(n)|<=O(log h*(n)).**
- **Complexidade espacial**: Mantém todos os nós em memória.
- **Optimalidade:** Sim.

1.4 Heurísticas

Sejam

$h_1(n)$=numero de peças na posição errada

$h_2(n)$=soma das distâncias de Manhattan

duas heurísticas. Se $h_2(n) >= h_1(n)$ para todo o *n*, então h_2 **domina** h_1 e é melhor para a pesquisa. Caso h_2 não domine h_1 e vice versa, pode-se usar $max\{h_2, h_1\}$.

As heurísticas admissíveis podem ser derivadas a partir do custo **exacto** de uma solução para uma versão **relaxada** do problema.

1.5 Pesquisas informadas com memória limitada

1.5.1 A* por aprofundamento progressivo (Iteractive deepening A* - IDA*)

É a utilização do aprofundamento progressivo aplicado ao A* só que em vez de limitar a profundidade limita-se o custo *f*.

- **Completa:** Sim.
- **Complexidade temporal**:
- **Complexidade espacial**: Mantém todos os nós em memória.
- **Optimalidade:** Sim.

1.5.2 A* de memória limitada simplificado (Simplified memory-bounded A* - SMA*)

Tal como o A* expande o melhor folha até ficar com a memória cheia. Quando a memória fica toda ocupada, esquece a folha mais antiga com o pior valor e guarda no pai o seu valor para possível regeneração.

- Utiliza toda a memória disponível.
- É completo se existir uma solução cujo caminho caiba em memória.
- É óptimo se existir uma solução óptima cujo caminho caiba em memória, caso contrário devolve a melhor solução cujo caminho cabe em memória.

1.6 Algoritmos de melhoramento iterativo

Em muitos problemas de optimização, o caminho para a solução é irrelevante; o estado objectivo é a própria solução (problema das 8 Rainhas ou problema do Caixeiro Viajante). Nesta situação podem-se usar algoritmos de melhoramento iterativo; mantém-se um único estado corrente e tenta-se altera-lo para melhorar a sua qualidade.

Considera-se que os estados estão dispostos numa superfície e que a altura de um ponto corresponde à função de avaliação.

1.6.1 Trepa-colinas (Hill-climbing)

Este algoritmo move-se continuamente na direcção de um valor mais alto. Quando existe mais do que um sucessor com o mesmo a escolha é feita aleatoriamente.

Tem 3 problemas conhecidos:

- Pode ficar preso facilmente em máximos locais;
- Pode ficar a passear aleatoriamente em planaltos;
- Travessia difícil em cristas de máximos locais.

Em espaços contínuos também existe o problema na escolha do tamanho do passo:

- Muito grande pode saltar máximos;
- Muito pequeno pode ter uma convergência muito lenta.

1.6.2 Trepa-colinas com recomeços aleatórios (Random-restart hill-climbing)

Para evitar ficar preso em máximos locais, tenta novamente com um novo estado inicial gerado aleatoriamente e guarda-se o melhor deles ao fim de um determinado numero de iterações.
O sucesso deste algoritmo depende muito da paisagem do espaço de estados: poucos máximos locais e planaltos é a situação ideal.

1.6.3 Recristalização simulada (Simulated annealing)

Para fugir de máximos locais, este algoritmo permite alguns movimentos piores mas decrescendo gradualmente o seu tamanho e a sua frequência.

- Selecciona-se um movimento aleatório a cada iteração
- Se o estado for melhor, transita-se sempre
- Se o estado for pior só transita de acordo uma probabilidade que decresce exponencialmente $e^{\Delta E/T}$

1.7 Algoritmos genéticos

Começa com uma população de indivíduos (cromossomas) gerados aleatoriamente. Faz-se evoluir em simultâneo a população (soluções alternativas) por intermédio de operadores de selecção, reprodução e mutação.

- A população é constituída por um numero geralmente fixo de indivíduos.
- Cada cromossoma (ou indivíduo) é representado por uma cadeia de símbolos de um alfabeto, normalmente uma cadeia de 0's e 1's.
- A qualidade de cada indivíduo é avaliada por uma função de avaliação (fitness function) que influencia a probabilidade de selecção dos indivíduos para reprodução.
- A reprodução combina dois indivíduos para gerar novo(s) indivíduo(s)
- Os filhos podem sofrer mutações genéticas aleatórias no seu cromossoma antes de serem colocados na população (geração seguinte)
- Uma geração pode ser substituída integralmente pelos seus descendentes ou manter alguns dos seus indivíduos mais aptos.

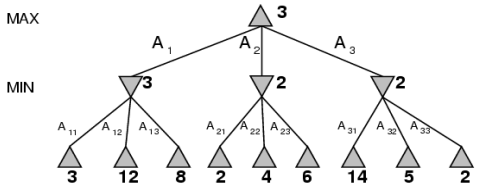
1.8 Problemas de satisfação de restrições (Constraint restriction problem CSP)

É um tipo de problema onde um estado é definido com variáveis X_i que tomam valores num domínio D_i. O teste objectivo é um conjunto de restrições especificando as combinações permitidas para subconjuntos de variáveis.

2. Jogos

2.1 Minimax

O número em cada nó representa o valor da **função de utilidade** que dá um valor numérico à solução de um jogo. Este algoritmo está desenhado de modo a determinar a melhor estratégia para Max.



O algoritmo usa a pesquisa em profundidade e assume que o programa tem tempo suficiente para pesquisar a árvore inteira. Para jogos reais é impraticável mas serve como base para métodos mais realistas.

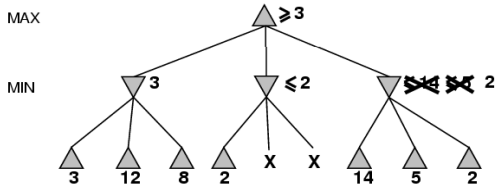
- **Completo:** Só se a árvore for finita.
- **Complexidade temporal**: $O(b^m)$
- **Complexidade espacial**: O(bm) exploração pelo melhor primeiro
- **Optimalidade:** Sim contra um adversário perfeito. Caso contrário …

2.2 Corte α - β

Em vez de usar uma função de utilidade, a ideia é limitar a profundidade e usar uma **função de avaliação** para dar uma estimativa razoável da posição.

α é o melhor valor para Max (o mais alto) encontrado até ao momento; β é o melhor valor para Min (o mais baixo) encontrado até ao momento.

