Exercícios de Inteligência Artificial

Licenciatura em Engenharia Informática e de Computadores
Instituto Superior Técnico

2 de maio de 2023

Este documento contém um conjunto de exercícios propostos para a unidade curricular de Inteligência Artificial (IA) da Licenciatura em Engenharia Informática e de Computadores (LEIC) do Instituto Superior Técnico.

A unidade curricular de Inteligência Artificial faz parte do plano curricular da LEIC desde a sua criação em 1989. A maioria dos exercícios deste documento foram sendo propostos pelo corpo docente de IA ao longo dos anos. Alguns destes exercícios foram apresentados em testes ou exames.

Conteúdo

Capítulo 1

Procura Cega

Exercício 1

Considere o problema da Torre de Hanoi com 3 discos. Neste problema existem três pinos A, B e C e 3 discos de diâmetros diferentes, estando no início todos os discos colocados no pino A com os discos maiores por baixo dos mais pequenos. O problema consiste em deslocar todos os discos para o pino C, um a um, de modo a que fiquem na mesma posição relativa nunca podendo, em momento algum, existir um disco em cima de outro de menor diâmetro. O pino B pode ser usado como pino auxiliar.

a) Formule o problema da Torre de Hanoi com 3 discos como um problema de procura.

b) Qual o método de procura cega que escolheria para resolver o problema e porquê?

Resolução

A formulação do problema corresponde a identificar o conjunto de estados, o estado inicial, a função sucessores, a função objetivo e a função de custo (de caminhos).

Assim, o conjunto de estados com que vamos trabalhar é representado pelo conjunto de tuplos (X, Y, Z) em que X, Y, e Z são um de A, B ou C e que representam, respetivamente, os pinos em que estão o maior disco, o disco intermédio e o disco mais pequeno. Assim, o estado (A, A, B) representa o estado em que no pino A estão os dois maiores discos, no pino B está o menor disco e o pino C está vazio.
X = estado atual x = novo estado

O estado inicial é o estado (A, A, A).

A função sucessores recebe como argumento um estado e retorna o conjunto de estados que lhe podem suceder. Os estados que podem suceder a um estado (X, Y, Z) correspondem à união dos seguintes conjuntos:

- $\{(x, Y, Z) \mid Y \neq X \wedge Z \neq X \wedge x \neq X \wedge x \neq Y \wedge x \neq Z\}$
- $\{(X, y, Z) \mid Z \neq Y \wedge y \neq Y \wedge y \neq Z\}$
- $\{(X, Y, z) \mid z \neq Z\}$

A função objetivo retorna verdade para o estado (C, C, C) e falso para todos os outros estados.

A função de custo (de caminhos) calcula o custo desde a raiz da árvore até ao nó que está a ser considerado, somando-se o custo das várias transições envolvidas no caminho. Para este problema, pode-se considerar um custo unitário para cada transição, de modo que o custo de um caminho corresponde ao número de ações consideradas nesse caminho.

Exercício 2

Considere o problema das vasilhas. Dispomos de duas vasilhas, com capacidades de 3 e 5 litros. Inicialmente as vasilhas estão vazias e pretendemos medir uma certa quantidade de água. As ações que podemos executar são:

- encher totalmente cada uma das vasilhas;
- verter o conteúdo de uma vasilha noutra, até a primeira ficar vazia ou a segunda cheia;
- deitar fora todo o conteúdo de uma vasilha.

Para resolver este problema utilizando os algoritmos de procura estudados vamos usar a seguinte representação em Python para o problema:

```
class Estado:
    def __init__(self, v3 = 0, v5 = 0):
        self.v3 = v3  # líquido existente na vasilha de 3 litros
        self.v5 = v5  # líquido existente na vasilha de 5 litros
```

A classe possui ainda os seguintes métodos: `encher_3`, `encher_5`, `verter_5_para_3`, `verter_3_para_5`, `esvaziar_3` e `esvaziar_5`. Cada um destes métodos devolve um novo estado que resulta de aplicar esse operador. Por exemplo, para um estado em que a vasilha de três litros está vazia e a vasilha de cinco litros tem quatro litros, o operador `verter_5_para_3` devolve um estado em que a primeira vasilha tem três litros e a segunda um litro.

Considere ainda a seguinte classe que implementa parcialmente o problema das vasilhas utilizando as classes do ficheiro `search.py`.

```
from search import Problem

class ProblemaVasilhas(Problem):

    def __init__(self, objetivo3, objetivo5):
        self.initial = Estado(0, 0)  # inicialmente ambas as vasilhas
                                     # estão vazias
        self.objetivo3 = objetivo3
        self.objetivo5 = objetivo5

    def actions(self, estado: Estado) -> list:
        raise NotImplementedError()
```

```
def result(self, estado: Estado, acao) -> Estado:
    raise NotImplementedError()

def goal_test(self, estado: Estado) -> bool:
    raise NotImplementedError()
```

a) Defina os métodos `encher_3`, `verter_5_para_3` e `esvaziar_5` da classe `Estado`.

b) Defina o método `actions` da classe `ProblemaVasilhas`. Este método recebe um estado e retorna uma lista com todas as ações que podem ser executadas nesse estado.

c) Tendo em conta os métodos definidos para cada uma das ações, defina o método `result` que dado um estado e uma ação retorna o estado resultante de se aplicar a ação no estado recebido.

d) Defina o método `goal_test`, que dado um estado verifica se esse estado é uma solução para o problema.

Resolução

a)
```
def encher_3(self):
    return Estado(3, self.v5)

def verter_5_para_3(self):
    liquido_a_verter = min(3 - self.v3, self.v5)
    return Estado(self.v3 + liquido_a_verter,
                  self.v5 - liquido_a_verter)

def esvaziar_5(self):
    return Estado(self.v3, 0)
```

b)
```
def actions(self, estado: Estado) -> list:
    return ["encher3", "encher5",
            "verter3_5", "verter5_3",
            "esvaziar3", "esvaziar5"]
```

c)
```
def result(self, estado: Estado, acao) -> Estado:
    if acao == "encher3":
        return estado.encher_3()
    elif acao == "encher5":
        return estado.encher_5()
    elif acao == "verter3_5":
        return estado.verter_3_para_5()
    elif acao == "verter5_3":
        return estado.verter_5_para_3()
    elif acao == "esvaziar3":
        return estado.esvaziar_3()
    elif acao == "esvaziar5":
        return estado.esvaziar_5()
```

```
        else:
            raise NotImplementedError()

d) def goal_test(self, estado: Estado) -> bool:
        return estado.v3 == self.objetivo3 and \
               estado.v5 == self.objetivo5
```

Exercício 3

Considere o espaço de estados definido pelo estado inicial, 1, e pela função que gera os sucessores de um estado, sucessores$(n) = \{2n, 2n+1\}$. Considerando o estado objetivo 9, desenhe a árvore de procura indicando a ordem de geração e a ordem de expansão para cada nó usando:

a) Procura em profundidade primeiro;

b) Procura em largura primeiro;

c) Procura em profundidade limitada com limite de profundidade 4;

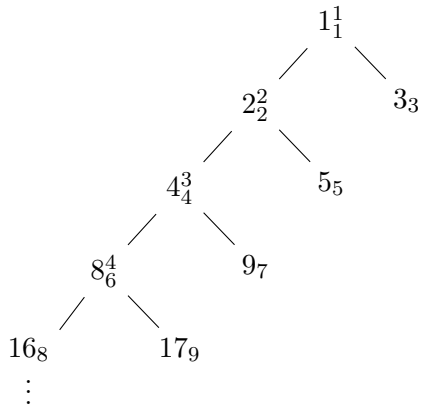
d) Procura em profundidade iterativa.

(Em caso de empate, explore os nós por ordem numérica crescente.)

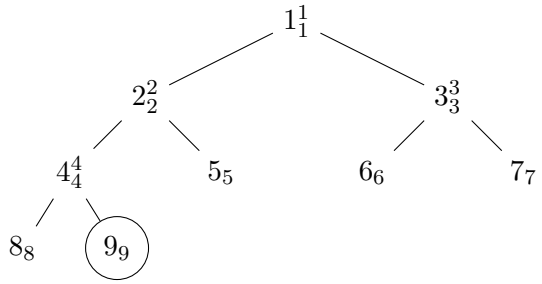
Resolução

Notação: N^{oe}_{og} oe : ordem de expansão; og : ordem de geração

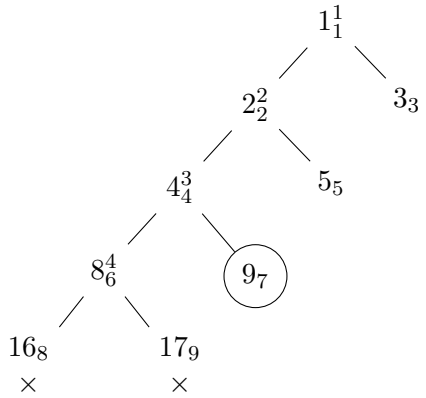
a) Procura em profundidade primeiro:



b) Procura em largura primeiro;



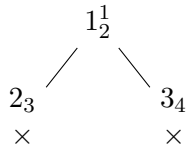
c) Procura em profundidade limitada com limite de profundidade 4;



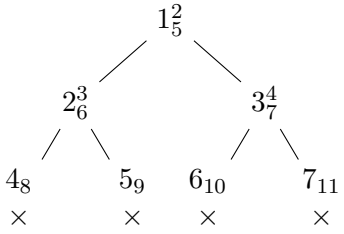
d) Procura em profundidade iterativa.

