Soluções de IA - Exercícios sugeridos do livro do Russel (ver exercícios sugeridos na página `http://www.cos.ufrj.br/ ines/courses/cos740/listex.html`)

Cap. 2

2.1 Uma medida de desempenho é usada por um observador externo para avaliar o comportamento do agente de acordo com seus objetivos (pode ser medida de eficiência, p.e. tempo, medida de qualidade etc). Uma função utilidade é utilizada pelo próprio agente para avaliar seu comportamento a cada passo.

Cap. 3

3.3 Possíveis soluções (bem abstratas...):

a.

- Estado inicial: Olhando a primeira página do primeiro livro.
- Estado final: Olhando a entrada no catálogo que contém o número do Sr. Jimwill Zollicoffer em Alameda.
- Operadores: Olhar um novo livro; olhar uma nova página; olhar uma nova entrada numa página.
- Custo: podemos medir em segundos, por exemplo: 10seg para achar um novo livro, 2 para uma nova página etc.

b.

- Semelhante ao anterior, porém o estado final é testar na listagem se existe alguém com nome Jimwill (ou inicial J) em Alameda. Podemos ainda verificar se esta é a pessoa procurada, ligando para o número encontrado no catálogo.

c.

- Estado inicial: algum local na floresta amazônica
- Objetivo: estar no mar
- Operadores: seguir corrente acima, seguir corrente abaixo, voltar à terra, seguir orientações norte, sul, leste, oeste
- custo: distância percorrida

d.

- Estado inicial: nenhuma região colorida
- Objetivo: todas as regiões coloridas e nenhuma região adjacente com a mesma cor
- Operadores: atribuir uma cor a uma região
- custo: número de atribuições de cores

e.

- Estado inicial: como descrito no texto
- Objetivo: macaco tem bananas
- Operadores: subir no banco, descer do banco, mover banco de uma posição para outra, andar de uma posição para outra, pegar bananas (se estiver em cima do banco)
- custo: número de operações

f.

- Estado inicial: alguma loalização na cidade
- Objetivo: estar na farmácia
- Operadores: andar em frente, virar à direita, virar à esquerda, procurar por placa de farmácia
- custo: número de passos percorridos

3.5 Busca com custos negativos e ciclos são fontes de problemas para todo algoritmo de otimização. Na prática, indicam uma falha na modelagem do domínio do problema.

a) Não. O algoritmo continua tendo que explorar todos os ramos, porque cada ramo pode levar a uma seqüência de transições de custo negativo.

b) Se houver um ciclo com custos negativos, um agente racional entraria num loop. A única razão para não entrar no loop, seria encontrar um outro ramo com custo melhor do que o ramo que iria causar o ciclo.

c) Pessoas não entram em loop, porque cada estado repetido nunca é exatamente igual ao que já passou (por exemplo, a pessoa pode ficar entediada de ver sempre a mesma paisagem...). Agentes artificiais podem evitar loops como este adicionando, por exemplo, um tempo limitado para que o agente saia de um estado.

3.6
a)

```
function GENERAL_SEARCH2(problem, QUEUEING_FN) retorna solucao ou
falha
nodes <- MAKE_QUEUE(MAKE_NODE(INITIAL_STATE[problem]))
loop do
  if nodes is empty then return falha
  node <- REMOVE_FRONT(nodes)
  new_nodes <- EXPAND(node,OPERATORS[problem])
  if GOAL_TEST[problem] aplicado ao estado de qq nÃ³ de new_nodes
  suceder then return node
  nodes <- QUEUEING_FN(nodes,new_nodes)
end
```

b) Solução possível: adicionar o GOAL_TEST à? função QUEUEING_FN. Se o teste suceder, a função enfileira aquele nó no início da fila. Caso contrário, é adicionado a sua posição normal. NB: não necessariamente este teste garante que a solução ótima vá ser encontrada.