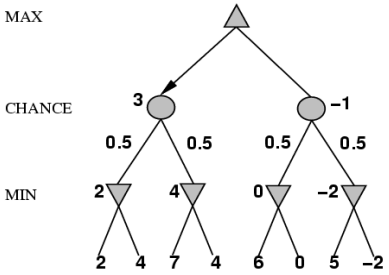
O teste alfa-beta procura novos valores para α e β à medida que vai avançando e corta a árvore (i.e. termina a chamada recursiva) assim que encontra valores piores que α e β.



O corte não afecta o resultado final. Uma boa ordenação dos movimentos melhora o efeito de corte. Com "ordenação perfeita" a complexidade temporal é $O(b^{m/2})$ duplica a profundidade da pesquisa.

2.3 Algoritmo para jogos não deterministas

Factor sorte introduzido pelos dados ou pelo baralhar das cartas. Os nós de sorte são representados por círculos e têm é o valor esperado.



3. Representação de conhecimento

3.1 Exemplos

3.1.1 Nqueens (1ª versão)

```
const size = 8.

col(1..size).
row(1..size).
queen(1..size).


at(I,X,Y) :- queen(I), row(X), col(Y), not not_at(I,X,Y).
not_at(I,X,Y) :- queen(I), row(X), col(Y), not at(I,X,Y).

:-  queen(I), row(X), col(Y), row(XX), col(YY), at(I,X,Y), at(I,XX,YY), not eq(X,XX).
:-  queen(I), row(X), col(Y), row(XX), col(YY), at(I,X,Y), at(I,XX,YY), not eq(Y,YY).

placed(I) :- queen(I), row(X), col(Y), at(I,X,Y).
:- queen(I), not placed(I).

:-  queen(I), row(X), col(Y), col(YY), queen(J), at(I,X,Y), at(J,X,YY), not eq(I,J).
:-  queen(I), row(X), col(Y), row(XX), queen(J), at(I,X,Y), at(J,XX,Y), not eq(I,J).


:-  queen(I), queen(J), row(X), col(Y), row(XX), col(YY), at(I,X,Y), at(J,XX,YY), not eq(I,J),
    eq(abs(X-XX),abs(Y-YY)).

hide not_at(I,X,Y).
hide placed(I).
hide col(X).
```

```
hide row(X).
hide queen(X).
```

3.1.2 Nqueens (2ª versão)

```
const size = 8.

col(1..size).
row(1..size).

not_in(X,Y) :- row(X), col(Y), not in(X,Y).

in(X,Y) :- row(X), col(Y), not not_in(X,Y).

has_queen(X) :- row(X), col(Y), in(X,Y).

:-  row(X), not has_queen(X).

:-  row(X), col(Y), row(XX), not eq(X,XX), in(X,Y), in(XX,Y).

:-  row(X), col(Y), col(YY), not eq(Y,YY), in(X,Y), in(X,YY).

:- row(X), col(Y), row(XX), col(YY), not eq(X,XX), not eq(Y,YY), in(X,Y), in(XX,YY),
eq(abs(X-XX), abs(Y-YY)).

hide not_in(X,Y).
hide row(X).
hide col(Y).
hide has_queen(X).
```

3.1.3 Nqueens (3ª versão)

```
const size = 8.

col(1..size).
row(1..size).

not_in(X,Y) :- row(X), col(Y), col(YY), not eq(Y,YY), in(X,YY).

not_in(X,Y) :- row(X), col(Y), row(XX), not eq(X,XX), in(XX,Y).

in(X,Y) :- row(X), col(Y), not not_in(X,Y).

:-  row(X), col(Y), row(XX), col(YY), not eq(X,XX),  not eq(Y,YY), in(X,Y), in(XX,YY), eq(abs(X-XX),abs(Y-
YY)).

hide not_in(X,Y).
hide row(X).
hide col(Y).
```

3.1.4 Estabelecer ordens lineares

```
Dado um conjunto de objectos, estabelecer uma ordem linear entre eles. Cada conjunto de
resposta corresponde a uma ordem linear, e cada possível ordem linear corresponde a um
objecto.
Para tal, primeiro define-se uma ordem prec. Depois, testa-se se essa ordem é linear tentando
definir o primeiro (first), último (last) e o sucessor (succ) com respeito à relação prec.
Depois é só verificar se:
- existe o primeiro
- existe o último
- o último é atingível a partir do primeiro através do operador succ;
Se sim, então a ordem é linear. Caso contrário, a ordem não é linear e o conjunto de resposta
é descartado.

obj(a).
obj(b).
obj(c).
obj(d).

prec(X,Y) :- obj(X), obj(Y), not prec(Y,X),  neq(X,Y).
prec(X,Y) :- obj(X), obj(Y), obj(Z), prec(X,Z), prec(Z,Y).
%It works even if the previous rule is removed.
```

```
not_succ(X,Z) :- obj(X), obj(Z), prec(Z,X).
not_succ(X,Z) :- obj(X), obj(Y), obj(Z), prec(X,Y), prec(Y,Z).
not_succ(X,Y) :- obj(X), obj(Y), eq(X,Y).
succ(X,Y) :- obj(X), obj(Y), not  not_succ(X,Y).

not_first(X) :- obj(X), obj(Y), prec(Y,X).
first(X) :- obj(X), not not_first(X).

not_last(X) :-  obj(X), obj(Y), prec(X,Y).
last(X) :- obj(X), not not_last(X).

reachable(X) :- obj(X), first(X).
reachable(Y) :- obj(X), obj(Y), reachable(X), succ(X,Y).

linear :- obj(X), reachable(X), last(X).
inconsistent :- not inconsistent, not linear.

hide obj(X).
hide prec(X,Y).
hide not_succ(X,Y).
hide not_first(X).
hide not_last(X).
hide reachable(X).
hide linear.
hide inconsistent.
```

4. Planeamento

4.1 Trabalhos

4.1.1 Sapatos

```
accao       calcarSap(Pe)
condicoes   [meia(Pe)]
efeitos     [sapato(Pe)]
restricoes  [].

accao       calcarMeia(Pe)
condicoes   []
efeitos     [meia(Pe)]
restricoes  [].

inicial     [].

objectivos [sapato(esq),sapato(dir)].
```

4.1.2 Viagem

```
accao          go(X,Y)
condicoes      [at(X),road(X,Y)]
efeitos        [at(Y),-at(X)]
restricoes     [(X\==Y)].


inicial [at(lisboa),road(lisboa,evora),road(evora,faro),
         road(lisboa,coimbra),road(coimbra,porto),road(evora,coimbra)].

objectivos [at(porto)].
```