$lim = 0$

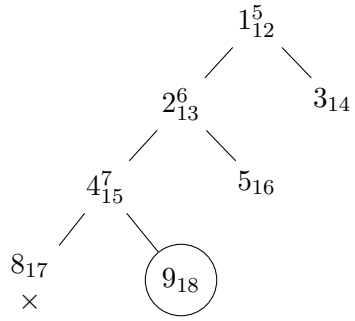
$lim = 1$



$lim = 2$

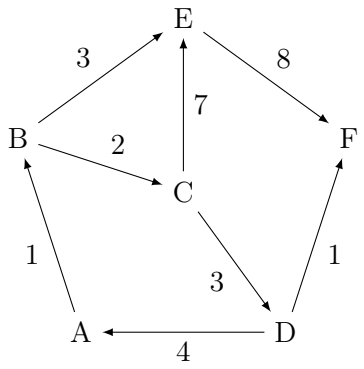


$lim = 3$



Exercício 4

Considere o seguinte espaço de estados contendo os estados A, B, C, D, E e F e pelas transições representadas (os números representam os custos das operações). Em caso de empate na colocação dos nós na fronteira, utiliza-se a ordem alfabética dos estados para desempatar.



Suponha que quer resolver o problema de ligar o estado A ao estado F. Escreva a ordem pela qual os nós foram gerados e expandidos para cada uma das seguintes estratégias de procura:

a) Procura em profundidade primeiro;

b) Procura em largura primeiro;

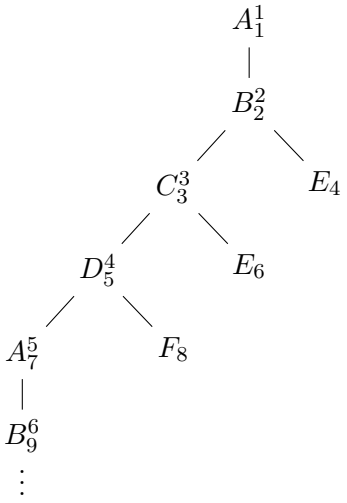
c) Procura de custo uniforme;

d) Procura em profundidade iterativa.

Resolução

Notação: N^{oe}_{og} oe : ordem de expansão; og : ordem de geração

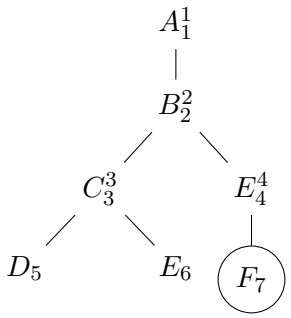
a) Procura em profundidade primeiro:



Ordem de Geração: A [B C E D E A F] B ...

Ordem de Expansão: [A B C D] A ...

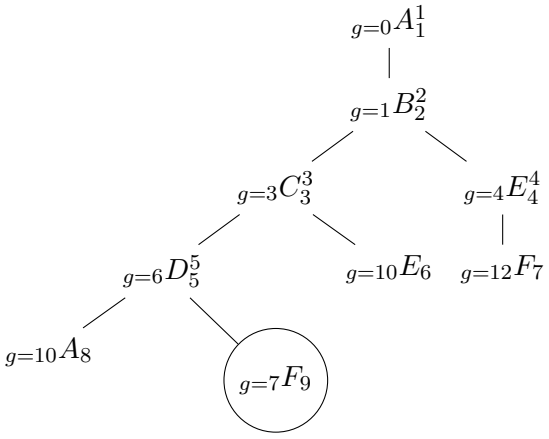
b) Procura em largura primeiro:



Ordem de Geração: A B C E D E F

Ordem de Expansão: A B C E

c) Procura de custo uniforme:



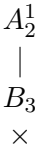
Ordem de Geração: A B C E D E F A F

Ordem de Expansão: A B C E D

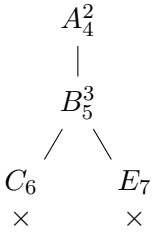
d) Procura em profundidade iterativa.

$lim = 0$

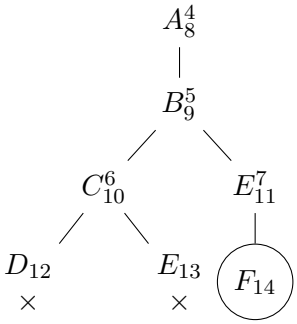
$lim = 1$



$lim = 2$



$lim = 3$

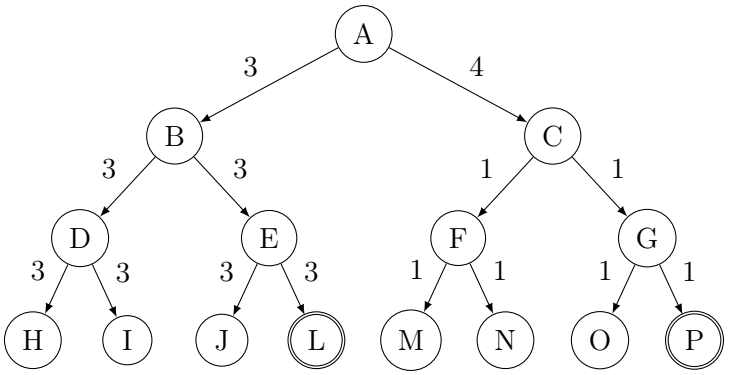


Ordem de Geração: A A B A B C E A B C E D E F

Ordem de Expansão: A A B A B C E

Exercício 5

Considere o seguinte espaço de estados com os estados de A a P, representado na figura a seguir:



Os estados são representados por um círculo com o seu nome. Os números nos arcos que ligam dois estados representam os custos das transições entre os dois estados. O problema que estamos a considerar tem como estado inicial o estado A e estados objetivo os estados L e P. Considere que os sucessores de um nó são gerados pela ordem alfabética dos estados (por exemplo, o primeiro sucessor de F é o nó M e não o nó N) e que em caso de empate na ordem de colocação de nós na fila de nós por tratar vigora a ordem alfabética.

a) Considere a procura em largura primeiro e escolha a única opção correta:

   i) O nó I é testado como solução antes do nó G e ambos os nós são testados antes de se encontrar a solução.

   ii) O nó G é testado como solução antes do nó N e ambos os nós são testados antes de se encontrar a solução.

   iii) O nó E não é testado como solução.

   iv) O nó D é testado como solução antes do nó J e ambos os nós são testados antes de se encontrar a solução.

   v) O nó O é testado como solução.

   vi) O nó I é testado como solução antes do nó E e ambos os nós são testados antes de se encontrar a solução.

b) Considere a procura em profundidade primeiro e escolha a única opção correta:

   i) O nó I é gerado antes do nó G e ambos os nós são gerados.

   ii) O nó G é gerado antes do nó I e ambos os nós são gerados.

   iii) O nó E não é gerado.

   iv) O nó C é gerado antes do nó J e ambos os nós são gerados.

   v) O nó C é gerado antes do nó G e ambos os nós são gerados.

   vi) O nó I é gerado antes do nó E e ambos os nós são gerados.

c) Considere a procura em profundidade limitada com limite 2 e escolha a única opção correta:

   i) São testados como solução exatamente 3 nós.

   ii) São testados como solução exatamente 4 nós.

   iii) São testados como solução exatamente 7 nós.

   iv) São testados como solução exatamente 8 nós.

   v) São testados como solução exatamente 11 nós.

   vi) São testados como solução exatamente 15 nós.

Resolução

a) iv

b) iv

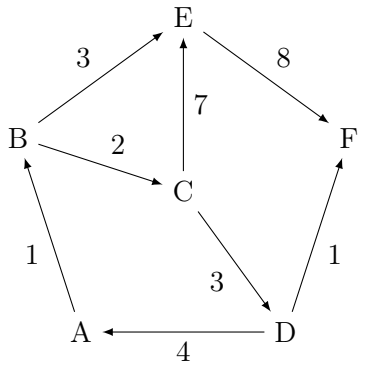
c) iii

Capítulo 2

Procura Informada

Exercício 1

Considere o seguinte espaço de estados contendo os estados de A a F, em que os valores associados às ligações entre dois estados correspondem ao custo da transição entre eles:



Suponha que quer resolver o problema de ligar o estado A ao estado F. Para tal, considere a seguinte função heurística:

x	A	B	C	D	E	F
h(x)	5	3	1	1	1	0

1.1) Usando a ordem alfabética crescente em caso de empate, indique a ordem pela qual os nós são gerados, processados e expandidos e a solução encontrada ao aplicar cada uma das seguintes estratégias de procura:

a) Procura Gananciosa;

b) Procura A*;

c) Procura IDA*;

d) Procura RBFS.

Resolução

OG: Ordem de Geração

OP: Ordem de Processamento

OE: Ordem de Expansão

Expoente: $f(x) = \begin{cases} h(x), & \text{se Procura Gananciosa} \\ c(x) + h(x), & \text{c.c.} \end{cases}$

a) **OG:** A^5 B^3 C^1 E^1 D^1 E^1 A^5 F^0
OP: A^5 B^3 C^1 D^1 F^0
OE: A^5 B^3 C^1 D^1
Solução: A B C D F

b) **OG:** A^5 B^4 C^4 E^5 D^7 E^{11} F^{12} A^{15} F^7
OP: A^5 B^4 C^4 E^5 D^7 F^7
OE: A^5 B^4 C^4 E^5 D^7
Solução: A B C D F

c) $lim = f(A) = 5$

OG: A^5 B^4 C^4 E^5 D^7 E^{11} F^{12}
OP: A^5 B^4 C^4 D^7 E^{11} E^5 F^{12}
OE: A^5 B^4 C^4 E^5

$lim = 7$

OG: A^5 B^4 C^4 E^5 D^7 E^{11} A^{15} F^7
OP: A^5 B^4 C^4 D^7 A^{15} F^7
OE: A^5 B^4 C^4 D^7
Solução: A B C D F

d) **OG:** A^5 B^4 C^4 E^5 D^7 E^{11}
OP: A^5 B^4 C^4 D^7
OE: A^5 B^4 C^4
A alternativa (E^5) é melhor que D^7. Actualizar $f(C) = 7$ e retroceder.