Cap. 4

4.3

- Objetivo: todo quadrado (com exceção dos hachurados) contém uma letra, e todo slot de palavra contém uma palavra do dicionário. Podemos adicionar a restrição de que nenhuma palavra pode ser usada mais de uma vez.
- Estado inicial: uma matriz de quadrados com todos vazios e um dicionário de palavras.
- operadores: utilizando uma solução baseada em colocar palavras (poderia ser orientada a preencher letras). Escolher um palavra do dicionário e colocá-la em qualquer slot de palavra com mesmo tamanho, a menos que haja letras conflitantes já colocadas em algum dos quadrados. O estado resultante terá uma palavra a mais colocada no tabuleiro e menos uma palavra disponível no dicionário.
- custo: não importa para este problema.
- estratégia de busca: a mais adequada é utilizar um algoritmo para satisfação de restrições. Variáveis são associadas aos slots de palavras, o domínio das variáveis seria uma lista de palavras de tamanho apropriado existentes no dicionÃ¡rio, as restrições seriam que nenhuma palavra pode ser usada mais de uma vez e, se duas palavras se interceptam, precisam se interceptar na mesma letra.
- heurística: várias escolhas. Variável most-constrained é uma heurística boa, porém pode ter um custo alto quando o dicionário é muito grande, pois não queremos perder muito tempo, por exemplo, examinando 50.000 palavras para descobrir quantas delas cabem em algum slot. Uma outra solução pode ser selecionar k palavras de forma aleatória e verificar quantas delas cabem em cada slot.

4.5 Um exemplo é tentar ir de Rimnicu Vilcea para Lugoj. O caminho mais curto vai pela parte sul do mapa passando por Mehadia, Dobreta e Craiova. Uma busca gulosa usando como heurística a distância em linha reta entre cidades, começando em Rimnicu Vilcea não encontraria a melhor solução. Começando em Lugoj, a heurística vai levar diretamente para Mehadia. Uma busca gulosa retornaria a Lugoj e entraria num ciclo entre estas duas cidades.
4.10 Solução: fazer a busca "forward" com uma heurística que estima a distância ao objetivo, e a busca "backward" com um heurística que estima a distância ao nó inicial. Problema: não há garantia de encontrar a solução ótima.

Cap. 6

6.1 Dada uma sentença S, construir uma tabela verdade com uma linha para cada possível combinação de valores verdade para as proposições em S, tal que a última coluna da tabela seja S. Se esta última coluna tem todos os valores verdadeiros, S é uma tautologia. Se todos os valores forem falsos, S é insatisfatÃ?vel.
6.3

a) válida

b) satisfatível para alguma interpretação

c) satisfatível para alguma interpretação

d) válida

e) válida

f) válida

g) válida

h) satisfatível para alguma interpretação

6.4 a sentença é equivalente à sentença 2+2=4, logo está falando de alguma expressão aritmética. No entanto, se considerarmos que 2+2=4 é sempre verdadeiro no sistema decimal, a sentença é uma tautologia, e portanto não está falando sobre nenhum dos dois, ou está falando sobre os dois.
6.5 Sentenças:

```
Mythical => Immortal
not Mythical => not Immortal ^ Mammal
Immortal V Mammal => Horned
Horned => Magical
```

Não podemos provar que é Mythical. Podemos provar que é Magical and Horned.
Para provar Horned:
Para Horned ser verdadeiro, ou Immortal ou Mammal tem que ser verdadeiro. Para a primeira sentença ser verdadeira, com Mythical verdadeiro, Immortal tem que ser verdadeiro. Por outro lado, pela segunda sentença, se Mythical for falso, not Immortal E Mammal têm que ser verdadeiros. Como Mythical nãoo pode ser falso e verdadeiro ao mesmo tempo, ou Mammal é verdadeiro ou Immortal é verdadeiro. Portanto, Horned tem que ser verdadeiro. Se Horned é verdadeiro, concluímos trivialmente que Magical também é verdadeiro.
6.7

a) 4 modelos, considerando os valores possíveis de C e D

b) 12 modelos, 4 valores possíveis para C e D, multiplicado pelas 3 maneiras como A V B podem ser verdadeiros (V-V, V-F e F-V)

c) 2 modelos, considerando os dois valores que D pode assumir

6.8 Um conectivo binário é definido por uma tabela verdade com 4 linhas (correspondentes às combinações possíveis de F e V para os dois literais envolvidos). Cada uma das 4 linhas da tabela verdade pode produzir um resultado F ou V. Portanto, podemos ter $2^4 = 16$ tabelas possíveis, e, portanto, 16 conectivos possíveis (isto é, cada saída diferente da tabela, um conjunto das 4 linhas, vai corresponder a um conectivo diferente). Seis deles correspondem a T, F, P, Q, not P e not Q. 4 deles são ∧, V, =¿, ¡=¿. Os seis restantes são a implicação inversa (¡=) e as negações de ∧ (nand), V (nor), =¿, ¡=¿, e ¡=.
6.9 Não faz muita diferença em relação ao conhecimento já armazenado e à nível de lógica, porque as respostas deveriam ser as mesmas, ou seja, as conseqüências lógicas continuam as mesmas. Do ponto de vista de implementação existe uma grande diferença, principalmente em termos de eficiência (tempo e espaço).
6.10 É viável, porém continuamos tendo que codificar 64 regras para WumpusAhead.
6.12 Assumindo as seguintes proposições: JP: Jones is a programmer, JK: Jones is an engineer, JM: Jones is a manager...

```
Each person has one job...
JP V JK V JM
SP V SK V SM
CP V CK V CM
Each job has one person
JP V SP V CP
JK V SK V CK
JM V SM V CM
Jones owes the programmer \$ 10 (portanto ele não é o programmer)
not JP
not JM
not SM
```