4.1.3 Blocos (Sussman)

```
accao       putTable(X)
condicoes   [on(X,Z),clear(X)]
efeitos     [clear(Z),on(X,mesa),-on(X,Z)]
restricoes  [(X\==Z)].

accao       putOn(X,Y)
condicoes   [on(X,Z),clear(X),clear(Y)]
efeitos     [clear(Z),on(X,Y),-on(X,Z),-clear(Y)]
restricoes  [(Y\==mesa),(X\==Y),(X\==Z),(Y\==Z)].


inicial [clear(c),on(c,a),on(a,mesa),clear(b),on(b,mesa)].

objectivos [on(a,b),on(b,c)].
```

4.1.4 Compras

```
accao         go(X,Y)
condicoes     [at(X)]
efeitos       [at(Y),-at(X)]
restricoes    [(X\==Y)].

accao         buy(X)
condicoes     [sell(L,X),at(L)]
efeitos       [have(X)]
restricoes    [].

inicial        [at(home),sell(super,banana),sell(hws,drill),sell(super,milk)].

Objectivos     [have(milk),have(drill),have(banana),at(home),at(super)].
```

4.1.5 Robot

Um robot desloca-se num mundo com 3 salas (com nomes s1, s2 e s3). Existe uma porta entre a sala s1 e a s2 e outra entre a sala s2 e s3. Quer isto dizer que, para o robot por exemplo ir da sala s1 para s3, tem que primeiro ir à sala s2, e só depois ir para s3. O robot tem uma garra com que pode agarrar um objecto de cada vez. Para agarrar um dado objecto b, o robot para além de não poder ter outro na garra, tem que estar na sala onde b se encontra. Quando o robot larga um objecto, o objecto fica na sala 1 o robot está esse momento. Inicialmente o robot está na sala s1, há o objecto b1 em s1, o b2 em s2 e o b3 em s3. Pretende-se que todos os objectos sejam colocados na sala s3.

```
accao          go(X,Y)
condicoes      [robAt(X),porta(X,Y)]
efeitos        [robAt(Y),-robAt(X)]
restricoes     [(X\==Y)].

accao          grab(O,R)
condicoes      [empty,objAt(O,R),robAt(R)]
efeitos        [has(O),-objAt(O,R),-empty]
restricoes     [].

accao          drop(O,R)
condicoes      [has(O),robAt(R)]
efeitos        [objAt(O,R),empty,-has(O)]
restricoes     [].

inicial        [robAt(s1), empty,objAt(b1,s1), objAt(b2,s2), objAt(b3,s3), porta(s1,s2), porta(s2,s3),
               porta(s2,s1), porta(s3,s2)].

Objectivos     [objAt(b1,s3),objAt(b2,s3),objAt(b3,s3)].
```

5. Redes de Bayes

Probabilidades **a priori** ou **probabilidades não condicionais** de proposições
e.g., P(Cavity =true) = 0.1 e P(Weather =sunny) = 0.72
corresponde à crença antes da chegada de (nova) evidência.

Distribuição de probabilidade dá valores a todas as atribuições possíveis:
P(Weather) = {0.72; 0.1; 0.08; 0.1} (normalizado, i.e., soma 1)

Distribuição de probabilidade conjunta para um conjunto de v.a.s. dá-nos a probabilidade de cada acontecimento atómico nessas v.a.s. (i.e., em cada ponto amostra)
P(Weather,Cavity) = uma matriz de 4 x 2 valores:

Weather	sunny	rain	cloudy	snow
Cavity=true	0.144	0.02	0.016	0.02
Cavity=false	0.576	0.08	0.064	0.08

Probabilidades a posteriori ou condicionais e.g., *P*(*cavity* | *toothache*) = 0.8 i.e., *toothache* é tudo o que sei

5.1 Probabilidade condicional

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} \text{ if } P(b) \neq 0$$

Regra do produto: é uma versão alternativa

$$P(a \wedge b) = P(a|b)P(b) = p(b|a)P(a)$$

Regra da cadeia: é derivada por aplicação sucessiva da regra do produto

$$\begin{aligned}\mathbf{P}(X_1, \ldots, X_n) &= \mathbf{P}(X_1, \ldots, X_{n-1})\ \mathbf{P}(X_n|X_1, \ldots, X_{n-1}) \\ &= \mathbf{P}(X_1, \ldots, X_{n-2})\ \mathbf{P}(X_{n-1}|X_1, \ldots, X_{n-2})\ \mathbf{P}(X_n|X_1, \ldots, X_{n-1}) \\ &= \ldots \\ &= \textstyle\prod_{i=1}^{n} \mathbf{P}(X_i|X_1, \ldots, X_{i-1})\end{aligned}$$

5.2 Regra de Bayes

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

Útil para obter a probabilidade do **diagnóstico** dada a probabilidade **causal**.

$$P(Causa|Efeito) = \frac{P(Efeito|Causa)P(Causa)}{P(Efeito)}$$

6. Exames

6.1 Exame 1ª chamada 2003

Grupo 1 (Pesquisa)

Um agricultor está na margem dum rio com uma raposa, uma galinha e um saco de milho, e quer levar tudo para a outra margem. Para tal dispõe de um barco pequeno onde só pode levar um (ou nenhum) dos itens de cada vez. Além disso, para garantir que não perde nada durante o processo, não pode dar-se ao luxo de deixar a raposa com a galinha numa das margens enquanto ele está na outra margem (se o fizer, arrisca-se a que a raposa coma a galinha). O mesmo se passa relativamente à galinha e ao milho. O que se pretende é saber que viagens é que o agricultor terá que fazer no barco, e transportando o quê em cada uma delas, por forma a no final ter a raposa, a galinha e o milho sãos e salvos na outra margem. Além disso pretende-se que tudo seja feito com o menor número de viagens possível.

1 a) Diga como formalizaria o problema por forma a que este pudesse ser resolvido por uma algoritmo normal de pesquisa em espaço de estados. Ou seja, diga como definiria estados, o estado inicial, a função de teste para estados objectivo e a função que devolve os sucessores dum estado.

```
Um estado é representado por e(R, G, M, B), com R = raposa, G = galinha, M = milho e B = barco
a tomar valores 1 –margem esquerda e 0 – margem direita.

e_inicial(1,1,1,1).

e_objectivo(0,0,0,0).

//passar o barco para a margem direita
sucessor(e(1,G,M,1), op(leva_raposa), e(0,G,M,0)):- \+e_invalido(e(0,G,M,0)).
sucessor(e(R,1,M,1), op(leva_galinha), e(_,0,_,0)) :- \+e_invalido(e(0,G,M,0)).
sucessor(e(R,G,1,1), op(leva_milho), e(_,_,0,0)) :- \+e_invalido(e(0,G,M,0)).
sucessor(e(R,G,M,1), op(leva_nada), e(_,_,_,0)) :- \+e_invalido(e(0,G,M,0)).

//passar o barco para a margem esquerda
sucessor(e(0,G,M,0), op(leva_raposa), e(1,_,_,1)) :- \+e_invalido(e(1,G,M,1)).
sucessor(e(R,0,M,0), op(leva_galinha), e(_,1,_,1)).
sucessor(e(R,G,0,0), op(leva_milho), e(_,_,1,1)).
sucessor(e(R,G,M,0), op(leva_nada), e(_,_,_,1)).

//Um estado é invalido se:
e_invalido(e(R,G,M,B)) :- G==R, B\=G, M\=G. //a galinha e a raposa estão sozinhas
e_invalido(e(R,G,M,B)) :- G==M, B\=G, R\=G. //a galinha e  milho estão sozinhas
```