OG: F^{12}
OP: E^5 F^{12}
OE: E^5

A alternativa (C^7) é melhor que F^{12}. Actualizar $f(E) = 12$ e retroceder.

OG: D^7 E^{11} A^{15} F^7
OP: C^7 D^7 F^7
OE: C^7 D^7
Solução: A B C D F

1.2) A heurística é admissível? Justifique.

Resolução

Sim, a heurística é admissível, pois nunca sobrestima o custo mínimo (h^*) para atingir uma solução a partir de qualquer um dos estados:

x	A	B	C	D	E	F
h(x)	5	3	1	1	1	0
h*(x)	7	6	4	1	8	0

1.3) A heurística é consistente? Justifique.

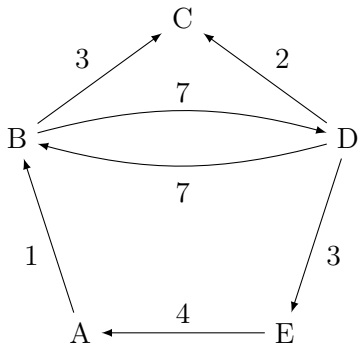
Resolução

Não, a heurística não é consistente, pois a estimativa para o estado A é superior à soma do custo da transição de A para B com a estimativa para o estado B, isto é:

$$h(A) > c(A, B) + h(B)$$

Exercício 2

Considere o seguinte espaço de estados contendo os estados de A a E, em que os valores associados às ligações entre dois estados correspondem ao custo da transição entre eles:



Suponha que quer resolver o problema de ligar o estado A ao estado E. Para tal, considere a seguinte função heurística:

x	A	B	C	D	E
h(x)	5	3	1	1	0

2.1) Usando a ordem alfabética crescente em caso de empate, indique a ordem pela qual os nós são gerados, processados e expandidos e a solução encontrada ao aplicar cada uma das seguintes estratégias de procura:

a) Procura Gananciosa;

b) Procura A*;

c) Procura IDA*;

d) Procura RBFS.

Resolução

OG: Ordem de Geração

OP: Ordem de Processamento

OE: Ordem de Expansão

Expoente: $f(x) = \begin{cases} h(x), & \text{se Procura Gananciosa} \\ c(x) + h(x), & \text{c.c.} \end{cases}$

a) **OG:** A^5 B^3 C^1 D^1 B^3 C^1 E^0

OP: A^5 B^3 C^1 D^1 E^0

OE: A^5 B^3 C^1 D^1

Solução: A B D E

b) **OG:** A^5 B^4 C^5 D^9 B^{18} C^{11} E^{11}

OP: A^5 B^4 C^5 D^9 C^{11} E^{11}

OE: A^5 B^4 C^5 D^9 C^{11}

Solução: A B D E

c) $lim = f(A) = 5$

OG: A^5 B^4 C^5 D^9

OP: A^5 B^4 C^5 D^9

OE: A^5 B^4 C^5

$lim = 9$

OG: A^5 B^4 C^5 D^9 B^{18} C^{11} E^{11}

OP: A^5 B^4 C^5 D^9 B^{18} C^{11} E^{11}

OE: A^5 B^4 C^5 D^9

$lim = 11$

OG: A^5 B^4 C^5 D^9 B^{18} C^{11} E^{11}

OP: A^5 B^4 C^5 D^9 B^{18} C^{11} E^{11}

OE: A^5 B^4 C^5 D^9 C^{11}

Solução: A B D E

d) **OG:** A^5 B^4 C^5 D^9 B^{18} C^{11} E^{11}

OP: A^5 B^4 C^5 D^9 C^{11} E^{11}

OE: A^5 B^4 C^5 D^9 C^{11}

Solução: A B D E

2.2) A heurística é admissível? Justifique.

Resolução

Sim, a heurística é admissível, pois nunca sobrestima o custo mínimo (h^*) para atingir uma solução a partir de qualquer um dos estados:

x	A	B	C	D	E
h(x)	5	3	1	1	0
h*(x)	11	10	∞	3	0

2.3) A heurística é consistente? Justifique.

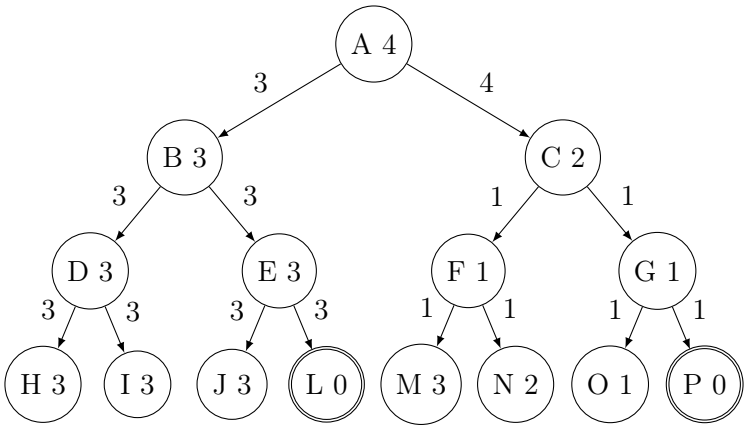
Resolução

Não, a heurística não é consistente, pois a estimativa para o estado A é superior à soma do custo da transição de A para B com a estimativa para o estado B, isto é:

$$h(A) > c(A, B) + h(B)$$

Exercício 3

Considere o espaço de estados, com os estados de A a P, representado na figura seguinte:



Os estados são representados por um círculo com o seu nome e têm associado um valor correspondente à estimativa dada por uma função heurística. Os valores associados aos arcos que ligam dois estados representam o custo de transição entre eles. O problema que estamos a considerar tem como estado inicial o estado A e como estados objectivo os estados L e P. Considere que os sucessores de um nó são gerados pela ordem alfabética dos estados (por exemplo, o primeiro sucessor de F é o nó M e não o nó N) e que, em caso de empate na ordem de colocação de nós na fila de nós por processar, vigora também a ordem alfabética.

a) Considere a procura gananciosa e escolha a única opção correcta:

   i) O nó I é testado como solução antes do nó G e ambos os nós são testados como solução.

   ii) O nó G é testado como solução antes do nó F e ambos os nós são testados como solução.

   iii) O nó M é testado como solução antes do nó P e ambos os nós são testados como solução.

   iv) O nó C é testado como solução antes do nó G e ambos os nós são testados como solução.

v) O nó B é testado como solução antes do nó G e ambos os nós são testados como solução.

vi) O nó F é testado como solução antes do nó O e ambos os nós são testados como solução.

b) Considere a procura IDA* e escolha a única opção correcta:

i) A solução é encontrada ao fim de 3 iterações.

ii) Os sucessores de B são gerados duas vezes.

iii) O nó F é gerado duas vezes.

iv) Os sucessores de D são gerados uma vez.

v) O nó M é testado como solução uma vez.

vi) O nó C é testado como solução uma vez.

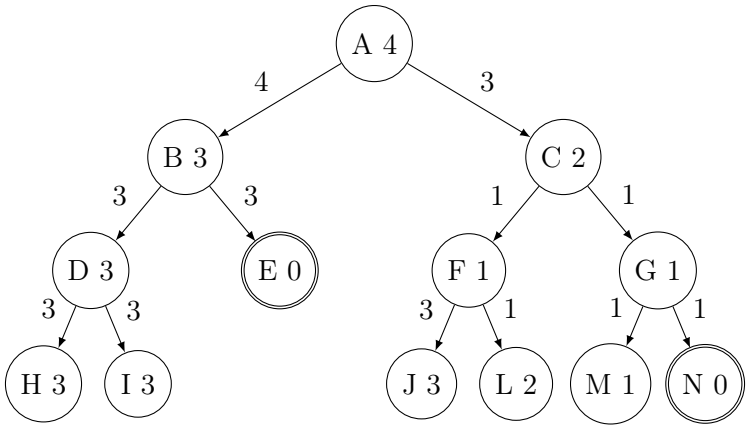
Resolução

a) iv

b) vi

Exercício 4

Considere o espaço de estados, com os estados de A a N, representado na figura seguinte:



Os estados são representados por um círculo com o seu nome e têm associado um valor correspondente à estimativa dada por uma função heurística. Os valores associados aos arcos que ligam dois estados representam o custo de transição entre eles. O problema que estamos a considerar tem como estado inicial o estado A e como estados objectivo os estados E e N. Considere que os sucessores de um nó são gerados pela ordem alfabética dos estados (por exemplo, o primeiro sucessor de F é o nó J e não o nó L) e que, em caso de empate na ordem de colocação de nós na fila de nós por processar, vigora também a ordem alfabética.

a) Considere a procura gananciosa e escolha a única opção correcta:

i) O nó B é testado como solução e há mais 3 nós na fila de nós por testar.

ii) O nó C é testado como solução e há mais 3 nós na fila de nós por testar.

iii) O nó F é testado como solução e há mais 3 nós na fila de nós por testar.

iv) O nó G é testado como solução e há mais 3 nós na fila de nós por testar.

v) O nó B é testado como solução e há mais 2 nós na fila de nós por testar.

vi) O nó C é testado como solução e há mais 2 nós na fila de nós por testar.

vii) O nó G é testado como solução e há mais 2 nós na fila de nós por testar.

viii) O nó N é testado como solução e há mais 2 nós na fila de nós por testar.

b) Considere a procura IDA* e escolha a única opção correcta:

i) São feitos 6 testes para verificar se um nó é solução.

ii) São feitos 8 testes para verificar se um nó é solução.

iii) São feitos 10 testes para verificar se um nó é solução.

iv) São feitos 12 testes para verificar se um nó é solução.

v) São feitos 14 testes para verificar se um nó é solução.

vi) São feitos 15 testes para verificar se um nó é solução.

vii) São feitos 16 testes para verificar se um nó é solução.

viii) São feitos 17 testes para verificar se um nó é solução.