Solução: JK ∧ SP ∧ CM

Cap. 7

7.2 Assuma o seguinte vocabulário:

- Takes(x,c): aluno x matriculado no curso c
- Fails(x,c): aluno x reprovado no curso c
- Score(x,c): nota do aluno x no curso c
- H: um curso de história específico
- B: um curso de biologia específico
- Dislikes(x,y): pessoa x não gosta de pessoa y
- Shaves(x,y): pessoa x barbeia pessoa y
- Student(x), Person(x), Man(x), Woman(x), Barber(x), Vegetarian(x), Smart(x), Professor(x), Politician(x): predicados satisfeitos por pessoas x da categoria do predicado

- Fools(x,y,t): pessoa x engana pessoa y no tempo t

a. $\neg\forall x Student(x) => (Takes(x,H) \wedge Takes(x,B))$

b. $\exists x Student(x) \wedge Fails(x,H) \wedge \forall y Student(y) \wedge Fails(x,H) \Rightarrow x = y$

c. $\exists x Student(x) \wedge Fails(x,H) \wedge Fails(x,B) \wedge \forall y Student(y) \wedge Fails(x,H) \wedge Fails(x,B) \Rightarrow x = y$

d. $\exists x \forall y Score(x,H) > Score(y,B)$

e. $\forall x Person(x) \wedge (\forall y Vegetarian(y) \Rightarrow Dislikes(x,y)) \Rightarrow Smart(x)$

f. $\forall x, y Person(x) \wedge Smart(y) \wedge Vegetarian(y) \Rightarrow \neg Likes(x,y)$

g. $\exists Woman(x) \wedge \forall y Man(y) \wedge \neg Vegetarian(y) \Rightarrow Likes(x,y)$

h. $\exists x Barber(x) \wedge \forall Man(y) \wedge \neg Shaves(y,y) \Rightarrow Shaves(x,y)$

i. $\forall x, y Person(x) \wedge Professor(y) \wedge \neg Smart(y) \Rightarrow \neg Likes(x,y)$

j. $\forall x Politician(x) \Rightarrow (\exists y \forall t Person(y) \wedge Fools(x,y,t)) \wedge (\exists t \forall y Person(y) \Rightarrow Fools(x,y,t)) \wedge \neg(\forall t \forall y Person(y) \Rightarrow Fools(x,y,t))$

7.6 NB: Pode ser que vocês utilizem uma abordagem diferente com nomes de predicados diferentes...Atenção para a definição recursiva utilizada para NthCousin na última sentença.

- $GrandChild(c,a) \equiv \exists b Child(c,b) \wedge Child(b,a)$
- $GreatGrandParent(a,d) \equiv \exists b, c Child(d,c) \wedge Child(c,b) \wedge (b,a)$
- $Brother(x,y) \equiv Male(x) \wedge Sibling(x,y)$
- $Sister(x,y) \equiv Female(x) \wedge Sibling(x,y)$
- $Daughter(d,p) \equiv Female(d) \wedge Child(d,p)$
- $Son(s,p) \equiv Male(s) \wedge Child(s,p)$
- $AuntOrUncle(a,c) \equiv \exists p Child(c,p) \wedge Sibling(a,p)$
- $Aunt(a,c) \equiv Female(a) \wedge AuntOrUncle(a,c)$
- $BrotherInLaw(b,x) \equiv \exists m Spouse(x,m) \wedge Sister(s,m)$
- $FirstCousin(c,k) \equiv \exists p AuntOrUncle(p,c) \wedge Parent(p,k)$
- $NthCousin(n,c,k) \equiv \exists p, f Parent(p,c) \wedge NthCousin(n-1,f,p) \wedge Child(k,f)$