1 b) Uma pesquisa em profundidade seria completa para a resolução deste problema? Justifique, gizando um exemplo concreto de evolução deste algoritmo neste problema que mostre da completude ou não.

```
Uma pesquisa em profundidade não seria completa devido aos ciclos. Começando no estado
inicial, nada impede do barco andar vazio para a margem direita, voltar novamente vazio e
recomeçar tudo de novo indefinidamente sem nunca achar uma solução.
```

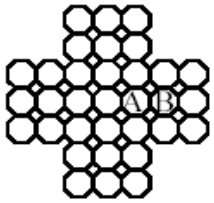
1 c) Defina uma heurística admissível (não trivial) para que este problema possa ser melhor resolvido usando o algoritmo A*. Justifique que de facto a heurística que apresenta é admissível.

```
Uma heurística é admissível se h(n)<=h*(n), com h*(n)=custo verdadeiro a partir do nó n).
Neste caso estamos interessados no menor número de viagens possível, logo uma heurística
poderia ser o numero de itens no lado esquerdo da margem.
```

1 d) Suponha agora uma variante deste problema em que, em vez de se pretender um número mínimo de viagens, se pretende saber a solução com um número de viagens fixo à partida (por exemplo 7 viagens). Se quisesse resolver este novo problema com um algoritmo iterativo de melhoramento (tipo "hill climbing"), como definiria *solução*, *vizinhança* e a *função de avaliação* do custo de cada solução.

```
Vamos representar uma solução como uma lista [O1, O2, O3,...,O7] em que cada elemento obj(O,M)
é o objecto  O (raposa, galinha, milho, nada) transportado para a margem M (esq, dir). A lista
terá 7 elementos.
A vizinhança é definida como a alteração de um elemento na lista.
A função de avaliação será o numero de objectos na margens esquerda (inicial) ou uma numero
muito grande para uma solução inválida.
```

Grupo 2 (Planeamento)



Certamente que conhece o jogo do solitário. Neste jogo existe um tabuleiro, semelhante ao que se mostra na figura, e onde inicialmente todas as casas do tabuleiro, com excepção da do meio, têm uma pedra. O objectivo é chegar a uma configuração onde só exista uma pedra no centro do tabuleiro. Para retirar uma peça do tabuleiro há que fazer um movimento em que se desloca uma peça adjacente àquela que se quer retirar para um posição livre do tabuleiro, também adjacente à peça a retirar, e em linha com a peça que se está a mover. No fundo, isto é semelhante à forma como se comem peças num jogo de damas, só que aqui também se podem comer peças com movimentos na horizontal, vertical e quer para a frente quer para trás (ao invés das damas em que só se podem comer peças na diagonal, para a frente). Na figura, para retirar uma peça que esteja na posição A, pode deslocar-se uma peça em B para o centro do tabuleiro. Pretende resolver-se o jogo do solitário usando um algoritmo de planeamento de ordens parciais.

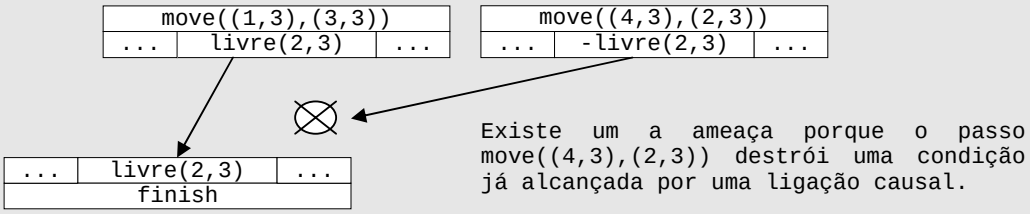
2 a) Comece por descrever as acções de *start* e *finish* com que o algoritmo do POP começaria para resolver este problema. Para tal vai ter que se usar um conjunto de predicados que descrevam um estado do problema. Sugere-se o uso dos predicados `livre(Posição)` e `com_peca(Posição)` para a descrição de estados.

```
// com_peca(0,0) é o canto superior esquerdo
start [com_peca(2,0), com_peca(3,0), com_peca(4,0),
       com_peca(2,1), com_peca(3,1), com_peca(4,1),
com_peca(0,2),com_peca(1,2),com_peca(2,2),com_peca(3,2),com_peca(4,2),com_peca(3,2), com_peca(4,2),
com_peca(0,3),com_peca(1,3),com_peca(2,3), livre(3,3), com_peca(4,3),com_peca(3,3), com_peca(4,3),
com_peca(0,4),com_peca(1,4),com_peca(2,4),com_peca(3,4),com_peca(4,4),com_peca(3,4), com_peca(4,4),
       com_peca(2,5), com_peca(3,5), com_peca(4,5),
       com_peca(2,6), com_peca(3,6), com_peca(4,6)].

finish [livre(2,0), livre(3,0), livre(4,0),
       livre(2,1), livre(3,1), livre(4,1),
livre(0,2),livre(1,2),livre(2,2),livre(3,2),livre(4,2),livre(3,2), livre(4,2),
livre(0,3),livre(1,3),livre(2,3), com_peca (3,3), livre(4,3),livre(3,3), livre(4,3),
livre(0,4),livre(1,4),livre(2,4),livre(3,4),livre(4,4),livre(3,4), livre(4,4),
       livre(2,5), livre(3,5), livre(4,5),
       livre(2,6), livre(3,6), livre(4,6)].
```

2 b) Apresente um operador STRIPS que descreva um movimento no jogo do solitário. Assuma que está definido um predicado `em_linha(P1,P2,P3)` que é verdade sempre que as posições P1, P2 e P3 estão em linha e que P1 é adjacente de P2 e P2 é adjacente de P3.

```
accao       move (P1,P3)
condicoes   [em_linha(P1,P2,P3), com_peca(P1), com_peca(P2), livre(P3)]
efeitos     [-com_peca(P1), livre(P1), - com_peca(P2), livre(P2), -livre(P3), com_peca(P3)]
restricoes  [(P1\==P2), (P1\==P3), (P2\==P3)]. // não é necessário se em_linha já verificar
```

2 c) Descreva como é que pode ocorrer uma situação em que, na aplicação do algoritmo do POP a este problema, um passo ameace um link já existente no plano parcial que se está a construir. Dê um exemplo concreto duma tal ameaça.