Resolução

a) iv

b) i

Capítulo 3

Procura em Ambientes Complexos

Exercício 1

Imagine que vai mudar de apartamento e que tem que de se mudar do seu antigo apartamento para o novo. Considere os seguintes dados:

- Uma lista $L = \{a_1, a_2, \cdots, a_n\}$ de n items, cada um com um tamanho $s(a_i) > 0$.
- Um conjunto de M caixas, cada uma com uma capacidade C (assuma que $M \times C$ excede em muito a soma dos tamanhos dos seus itens).

O objetivo é empacotar os items no menor número de caixas possível, garantindo que a soma do tamanho dos items colocados numa caixa não excede a sua capacidade.

Formule este problema como um problema de procura local, definindo estados, vizinhança e função de avaliação. Pondere como podem ser evitados mínimos locais.

Resolução

Estados: Qualquer combinação de items dentro das caixas e items fora da caixa;
Estado inicial: Caixas sem items;
Solução: Qualquer combinação com todos os items dentro das caixas;
Vizinhança: Colocar ou retirar um item de uma caixa;
Função de avaliação: minimizar número de caixas usadas.

Para evitar mínimos locais a função de avaliação tem de considerar também a maximização do número de items dentro das caixas.

Exercício 2

Considere uma máquina de escalonamento de processos em que o objetivo é minimizar o tempo de execução e a vizinhança é:

N_1 : Mover um processo de uma máquina para a outra.

N_2 : Trocar dois processos entre máquinas.

N_3 : Trocar um processo de uma máquina por dois processos de uma outra máquina.

Supõe que utilizamos um steepest descent, i.e. encontrar a melhor solução na vizinhança. Qual é a complexidade de aplicar o steepest descent para cada vizinhança?

Resolução

Sendo n o número de processos, então a complexidade é a seguinte:

N_1 : complexidade $O(n)$

N_2 : complexidade $O(n^2)$

N_3 : complexidade $O(n^3)$

Exercício 3

Nos algoritmos genéticos as gerações seguintes são formuladas através de recombinação do genoma dos pais e, opcionalmente, seguido de mutação.

a) Explique o objetivo da recombinação.

Resolução

O principal objetivo da recombinação (recombination ou cross-over) é para combinar material genético de dois parentes diferentes para gerar descendência que herdam propriedades desejáveis de ambos os pais, resultando numa solução com melhor fitness.

b) Explique também o objetivo de introduzir mutações.

Resolução

O objetivo de mutação é introduzir diversidade na população para garantir que o algoritmo não converge prematuramente.

c) Porque deve ser a frequência de mutação (probabilidade de a descendia ser mutada) escolhida de forma cuidada?

Resolução

Se a frequência de mutação é muito baixa, a variabilidade introduzida na população é muito baixa o que resultará numa convergência prematura. Se a frequência é muito alta, então as propriedades herdadas que conferem melhor fitness são perdidas de uma geração para a outra, aproximando o algoritmo de uma procura aleatória.

d) Descreva, em pseudocódigo, a estrutura básica de um algoritmo genético e de forma sucinta os componentes principais.

Resolução

Step 1 Inicialização: Gerar a população inicial de soluções.

Step 2 Selecionar pais: selecionar um subgrupo da população contendo n soluções.

Step 3 Reprodução: Gerar $n/2$ pares de pais; Gerar descendente para cada par usando recombinação; Aplicar mutação com a frequência definida.

Step 4 Gerar população: Adicionar os descendentes à população.

Step 5 Continuar: se o critério de terminação for atingido, terminar. caso contrário, voltar ao Step 2.

Step 1, a população inicial é gerada, que pode consistir de um grupo de solulçoes geradas aleatoriamente.
Step 2, os pais que serão utilizados na recombinação são selecionados, tipicamente baseado no fitness (probabilidade de escolha aumenta com o fitness).
Step 3, descrito na alínea a) e b).
Setp 5, o(s) critério(s) de convergência são verificados, por exemplo, (i) se o fitness convergiu, i.e. não melhorou durante um certo número de gerações; ou (ii) um determinado número de gerações foi atingido.

Exercício 4

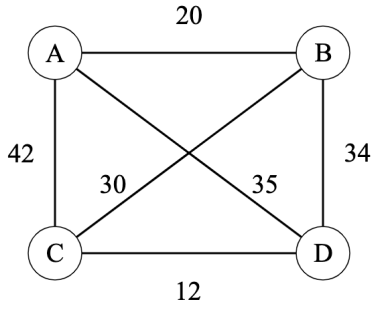
Considere o problema das 4 raínhas e o uso de *local beam search* para o resolver, com k=2. Ilustre a aplicação do algoritmo a este problema. Considere que os estados iniciais são gerados aleatoriamente, pelo que ficam ao seu critério. Defina a forma de avaliação para cada estado.

Resolução

Inicialmente, colocar as quatro rainhas no tabuleiro, uma em cada coluna. Usar como heurística h o número de ataques. Em cada passo seleccionar os k = 2 sucessores com menor valor de h.

Exercício 5

Considere o problema do caixeiro viajante, i.e. *traveling salesman person*, e o algoritmo de *simulated annealing*. Considere a instância que se segue, caracterize um estado e defina uma função de avaliação. Ilustre o funcionamento do algoritmo ao longo de 4 iterações.



Resolução

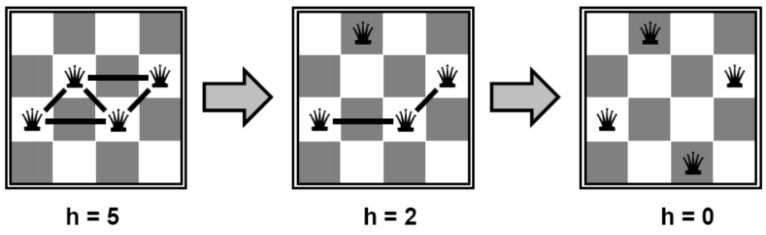
Um estado é caracterizado por uma lista cujos elementos são as localidades A, B, C e D. A ordem dos elementos na lista define a ordem pela qual o caixeiro viajante percorre as localidades, voltando no final à localidade inicial. Dada uma lista de localidades, a função de avaliação devolve a soma das distâncias entre as localidades, percorridas pela ordem em que constam na lista.
Para o caso de TSP, o algoritmo simulated anneling começa por gerar uma lista com as localidades ordenadas aleatoriamente. Em cada iteração, duas localidades trocam a posição na lista. Se a distância entre as localidades diminui, então é mantida a nova ordenação (SIM). Se a distância aumenta, então é mantida a nova ordenação com uma dada probabilidade (Sim/Não).

A aplicação do algoritmo simulated anneling envolve aleatorização. Assim, o exemplo que se encontra de seguida é meramente ilustrativo.
Consideremos o estado inicial, gerado aleatoriamente, (A,C,B,D) correspondente à distância 141. A partir deste estado inicial, podemos obter as 4 iterações que se seguem:

	Novo Estado	Distância	Delta	Aceite?
Iteração 1	(B,C,A,D)	141	0	Não
Iteração 2	(A,C,D,B)	108	23	SIM
Iteração 3	(A,B,D,C)	108	0	Sim
Iteração 4	(A,B,C,D)	97	11	SIM

Exercício 6

Considere o seguinte cenário do problema 4 Rainhas em que inicialmente havia 5 ataques (h=5), após a mudança de posição os ataques passaram para 2 (h=2) e depois passaram a ser 0 (h=0).



Crie um cenário aleatório do problema 5 Rainhas (estado inicial) onde os ataques devem ser 6 (h=6). Crie 5 filhos a partir do estado inicial, alterando as posições de cada rainha, uma de cada vez. Calcule o valor da heurística para cada estado filho e determine qual deve escolher usando o algoritmo de *hill climbing*. Simule mais duas iterações do algoritmo.

Resolução

Comecemos por recordar como formulamos o problema de 5 rainhas no contexto do algoritmo *hill climbing*.
Estado: 5 rainhas num tabuleiro 5x5 com uma rainha por coluna.
Estado inicial: gerado aleatoriamente.
Solução: 5 rainhas no tabuleiro sem ataques entre rainhas.
Vizinhança: uma rainha é movida para outra posição na mesma coluna.
Função de avaliação: número de pares de rainhas que se atacam.

Para representar um estado podemos usar uma lista de inteiros, em que o elemento na posição i da lista corresponde à linha em que está colocada a rainha que está posicionada na coluna i.
Consideremos o estado inicial que se segue, com h=6: (3,4,1,2,1)
Usando a representação por listas, os filhos gerados poderão ser os seguintes:
(4,4,1,2,1) h=5
(3,2,1,2,1) h=7
(3,4,2,2,1) h=4
(3,4,1,3,1) h=4
(3,4,1,2,5) h=4

É seleccionado o estado filho (3,4,1,2,5) com h=4, ainda que existissem outros estados com igual valor de h.
A iteração seguinte pode gerar os seguintes filhos:
(4,4,1,2,5) h=3
(3,1,1,2,5) h=3
(3,4,4,2,5) h=3
(3,4,1,4,5) h=4
(3,4,1,2,3) h=7

É assim seleccionado o estado filho (3,1,1,2,5) com h=3, ainda que existissem outros estados com igual valor de h.
A iteração seguinte pode gerar os seguintes filhos:
(4,1,1,2,5) h=2
(3,2,1,2,5) h=5
(3,1,4,2,5) h=0
(3,1,1,1,5) h=3
(3,1,1,2,3) h=6

É assim seleccionado o estado filho (3,1,4,2,5) com h=0 que é uma solução.