7.7 Estes axiomas definem bem pertinência de conjuntos, porém não dizem nada sobre se um elemento não está no conjunto. Num sistema Prolog, que usa negação por falha e não negação lógica, funciona bem, porém em lógica pura podemos não conseguir provar se um elemento não pertence a um conjunto.
7.11

a. $At(Robot, Arad, So)$.

b. $\exists s At(Robot, Bucharest, s)$.

c. Uitlizando cálculo de situações (axioma do estado sucessor): $\forall a, x, y, s At(Robot, y, Result(a, s)) \equiv [(a = Go(x, y) \wedge DirectRoute(x, y) \wedge At(Robot, x, s)) \vee (At(Robot, y, x) \wedge \neg(\exists z a = Go(y, z) \wedge z \neq y))]$

d. Para representar qtde de combustível: $Fuel(Robot, s)$. Distância entre duas cidades medida em unidades de combustível consumido: $Distance(x, y)$. $Full$ vai ser uma constante representando a capacidade do tanque de combustível.

e. Situação inicial: $At(Robot, Arad, So) \wedge Fuel(Robot, s) = Full$. Axiomas para localização:

- $\forall a, x, y, s At(Robot, y, Result(a, s)) \equiv [(a = Go(x, y) \wedge DirectRoute(x, y) \wedge At(Robot, x, s) \wedge Distance(x, y) < Fuel(Robot, s)) \vee (At(Robot, y, s) \wedge \neg(\exists z a = Go(y, z) \wedge z \neq y))]$
- $\forall a, x, y, s Fuel(Robot, Result(a, s)) = f \equiv [(a = Go(x, y) \wedge DirectRoute(x, y) \wedge At(Robot, x, s) \wedge Distance(x, y) < Fuel(Robot, s) \wedge f = Fuel(Robot, s) - Distance(x, y)) \vee (f = Fuel(Robot, s) \wedge \neg(\exists v, w a = Go(v, w) \wedge v \neq w))]$
- Solução simples: adicionar $GasStation(x)$, que vai ser verdadeiro se a cidade x tiver postos de abastecimento. A ação $Fillup$ é descrita adicionando uma nova cláusula para o axioma que fla sobre Fuel, dizendo que $f = Full$ quando $a = Fillup$.

7.14 Como discutido em aula, é possível construir uma solução para o mundo do wumpus utilizando qualquer linguagem de programação. Porém, existem algumas vantagens de se utilizar uma representação lógica: não é necessário implementar os mecanismos de inferência que provavelmente seriam necessários na sua implementação em linguagem não lógica; é mais fácil representar situações, por exemplo, em que o agente pode ir para varias possíveis localizações no tabuleiro utilizando o conectivo lógico OU (ou seja, é mais fácil representar ambiguidades).

Cap. 9

9.1

a. {x/A, y/B, z/B} ou alguma permutação destas substituições.

b. Não há unificador possível.

c. {y/John, x/John}.

d. Não há unificador.

9.4

a. $Horse(x) \Rightarrow Mammal(x), Cow(x) \Rightarrow Mammal(x), Pig(x) \Rightarrow Mammal(x)$

b. $Offspring(x, y) \wedge Horse(y) \Rightarrow Horse(y)$

c. $Horse(BlueBeard)$

d. $Parent(BlueBeard, Charlie)$

e. $Offspring(x, y) \Rightarrow Parent(y, x), Parent(x, y) \Rightarrow Offspring(y, x)$

f. $Mammal(x) \Rightarrow Parent(G(x), x)$

9.5

na árvore de prova há um ciclo, portanto não conseguimos encontrar a solução.

as regras como foram definidas causam um loop por causa da sentença (b). O loop ocorre por causa da ordenação das cláusulas. Se o provador de teoremas tentar encontrar todas as soluções, não podemos evitar que entre em loop.

h = BlueBeard e h = Charlie

possíveis soluções: limitar a profundidade da prova (como busca em profundidade limitada), suspender a prova quando o estado se repete (como em tabulação).

9.8

a. $\forall x Horse(x) \Rightarrow Animal(x), \forall x, h Horse(x) \wedge HeadOf(h, x) \Rightarrow \exists y Animal(y) \wedge HeadOf(h, y)$

b.

A. $\neg Horse(x) \vee Animal(x)$
B. $Horse(G)$
C. $HeadOf(H, G)$
D. $\neg Animal(y) \vee \neg HeadOf(H, y)$

c. Resolva D e C produzindo $\neg Animal(G)$. Resolva esta nova sentença com A produzindo $\neg Horse(G)$. Resolva esta nova sentença com B para obter uma contradição.