2 d) Descreva brevemente, por palavras suas, como é que o algoritmo do POP evoluiria para a resolução deste problema, por oposição à forma como evoluiria um algoritmo de pesquisa em profundidade.

Parece-lhe uma boa ideia resolver este problema usando um planejador de ordens parciais? Será que se ganha alguma coisa na complexidade de resolução deste problema pelo facto do resultado pretendido ser uma ordem parcial e não uma ordem total, como se obteria por exemplo usando um algoritmo de pesquisa? Justifique.

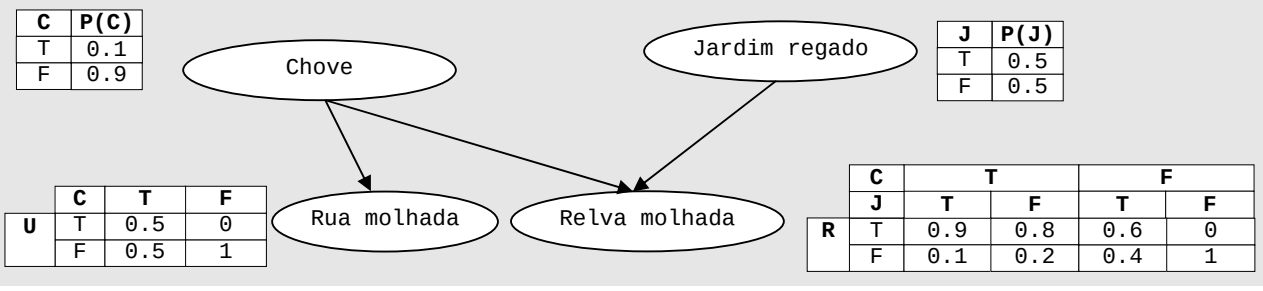
Grupo 3 (Redes de Bayes)

Um jardim perto de minha casa é regado de noite com alguma frequência. Diria que é regado em cerca de 50% das noites. Mesmo quando sei que o jardim foi regado à noite, verifico que nem sempre a relva do jardim está molhada de manhã. Além disso, por vezes chove durante a noite (considere-se que 10% das noites chove), o que também influencia o facto da relva estar ou não molhada pela manhã. Fazendo uma análise à posteriori, verifica-se que se chover então se o jardim for regado durante a noite 90% das vezes a relva está molhada pela manhã, e se não for regado só 80% das vezes é que está molhado. Se não chover então, mesmo havendo rega à noite, só 60% das vezes é que a relva está molhada (talvez porque quando não chove está em regra mais calor, fazendo com que mais vezes a relva entretanto seque). Se não chover e não houver rega, então a relva está garantidamente seca. Além disso, quando chove não é só a relva que pode ficar molhada. De facto, em 50% das manhãs seguintes a uma noite em que choveu, o chão da rua também está molhado. Sempre que não chove durante a noite o chão está seco na manhã seguinte.

3 a) Diga que variáveis aleatórias escolheria para representar o conhecimento acima e, para cada uma delas, qual o seu domínio.

```
J – Jardim regado
R – Relva molhada
C - Chove
U – Rua molhada
```

3 b) Apresente uma rede Bayesiana (topologia e tabelas nos nós) que represente esse conhecimento



3 c) Num dado dia, ia a sair de casa e, antes mesmo de olhar para o chão da rua, verifiquei que a relva estava molhada. Calcule:

3 c1) a probabilidade de ter chovido durante essa noite.

$$P(C = true \mid R = true) = \alpha P(C = true \wedge R = true)$$
$$= \alpha P(C) \sum_{j=\{true,\, false\}} P(R = true \mid C \wedge J = j) P(J)$$

$$C = true$$
$$P(C = true) \sum_{j=\{true,\, false\}} P(R = true \mid C = true \wedge J = j) P(J = j)$$
$$= 0.1 \times (0.9 \times 0.5 + 0.8 \times 0.5) = 0.085$$

$$C = false$$
$$P(C = false) \sum_{j=\{true,\, false\}} P(R = true \mid C = false \wedge J = j) P(J = j)$$
$$= 0.9 \times (0.6 \times 0.5 + 0 \times 0.5) = 0.27$$

$$\alpha = 0.085 + 0.27 = 0.355$$
$$P(C \mid R = true) = \left\langle \frac{0.085}{0.355}, \frac{0.27}{0.355} \right\rangle = \langle 0.239, 0.76 \rangle$$
$$P(C = true \mid R = true) = 0.239$$

3 c2) a probabilidade de, ao depois olhar para o chão da rua, vir a constatar que este também está molhado.

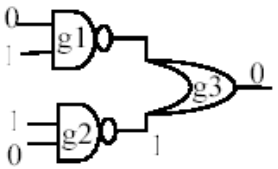
$$P(C = true \mid R = true \wedge U = true) = \alpha P(C = true \wedge R = true \wedge U = true)$$
$$= \alpha P(C = true) \sum_{u=\{true,\, false\}} P(U = u \mid C = true) \sum_{j=\{true,\, false\}} P(R = true \mid C \wedge J = j) P(J)$$

$$C = true$$
$$P(C = true) \sum_{j=\{true,\, false\}} P(R = true \mid C = true \wedge J = j) P(J = j)$$
$$= 0.1 \times (0.9 \times 0.5 + 0.8 \times 0.5) = 0.085$$

$$C = false$$
$$P(C = false) \sum_{j=\{true,\, false\}} P(R = true \mid C = false \wedge J = j) P(J = j)$$
$$= 0.9 \times (0.6 \times 0.5 + 0 \times 0.5) = 0.27$$

$$\alpha = 0.085 + 0.27 = 0.355$$
$$P(C \mid R = true) = \left\langle \frac{0.085}{0.355}, \frac{0.27}{0.355} \right\rangle = \langle 0.239, 0.76 \rangle$$
$$P(C = true \mid R = true) = 0.239$$

Grupo 4 (Representação do conhecimento)



Pretende-se implementar um sistema de diagnóstico de circuitos digitais usando *Answer-Set Programming.* Para simplificar a tarefa, vamos apenas considerar circuitos com dois tipos de gates (nand's e or's de 2 entradas) e assumir que nunca há avarias nas ligações entre as gates. Assume-se ainda que sempre que uma gate está avariada tem o valor 0 (zero) na sua saída.