Exercício 7

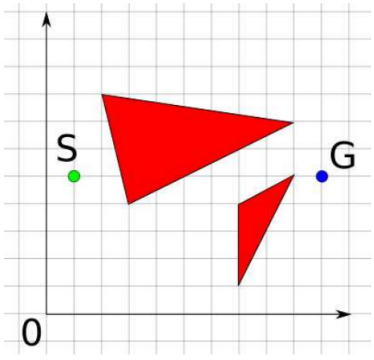
Considere o problema de encontrar o caminho mais curto entre dois pontos num plano que possui obstáculos poligonais convexos. A figura abaixo ilustra a navegação de robots entre polígonos. A origem O está nas coordenadas (0, 0). O estado inicial é em (1, 5). O objetivo está em (10, 5). Esta é uma idealização do problema que um robot pode ter de resolver para navegar num ambiente real. Considere o algoritmo de *hill climbing*.

(a) Explique como o algoritmo funcionaria para alcançar o objetivo.

(b) Mostre como obstáculos não convexos podem resultar num máximo local, usando um exemplo.

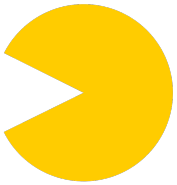
(c) É possível que a procura fique presa em obstáculos convexos?

(d) O algoritmo de *simulated annealing* escaparia aos minimos locais nesta família de problemas?



Resolução

(a) Um estado seria caracterizado pela posição do robot. A vizinhança consistiria nas 4 posições adjacentes à posição do robot, acautelando as posições com obstáculos e as posições com coordenadas negativas. A função de avaliação (h) seria a distância Euclideana entre a posição do robot e o objetivo. Em cada iteração seria seleccionada uma das posições adjacentes com menor valor de h.
(b) *Observação: pela forma como construímos a heurística, o problema está em ficar num mínimo local e não num máximo local como referido na pergunta.* Efectivamente, um obstáculo não convexo pode resultar num mínimo local. Imaginemos que temos um obstáculo não convexo com a forma que se segue. Este obstáculo está posicionado entre S e G. O robot começa por avançar e entrar na *boca do pacman*, chegando a uma situação em que o único estado na vizinhança implica andar para trás. Como isso implicaria aumentar o valor de h, o robot pára.



(c) Também é possível que o robot páre com um obstáculo convexo. No exemplo que é apresentado, o robot começaria por se deslocar em direção a G. A páginas tantas, teria de se desviar do obstáculo para uma posição mais afastada de G e portanto com menor valor de h.
(d) Sim, o algoritmo de *simulated annealing* escaparia aos máximos locais nesta família de problemas. Na situação em que o algoritmo de *hill climbing* pára, existe uma probabilidade não nula de o algoritmo de *simulated annealing* escolher um estado com menor valor de h e assim prosseguir a procura.

Exercício 8

Espaços continuos Considere o problema de escolher qual a aceleração de um veiculo por forma a reduzir o tempo de viagem t mas ao mesmo tempo não gastar demasiada energia. Considere que o veiculo terá de percorrer 2 unidades, que a aceleração inicial é 2, e que o custo é dado por $c = t + a$ onde t é o tempo que demora a percorrer 2 unidades e a é a aceleração.
Usando procura local escolha uma nova aceleração (assumindo um passo $\alpha = 0.1$).

Resolução

O tempo é dado por:

$$\begin{aligned} x &= \frac{1}{2}at^2 = 2 \\ t &= \left(\frac{4}{a}\right)^{(1/2)} = 2a^{-1/2} \end{aligned}$$

cost

$$\begin{aligned} c &= t + a \\ c &= 2a^{-1/2} + a \end{aligned}$$

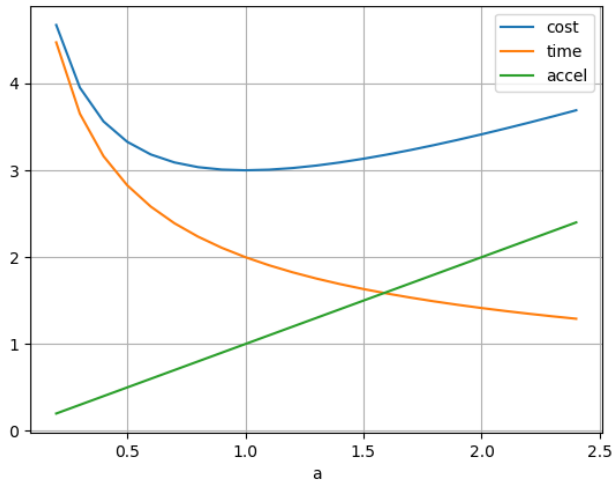
gradiente

$$\frac{dc}{da} = -a^{-3/2} + 1$$

passo de gradiente

$$\begin{aligned} a = a - \alpha\frac{dc}{da} &= a - \alpha(-a^{-3/2} + 1) \\ &= 1.93 \end{aligned}$$

$$\begin{aligned} custoinicial &= 2 + \sqrt{2} = 3.41 \\ custofinal &= 1.93 + \sqrt{4/1.93} = 3.37 \end{aligned}$$



Exercício 9

Ações não determinísticas

Considere o seguinte ambiente com as ações cima e direita. O estado objectivo é o estado D. Os estados A e B são escorregadios e por isso fazendo qualquer uma das ações um dos resultados é ficar no mesmo estado.

```
-----
|C|D|
-----
|A|B|
-----
```

a) Represente a função de transição:

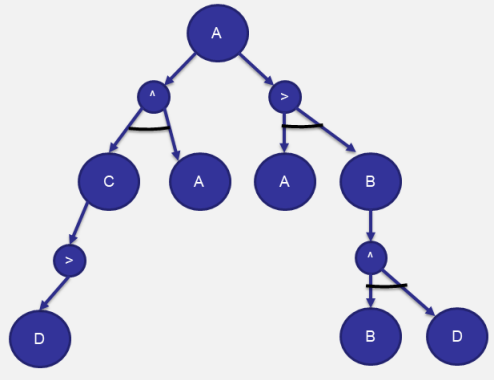
b) Calcule a árvore de procura usando o algoritmo AND-OR.

c) Qual a melhor solução para este problema?

Resolução

a) Represente a função de transição:
(A,Direita) -> [A,B]
(B,Direita) -> [B]
(C,Direita) -> [D]
(A,Cima) -> [A,C]
(B,Cima) -> [B,D]
(C,Cima) -> [C]

b) Calcule a árvore de procura AND-OR.



c) A melhor solução seria: [Cima, enquanto Estado == A faz Cima, Direita]

Exercício 10

Considere 2 salas A e B e um tigre. Há 4 ações EscutarA, EscutarB, EntrarA, EntrarB. Ao escutar uma sala é possível ouvir um barulho identificando com certeza se essa sala

tem ou não um tigre. O objetivo é entrar na sala sem tigre.

a) Represente a evolução da crença para cada uma das ações EscutarA e EscutarB após a crença inicial.

b) Tendo em conta a evolução da crença crie um plano de contingência para este problema.

Resolução

a) crença inicial é $b([\_, \_]; [T, \_]; [\_, T])$

Após EscutarA e ouvir ruído: $b([T, \_])$

Após EscutarA e não ouvir ruído: $b([\_, T]$

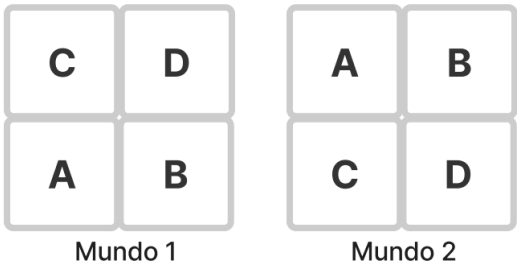
Após EscutarB e ouvir ruído: $b([\_, T]$

Após EscutarB e não ouvir ruído: $b([T, \_])$

b) [EscutarA, Se b((T, _)) então EntrarB caso contrário EntrarA]
ou
[EscutarB, Se b((_, T)) então EntrarA caso contrário EntrarB]

Exercício 11

Considere um agente que navega num mundo quadriculado com as ações Cima, Baixo, Esquerda e Direita. O agente sabe que começa no estado inicial A, mas não sabe se o mundo em que está é o 1 ou o 2 presentes na figura abaixo. O seu objétivo é chegar a D. Considere ainda que a crença inicial é b(1, 2) ou b = [1, 2].



(a) Qual será a crença após cada uma das ações?

(b) Qual o plano de contigência mais adequado para resolver a tarefa?

Resolução

(a) (A, Cima) -> [C, A]:
- Caso fique em C: b(1)
- Caso fique em A: b(2)
(A, Baixo) -> [A, C]:
- Caso fique em A: b(1)

- Caso fique em C: b(2)
(A, Direita) -> [B]: b(1,2)
(A, Esquerda) -> [A]: b(1,2)

(b) [Cima, Se Estado == A então (Direita, Baixo), Caso contrário Direita]
ou
[Baixo, Se Estado == C então Direita, Caso contrário Direita, Cima]
Nota: Estado == A ou Estado == C, podiam ser trocado pela belief (b(2)). ...