A ideia é fazer uma modelação genérica de problemas de diagnóstico destes circuitos de tal forma que, se a ela adicionarmos uma representação da topologia do circuito e factos sobre observações feitas nalguns dos pontos do mesmo, haja um conjunto de resposta por cada diagnóstico possível do circuito em causa contendo átomos da forma `avariado(gate)` por cada gate avariada de acordo com esse diagnóstico. No exemplo da figura, terá que haver um conjunto de resposta com **`{avariado(g3)}`** e outro com **`{avariado(g1), avariado(g3)}`**. Para isso há que escolher uma forma adequada de representar circuitos, e observações medidas em circuitos. Vamos adoptar a seguinte: para cada gate do tipo **Tipo**, identificada pelo nome **Id**, há um facto **`gate(Id,Tipo)`**; os pontos do circuito correspondem unicamente a entradas ou saídas de gates, representadas, respectivamente, por termos **`in(NumPorta,IdGate)`** e **`out(IdGate)`**; para cada ligação há um facto da forma **`ligado(Ponto1,Ponto2)`**; para cada valor observado há um facto da forma **`val(Ponto,Valor)`**. Por exemplo, o circuito e observações da figura são representados pelos factos:

```
ligado(out(g1),in(1,g3)).  gate(g1,nand).      val(in(1,g1),0).    val(in(2,g1),1).
ligado(out(g2),in(2,g3)).  gate(g2,nand).      val(in(1,g2),1).    val(in(2,g2),0).
                           gate(g3,or).        val(out(g2),1).     val(out(g3),0).
```

Repare que com esta representação até é possível prever o valor em todos os pontos do circuito para os quais é conhecido o input. Por exemplo, para o caso duma gate nand a funcionar correctamente:

```
val(out(G), V) :- gate(G,nand), not avariada(G), val(in(1,G),V1), val(in(2,G),V2),
                                        nand(V1,V2,V).
```

onde nand/3 é um predicado com 4 factos descrevendo o comportamento normal de uma gate nand. Há problema sempre que o valor previsto num qualquer ponto do circuito for diferente do valor observado nesse ponto.

4 a) Faça a modelação completa do sistema de diagnóstico descrito acima.

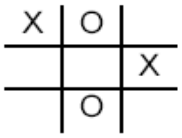
```
//Gerar todas as combinações possíveis de avarias
avariado(G):-gate(G,nand), not n_avariado(G).
n_avariado(G):- gate(G,nand), not _avariado(G).

//Não mostrar as gates que tenham um valor correcto
:-gate(G,nand),  val(in(1,G),V1),  val(in(2,G),V2),    val(out(G),Vout1),  nand(V1,V2,Vout2),
eq(Vout1, Vout2).

avariado(G):-gate(G,or), not n_avariado(G).
n_avariado(G):- gate(G,or), not _avariado(G).

//Não mostrar as gates que tenham um valor correcto
:-gate(G,or), val(in(1,G),V1), val(in(2,G),V2),  val(out(G),Vout1), or(V1,V2,Vout2), eq(Vout1,
Vout2).
```

6.2 Exame 2ª chamada 2003

Grupo 1 (Jogos de 2 oponentes)

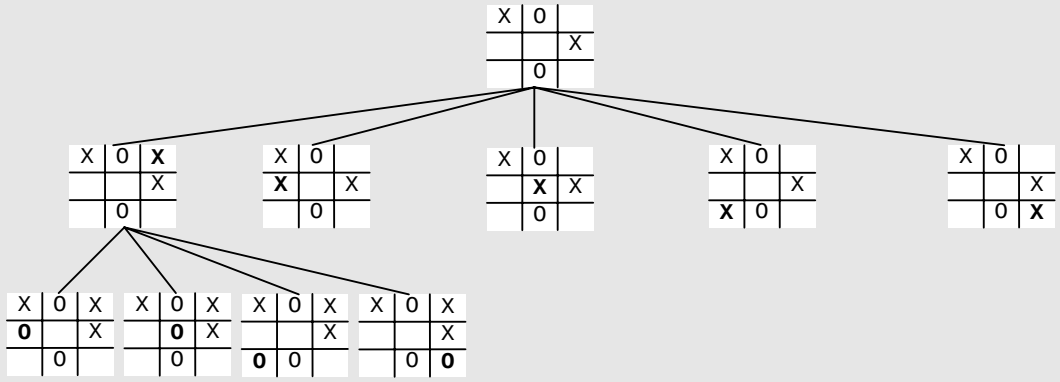
Considere que a seguinte função de avaliação (muitos simplista) para estados do jogo do galo:

$$F(S) = 6*D3x(S) + 2*D2x(S) + D1x(S) - (6*D3o(S) + 2*D2o(S) + D1o(S))$$



onde D*nj*(S) representa o número de linhas, colunas ou diagonais que contêm exactamente *n* peças do jogador *j* e nenhuma peça do outro jogador.
Na folha anexa encontra-se a árvore de pesquisa para este jogo, partindo da posição na figura, sabendo que o X joga a seguir, e com corte a profundidade 2.

1 a) Apresente, na folha anexa, o valor de cada um dos nós da árvore Min-Max e, com base nisso, diga qual o movimento que o 1º jogador deverá escolher.



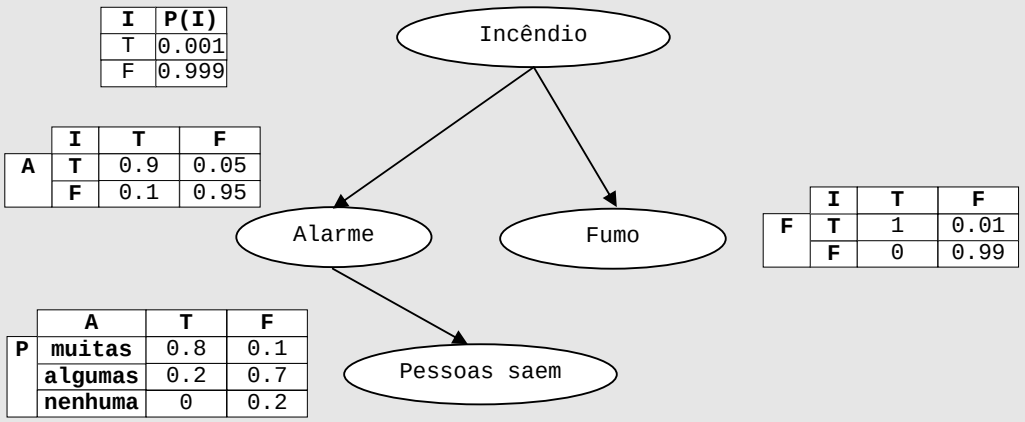
Grupo 3 (Redes de Bayes)