Observação parcial

Exercício 12

Considere a seguinte trajetória onde as ações são A e B e os estados vão de 1 a 5. Onde o estado terminal é o 5.

$$(1A- > 2); (2A- > 1); (1B- > 2); (2B- > 4); (4A- > 5)$$

a) Usando o algoritmo LRTA* calcule a função de transição e os valores de cada estado. Considere a heurística $h(1) = 3, h(2) = 2, h(3, 4) = 1h(5) = 0$. E que todas as ações têm custo 1.

b) Desenhe um ambiente que possa gerar esta trajetória.

Resolução

a) [X, Y]->Z, Result(X,Y)=Z
Custo_LRTA*(H(Result(X, Y)))):

- Se [X, U] não está na função de transiçao (ainda não foi descoberto): h(X)
- CC : Custo_Da_Acao(X, Y) + H(Result(X, Y))

(a) $[1, A]- > 2$
função de transição [1,A]->2
2 é objetivo? Não
2 é um novo estado? Sim, então: $H(2) = h(2) = 2$
H(1,2,3,4,5)=[0,2,0,0,0]
H[1] = min(Custo_LRTA*(H(Result(1, A))), Custo_LRTA*(H(Result(1, B))))
= min(Custo_Da_Acao(1, A) + H(Result(1, A)), Custo_Da_Acao(1, B) + H(Result(1, B)))
= min(1 + H(2), h(1))
= min(3,3)
= 3

(b) $[2, A]- > 1$
função de transição [1,A]->2 [2,A]->1
1 é objetivo? Não
1 é um novo estado? Não
H(1,2,3,4,5)=[3,2,0,0,0]
H[2] = min(1 + H(result(2,A)), 1 + H(result(2,B)))

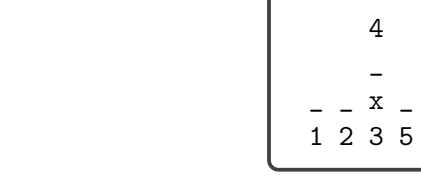
= min(1 + H(1), h(2)
= min(4,2)
= 2

(c) $[1, B] -> 2$
função de transição [1,A]->2 [1,B]->2 [2,A]->1
2 é objetivo? Não
2 é um novo estado? Não.
H(1,2,3,4,5)=[3,2,0,0,0]
H[1] = min(1 + H(result(1,A)), 1 + H(result(1,B)))
= min(1 + H(2), 1 + H(2))
= min(3,3)
= 3

(d) $[2, B] -> 4$
função de transição [1,A]->2 [1,B]->2 [2,A]->1 [2,B]->4
4 é objetivo? Não. 4 é um novo estado? Sim, então: $H(4) = h(4) = 1$
H(1,2,3,4,5)=[3,2,0,0,0]
H[2] = min(1 + H(result(2,A)), 1 + H(result(2,B)))
= min(1 + H(1), 1 + H(4))
= min(4,2)
= 2

(e) $[4, A] -> 5$
função de transição [1,A]->2 [1,B]->2 [2,A]->1 [2,B]->4 [4,A]->5
5 é objetivo? Sim.

b) Esta sequência pode representar um mini jogo de plataformas. A corresponde a andar para o lado e B corresponde a saltar para o lado. Andar para o lado no estado 2 leva de volta ao estado 1, o agente morre no estado 3.



Capítulo 4

Satisfação de Restrições

Exercício 1

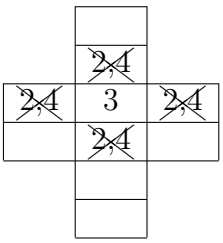
Um quadrado latino de dimensão n é uma matriz $n \times n$ cujas entradas são números de 1 a n e em que não existem linhas nem colunas com números repetidos. Considere o problema de preencher uma matriz $n \times n$ formando um quadrado latino. Formule este problema como um Problema de Satisfação de Restrições, indicando as variáveis, os domínios das variáveis e as restrições.

Resolução

Variáveis: n^2 variáveis, representadas por X_{ij} com $1 \leq i, j \leq n$, em que X_{ij} corresponde ao conteúdo da posição da matriz na linha i e na coluna j.
Domínio das variáveis: valores inteiros de 1 a n.
Restrições: Cada número ocorre exactamente uma vez por linha: $\forall_{1 \leq i \leq n} \text{AllDiff}(X_{i1}, X_{i2}, X_{i3}, \ldots, X_{in})$. Cada número ocorre exactamente uma vez por coluna: $\forall_{1 \leq j \leq n} \text{AllDiff}(X_{1j}, X_{2j}, X_{3j}, \ldots, X_{nj})$

Exercício 2

Considere a seguinte tabela, em que se deseja escrever os números de 0 a 9, sem repetições, de modo que nenhum número fique em contacto com o seu sucessor ou antecessor (por exemplo, se existir um 3 no quadrado central superior, não pode existir nem um 2 nem um 4 em cima, em baixo, ou num dos dois lados).



a) Formule este problema como um CSP, indicando as variáveis, os domínios das variáveis e as restrições.

Resolução

Variáveis: $X_1, X_2, \ldots, X_9, X_{10}$, Domínio: $X_j \in \{0, 1, 2, \ldots, 9\}, j = 1, \ldots, 10$. Considerando que as variáveis ocupam a tabela como se segue:

	X_1	
	X_2	
X_3	X_4	X_5
X_6	X_7	X_8
	X_9	
	X_{10}	

as restrições são as seguintes (note que esta é apenas uma de várias soluções possíveis): AllDiff$(X_1, \ldots, X_{10}), X_4 \neq X_3 \pm 1, X_4 \neq X_2 \pm 1, X_4 \neq X_5 \pm 1, X_4 \neq X_7 \pm 1, X_7 \neq X_6 \pm 1, X_7 \neq X_8 \pm 1, X_7 \neq X_9 \pm 1, X_1 \neq X_2 \pm 1, X_{10} \neq X_9 \pm 1, X_3 \neq X_6 \pm 1, X_5 \neq X_8 \pm 1$.

b) Qual o conjunto das variáveis com que pode iniciar a procura em árvore considerando a heurística do maior grau? Justifique.

Resolução

A heurística do maior grau escolhe a variável que está envolvida no maior número de restrições com variáveis ainda não atribuídas. Assim, para iniciar a procura em árvore tendo em conta esta heurística, as variáveis escolhidas seriam X_4 e X_7.

Exercício 3

Considere um quadrado mágico com a dimensão 3×3. A cada posição é atribuído um valor inteiro de 1 a 9. (Todas as posições têm valores diferentes.) O objectivo é colocar os valores de tal forma que cada linha, coluna e diagonal (de comprimento 3) tenham a mesma soma. Segue-se um exemplo de um quadrado mágico:

4	3	8
9	5	1
2	7	6

Considere este problema como um CSP. Duas formulações distintas têm como variáveis e respectivos domínios:

a) Uma variável para cada posição, com o domínio [1..9], cujo valor corresponde ao inteiro contido nessa posição.

b) Uma variável para cada número, com domínio [1..9], cujo valor corresponde à posição onde se encontra o número.

Para qual das formulações é mais simples definir as restrições? Defina as restrições para a formulação mais simples.

Resolução

A formulação a) é a mais simples. Para esta formulação, as variáveis com domínio [0..9] são as seguintes:

X1	X2	X3
X4	X5	X6
X7	X8	X9

E as restrições são as seguintes:
X1 + X2 + X3 = X4 + X5 + X6 = X7 + X8 + X9 = (linhas)
= X1 + X4 + X7 = X2 + X5 + X8 = X3 + X6 + X9 = (colunas)
= X1 + X5 + X9 = X3 + X5 + X7 (diagonais)

Exercício 4

Considere o problema de cripto-aritmética que se segue. A cada letra deve ser atribuído um dígito diferente, de modo a que a seguinte soma fique correcta:

$$\begin{array}{cccccc} & & & E & A & T \\ + & & T & H & A & T \\ \hline A & P & P & L & E & \end{array}$$

a) Formule este problema como um CSP. Sugestão: use como variáveis auxiliares X1, X2, X3, X4, cujo valor é 0 ou 1, e que representam os dígitos das dezenas resultantes da adição.

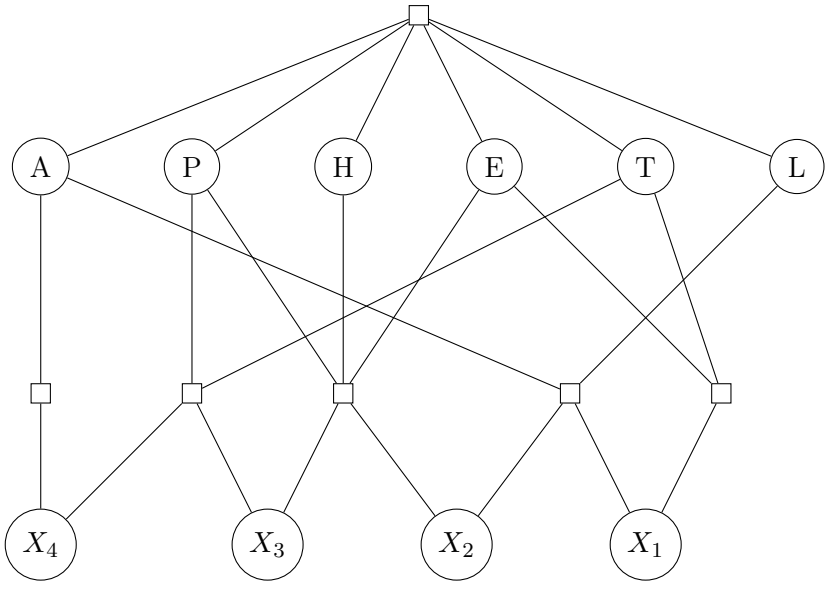
Resolução

Variáveis: A,P,T,E,H,L,X1,X2,X3,X4
Domínios: DA = DP = DT = DE = DH = DL =[0..9], DX1 = DX2 = DX3 = DX4 =0,1.
Restrições: AllDiff(A,P,T,E,H,L), T + T = E + 10 * X1, X1 + A + A = L + 10 * X2, X2 + E + H = P + 10 * X3, X3 + T = P + 10 * X4, X4 = A
(Solução do problema: 819 + 9219 = 10038)

b) Construa o respectivo grafo de restrições.