Um edifício de escritórios tem um alarme de incêndios que dispara 90% das vezes em que há mesmo um fogo. Por vezes também há falsos alarmes: mesmo não havendo fogo há uma probabilidade de 5% do alarme disparar. As pessoas que trabalham nesse edifício sabem desses falsos alarmes e, ao contrário do que deviam, nem sempre que o alarme toca saem do edifício. Verifica-se que apenas em 80% das vezes em que o alarme tocou é que saíram muitas pessoas do edifício; nas restantes vezes saíram apenas algumas pessoas. Com o funcionamento normal no edifício (isto é, quando o alarme não está a tocar) o movimento de saídas processa-se da seguinte forma: com uma probabilidade de 20% não está a sair ninguém; com uma probabilidade de 10% está a sair muita gente; caso contrário estão a sair apenas algumas pessoas. Além de tudo isto quando há fogo saí fumo das janelas. Aqui não se aplica a máxima de que "não há fumo sem fogo": por vezes, com probabilidade de 1%, há fumo a sair das janelas sem que haja fogo nenhum. A probabilidade de haver fogo em edifícios na zona onde este está situado é de 1 por mil.

3 a) Diga que variáveis aleatórias escolheria para representar o conhecimento acima e, para cada uma delas qual o seu domínio

```
A – Alarme toca {true, false}
I – Incêndio{true, false}
P – Pessoas saem{muitas, algumas, nenhuma}
F – Fumo{true, false}
```

3 b) Apresente uma rede Bayesiana (topologia e tabelas nos nós) que represente esse conhecimento



I	P(I)
T	0.001
F	0.999

	I	T	F
A	T	0.9	0.05
	F	0.1	0.95

	I	T	F
F	T	1	0.01
	F	0	0.99

	A	T	F
P	muitas	0.8	0.1
	algumas	0.2	0.7
	nenhuma	0	0.2

3 c) Calcule a probabilidade de haver fogo no edifício, sabendo que :

3 c1) estão muitas pessoas a sair do edifício

$$P(I = true \mid P = muitas) = \alpha P(I = true \wedge P = muitas)$$

$$= \alpha \sum_{a=\{true,\, false\}} \sum_{f=\{true,\, false\}} P(A = a \mid I) P(F = f \mid I) P(P = muitas \mid A = a) P(I)$$

$$= \alpha P(I) \sum_{a=\{true,\, false\}} P(A = a \mid I) P(P = muitas \mid A = a) \sum_{f=\{true,\, false\}} P(F = f \mid I)$$

$$I = true$$

$$= P(I = true) \sum_{a=\{true,\, false\}} P(A = a \mid I = true) P(P = muitas \mid A = a)$$

$$= 0.001 \times (0.9 \times 0.8 + 0.1 \times 0.1) = 0.00073$$

$$I = false$$

$$= P(I = false) \sum_{a=\{true,\, false\}} P(A = a \mid I = false) P(P = muitas \mid A = a)$$

$$= 0.999 \times (0.05 \times 0.8 + 0.95 \times 0.1) = 0.134865$$

$$\alpha = 0.00073 + 0.134865 = 0.135595$$

$$P(I \mid P = muitas) = \left\langle \frac{0.00073}{0.135595}, \frac{0.134865}{0.135595} \right\rangle = \langle 0.0054, 0.9946 \rangle$$

$$P(I = true \mid P = muitas) = 0.0054$$

3 c2) para além disso está a sair fumo das janelas.

$$P(I = true \mid P = muitas \wedge F = true) = \alpha P(I = true \wedge P = muitas \wedge F = true)$$
$$= \alpha \sum_{a=\{true,\, false\}} P(A = a \mid I) P(F = true \mid I) P(P = muitas \mid A = a) P(I)$$
$$= \alpha P(I) P(F = true \mid I) \sum_{a=\{true,\, false\}} P(A = a \mid I) P(P = muitas \mid A = a)$$

$$I = true$$
$$= P(I = true) P(F = true \mid I) \sum_{a=\{true,\, false\}} P(A = a \mid I = true) P(P = muitas \mid A = a)$$
$$= 0.001 \times 1 \times (0.9 \times 0.8 + 0.1 \times 0.1) = 0.00073$$

$$I = false$$
$$= P(I = false) \sum_{a=\{true,\, false\}} P(A = a \mid I = false) P(P = muitas \mid A = a)$$
$$= 0.999 \times 0.01 \times (0.05 \times 0.8 + 0.95 \times 0.1) = 0.00134865$$

$$\alpha = 0.00073 + 0.00134865 = 0.00207865$$

$$P(I \mid P = muitas) = \left\langle \frac{0.00073}{0.00207865}, \frac{0.00134865}{0.00207865} \right\rangle = \langle 0.35119, 0.6488 \rangle$$
$$P(I = true \mid P = muitas \wedge F = true) = 0.35119$$

6.3 Exame recurso 2003

Grupo 1 (Pesquisa)

Nas salinas da Ribeira do Sado existem 3 baldes para medir o sal a vender: um que leva 3 litros de sal, outro 5 litros e o outro 7 litros. Para vender quantidades de sal múltiplas destes 3 valores a coisa é simples: basta ir enchendo os baldes e despejando para o saco do cliente. Para outras quantidades a coisa já não é tão simples, mas nada que não se faça. Por exemplo, para vender 2 litros de sal, pode começar por se encher o balde de 7 litros, de seguida despejar o balde de 7 litros no de 5 até encher este último, e por fim despejar os 2 litros que ficaram no balde de 7 para o saco do cliente. O empregado que lá estava a vender o sal já tinha muita prática nesta forma de medir o sal, e fazia tudo muito rapidamente e sempre com um número mínimo de acções (em que se considera como uma acção o encher de um balde, o despejar o conteúdo de um balde para o saco do cliente, e o despejar de um balde para outro até que este último fique cheio ou que não haja mais sal no balde de origem). Mas agora veio um empregado novo que não se entende nada com este sistema de medida. De tal forma, que a direcção das salinas resolveu mandar fazer um programa para o ajudar.

1 a) Diga como formalizaria o problema por forma a que este pudesse ser resolvido por uma algoritmo de pesquisa em espaço de estados. I.e., diga como definiria estados, o estado inicial, a função de teste para estados objectivo e a função que devolve os sucessores dum estado.

```
e(B3,B5,B7,S).

e_inicial(0,0,0,0).

e_objectivo(0,0,0,2).

successor(e(),)
```

Grupo 2 (Redes de Bayes)

De há uns tempos para cá o número de mensagens de *spam* tem aumentado imenso. De tal forma, que começaram a proliferar pelo mercado vários programas para nos livrarem do *spam*, filtrando-o. Se já se debateu com este problema, e procurou programas para procederem a esta filtragem, certamente já reparado que a maioria deles anunciam que usam "tecnologia de *Bayesian networks*". Pois é! São mesmo as redes de Bayes que aprendeu nesta disciplina!