Resolução

Cada restrição é representada por um quadrado e os arcos fazem a ligação entre cada restrição e as variáveis envolvidas na restrição.



c) Qual o conjunto de variáveis com que pode iniciar a procura em árvore considerando a heurística do maior grau ("heuristic degree")?

Resolução

Variáveis T, E, A e P. Todas estas variáveis estão em 3 restrições, enquanto que as restantes aparecem apenas em 2 restrições.

d) Considere que está a usar procura local juntamente com a heurística que minimiza o número de conflitos e que é devolvida a atribuição A=4, E=8, H=2, L=3, P=0, T=9, X1=1, X2=0, X3=1, X4=1. Quais as variáveis que o algoritmo vai considerar mudar? Suponha que o algoritmo decidiu aleatoriamente alterar o valor atribuído à variável A. Que valor deve ser atribuído a esta variável?

Resolução

Variáveis A, L, X1, X2 ou X4. Valor 1.

Exercício 5

Considere uma formalização do problema das 4-rainhas num CSP. As variáveis são x_1, x_2, x_3, x_4, todas elas com o domínio $\{1, 2, 3, 4\}$. $x_j = i$ significa que existe uma rainha na posição (i, j). Quais as restrições do problema? (Pode usar o operador AllDiff$(x_1, \ldots, x_n)$, que obriga a que todos os valores das variáveis $(x_1, \ldots, x_n)$ tenham valores diferentes.)

Resolução

A restrição AllDiff(x_1, x_2, x_3, x_4) assegura que não existem rainhas na mesma linha. A restrição $\forall_{i,j} : i \neq j \rightarrow |x_i - x_j| \neq |i - j|$ assegura que não existem rainhas na mesma diagonal. (Nota: Com esta formulação para as variáveis nunca podemos ter duas rainhas na mesma coluna.)

Exercício 6

Considere o problema do Sudoku para uma matriz 9×9, com 9 sub-matrizes 3×3. O objectivo é preencher todas as entradas com valores de 1 a 9, garantindo que cada número ocorre exactamente uma vez por linha, por coluna e por sub-matriz 3×3. Formule este problema como um CSP, indicando as variáveis (e respectivos domínios) e as restrições. Pode usar o operador AllDiff$(x_1, ..., x_n)$, que obriga a que todos os valores das variáveis $(x_1, ..., x_n)$ tenham valores diferentes.

Resolução

Variáveis: 81 variáveis designadas por $x_{i,j}$ com $0 \leq i, j \leq 8$.
Domínio da variáveis: [1..9].
A cada variável $x_{i.j}$ é atribuído o valor contido na posição $[i, j]$ da matriz.
Restrições:
Cada número ocorre exactamente uma vez por linha:
$\forall_{0 \leq i \leq 8}$ AllDiff$(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}, x_{i,5}, x_{i,6}, x_{i,7}, x_{i,8})$
Cada número ocorre exactamente uma vez por coluna:
$\forall_{0 \leq j \leq 8}$AllDiff$(x0, j, x1, j, x2, j, x3, j, x4, j, x5, j, x6, j, x7, j, x8, j)$
Cada número ocorre exactamente uma vez por sub-matriz:

$\forall_{i\in\{0,3,6\}}\forall_{j\in\{0,3,6\}}\text{AllDiff}(x_{i,j}, x_{i+1,j}, x_{i+2,j}, x_{i,j+1}, x_{i+1,j+1}, x_{i+2,j+1}, x_{i,j+2}, x_{i+1,j+2}, x_{i+2,j+2})$

Exercício 7

Considere um CSP com as seguintes características:
$V = \{x, y, z\}, Dx = \{0, 4, 6, 8\}, Dy = \{3, 6, 9\}, Dz = \{-1, 1, 2\}$.
Restrições:
R1: $x + z = y$
R2: $4x - y > 8$
R3: $2x/z = 4$
Realize as seguintes procuras usando ordem alfabética crescente para atribuir variáveis e ordem numérica crescente para atribuir valores.

a) Procura com retrocesso sem efectuar qualquer tipo de inferência.

Resolução

X=0
Y=3 ✗R2 6 ✗R2 9 ✗R2
X=4
Y=3
Z=-1 ✗R3 1 ✗R1 2 ✗R1
Y=6
Z=-1 ✗R1 1 ✗R1 2 ✓

Solução: X=4, Y=6, Z=2.

b) Procura com retrocesso utilizando o algoritmo olhar-para-a-frente (forward checking).

Resolução

X=0 $D_Y = \{\}$ R2
X=4 $D_Y = \{3, 6\}$ R2, $D_Z = \{2\}$ R3
Y=3 $D_Z = \{\}$ R1
Y=6 $D_Z = \{2\}$ R3
Z=2 ✓

c) Procura com retrocesso utilizando consistência de arcos (de acordo com o algoritmo AC-3).

Resolução

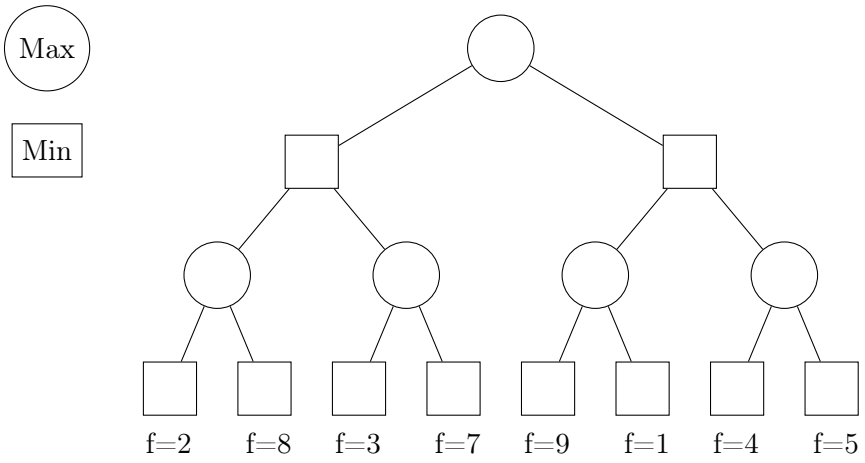
X=0 $D_Y = \{\}$ R2
X=4 $D_Y = \{6\}$ R1+R2, $D_Z = \{2\}$ R3
Y=6 $D_Z = \{2\}$ R3
Z=2 ✓

Capítulo 5

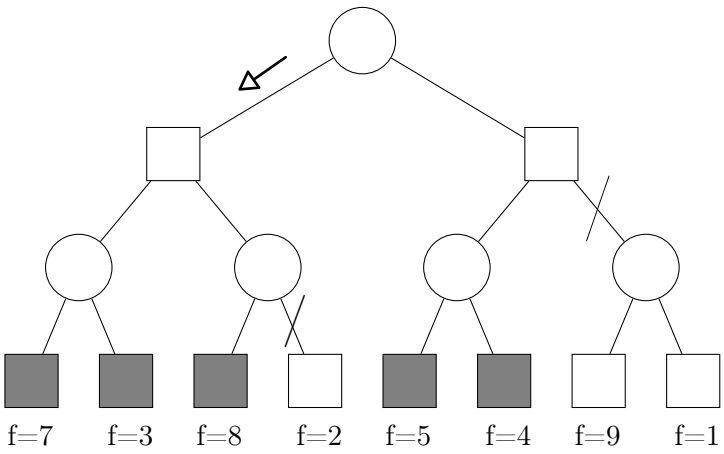
Procura Adversária e Jogos

Exercício 1

Considere a seguinte árvore de procura de dois agentes. Reordene as folhas de modo a maximizar o número de cortes com uma procura da esquerda para a direita. Assinale os cortes e a jogada escolhida. Escureça os nós terminais visitados. (Note que existe mais do que uma solução. Tem apenas que assinalar uma delas.)