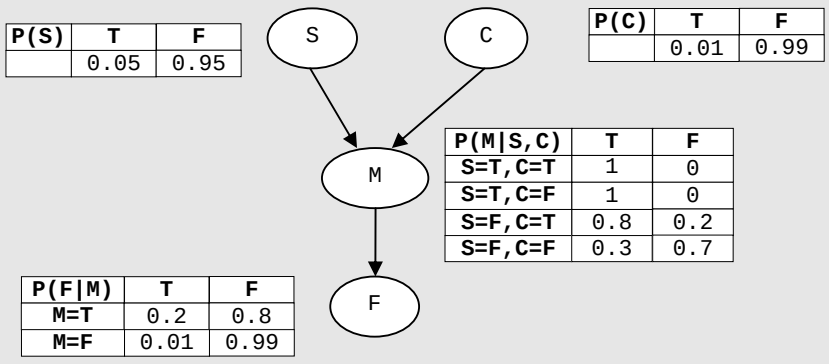
A ideia é simples. O que esses programas têm é uma rede de Bayes com nós para critérios definidos sobre mensagens e relacionados com o facto duma mensagem ser ou não *spam* (e.g. ter certas palavras no **subject**, ter certos tipos de endereços no **from**, ter uma percentagem grande de figuras no **corpo** da mensagem, etc.). Têm também um nó para determinar a probabilidade duma mensagem ser *spam*.

Recebida uma mensagem são avaliados os tais critérios e, com base neles e na rede, determinada a probabilidade da mensagem ser *spam*. Se essa probabilidade for maior que um certo limite, a mensagem é automaticamente filtrada. Além disso, os valores nas tabelas de probabilidades dos nós são actualizados ao longo do tempo de acordo com classificações das mensagens feitas pelo utilizador à posteriori. Essas classificações permitem saber, para cada combinação dos critérios, quantas mensagens foram ou não *spam* e, assim, preencher e actualizar as tabelas.

Imagine agora uma versão simplificada dum desses programas, onde os únicos critérios usados são: o facto da palavra "Viagra" vir no **subject**; o facto da sequência de palavras "lowest price" vir no **corpo** da mensagem; e o facto do endereço no **from** ser fictício. Além disso, após algum tempo de uso verificou-se que: 20% das mensagens classificadas como *spam* tinham endereço fictício no **from**, enquanto apenas 1% das mensagens que não foram *spam* tinham endereços fictícios; todas as mensagens com "Viagra" no **subject** foram classificadas como *spam*; das mensagens sem "Viagra" no **subject** mas com "lowest price" no **corpo**, 80% foram spam; das restantes mensagens, sem "Viagra" nem "lowest price", mesmo assim 30% foram *spam*. Verificou-se ainda que 5% das mensagens recebidas tinham "Viagra" no **subject** e que 1% das mensagens tinham "lowest price" no **corpo**.

2 a) Sugira uma topologia para a rede de Bayes deste programa simplificado de filtro de *spam*.

M – mensagem é spam
S – subject contém "Viagra"
C – corpo contém "lowest price"
F – from é fictício



P(S)	T	F
	0.05	0.95

P(C)	T	F
	0.01	0.99

P(M|S,C)	T	F
S=T,C=T	1	0
S=T,C=F	1	0
S=F,C=T	0.8	0.2
S=F,C=F	0.3	0.7

P(F|M)	T	F
M=T	0.2	0.8
M=F	0.01	0.99

2 b) Apresente as tabelas de probabilidades em cada um dos nós dessa rede, na situação descrita.

(em cima)

2 c) Calcule a probabilidade de uma dada mensagem ser *spam*, sabendo que:

2 c1) se detectou que tinha "lowest price" no **corpo** da mensagem

$$P(M = true \mid C = true) = \alpha P(M \wedge C)$$
$$= \alpha \sum_{s=\{true,\, false\}} P(M \mid C, S) P(S) P(C)$$
$$= \alpha P(C) \sum_{s=\{true,\, false\}} P(M \mid C, S = s) P(S = s)$$

$$M = true$$
$$P(C = true) \sum_{s=\{true,\, false\}} P(M = true \mid C = true, S = s) P(S = s)$$
$$= 0.01 \times (1 \times 0.05 + 0.8 \times 0.95) = 0.0081$$

$$M = false$$
$$P(C = true) \sum_{s=\{true,\, false\}} P(M = false \mid C = true, S = s) P(S = s)$$
$$= 0.01 \times (0 \times 0.05 + 0.2 \times 0.95) = 0.0019$$

$$\alpha = 0.0081 + 0.0019 = 0.01$$
$$P(M \mid C = true) = \left\langle \frac{0.0081}{0.01}, \frac{0.0019}{0.01} \right\rangle = \langle 0.81, 0.19 \rangle$$
$$P(M = true \mid C = true) = 0.81$$

2 c2) para além disso se verificou que o endereço no **from** é fictício

$$P(M = true \mid C = true, F = true) = \alpha P(M \wedge C \wedge F)$$
$$= \alpha \sum_{s=\{true,\, false\}} \sum_{f=\{true,\, false\}} P(M \mid C = true, S = s) P(S = s) P(C = true) P(F = f \mid M)$$
$$= \alpha P(C = true) \sum_{s=\{true,\, false\}} P(M \mid C = true, S = s) P(S = s) \sum_{f=\{true,\, false\}} P(F = f \mid M)$$

$$M = true$$
$$P(C = true) \sum_{s=\{true,\, false\}} P(M = true \mid C = true, S = s) P(S = s) \sum_{f=\{true,\, false\}} P(F = f \mid M)$$
$$= 0.01 \times (1 \times 0.05 + 0.8 \times 0.95) \times (0.2 + 0.8) = 0.0081$$

$$M = false$$
$$P(C = true) \sum_{s=\{true,\, false\}} P(M = false \mid C = true, S = s) P(S = s)$$
$$= 0.01 \times (0 \times 0.05 + 0.2 \times 0.95) \times (0.01 + 0.99) = 0.0019$$

$$\alpha = 0.0081 + 0.0019 = 0.01$$
$$P(M \mid C = true) = \left\langle \frac{0.0081}{0.01}, \frac{0.0019}{0.01} \right\rangle = \langle 0.81, 0.19 \rangle$$
$$P(M = true \mid C = true) = 0.81$$