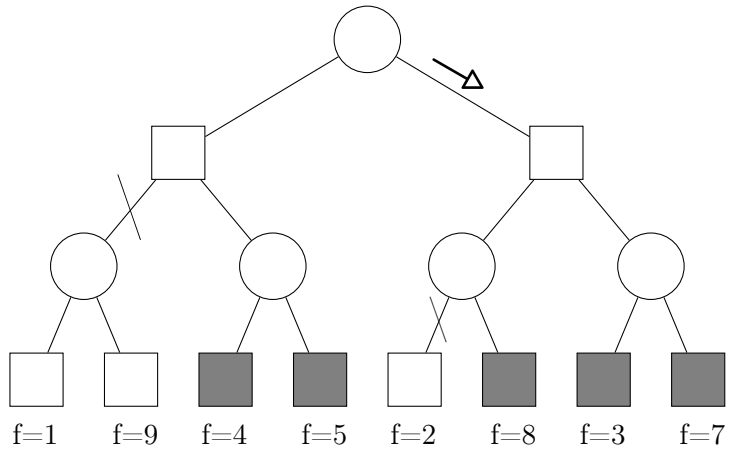
Resolução



Exercício 2

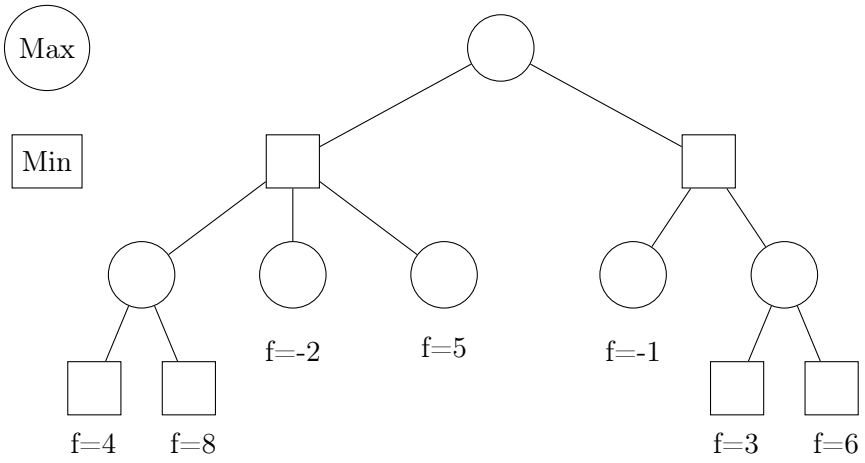
Considere a árvore de procura de dois agentes do exercício anterior. Reordene as folhas de modo a maximizar o número de cortes com uma procura da **direita para a esquerda**. Assinale os cortes e a jogada escolhida. Escureça os nós terminais visitados. (Note que existe mais do que uma solução. Tem apenas que assinalar uma delas.)

Resolução



Exercício 3

Considere a seguinte árvore de procura de dois agentes, em que estão assinalados os valores da função de avaliação para os estados terminais.

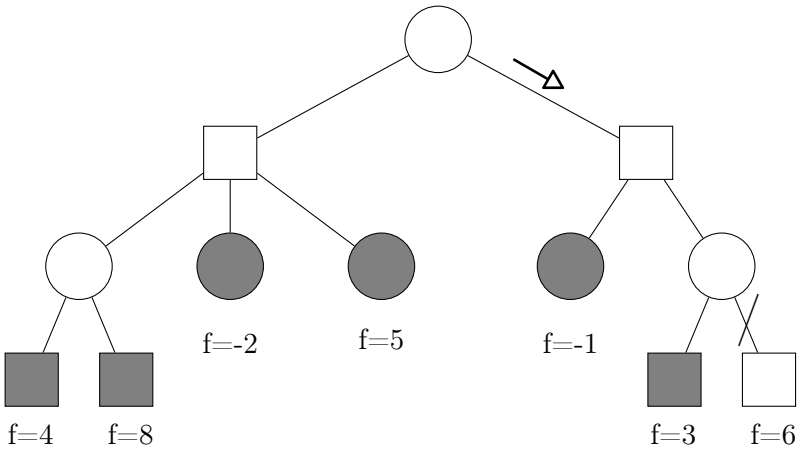


a) Efectue uma procura alfa-beta da esquerda para a direita, assinalando os cortes e a jogada escolhida.

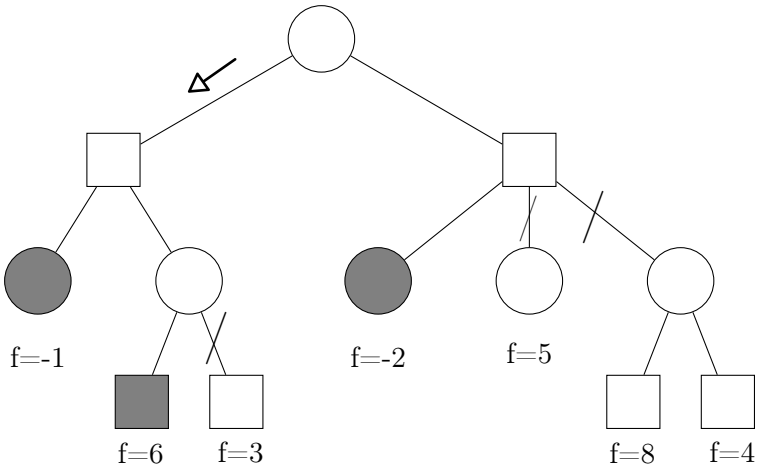
b) Desenhe a árvore anterior ordenando os sucessores de modo a maximizar o número de cortes ao efectuar uma procura alfa-beta da esquerda para a direita. Assinale os cortes e a jogada escolhida. (Note que existe mais do que uma solução. Tem apenas que apresentar uma delas.)

Resolução

a)

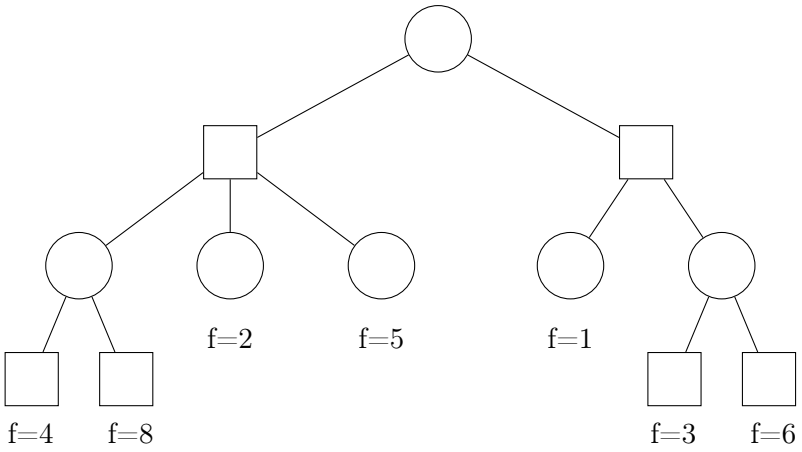


b)



Exercício 4

Considere a seguinte árvore de procura de dois agentes, em que estão assinalados os valores da função de avaliação para os estados terminais.



Ao aplicar o algoritmo Minimax com cortes alfa-beta da esquerda para a direita foram visitados os nós terminais que têm os seguintes valores de f:

i) 4, 2, 1

ii) 4, 2, 5, 1

iii) 4, 2, 5, 3, 6

iv) 4, 8, 1, 3

v) 4, 8, 1, 3, 6

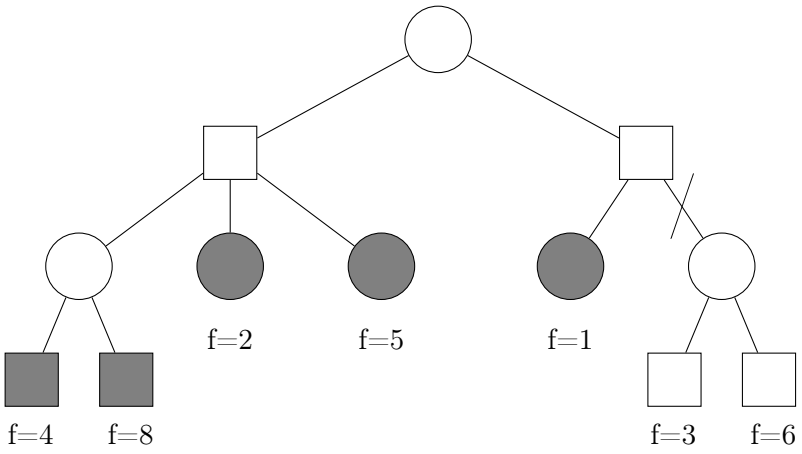
vi) 4, 8, 2, 5, 1

vii) 4, 8, 2, 5, 1, 3, 6

viii) 4, 8, 3, 6

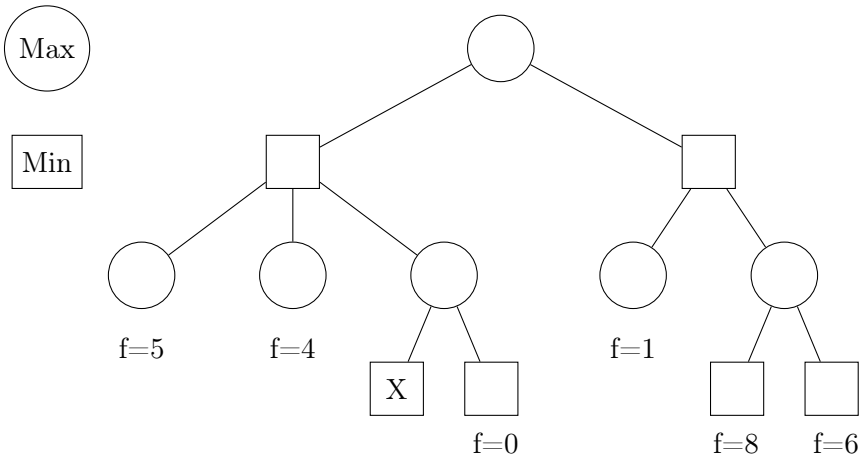
Resolução

vi) 4, 8, 2, 5, 1.



Exercício 5

Considere a seguinte árvore de procura de dois agentes, em que os valores f assinalados correspondem a valores da função de avaliação para estados terminais. O nó terminal assinalado com X não tem valor de f atribuído. Há um conjunto de valores que se for atribuído à função de avaliação deste nó minimiza o número de nós visitados durante a execução do algoritmo minimax alfa-beta da esquerda para a direita. Qual o menor destes valores?



i) -1

ii) 0

iii) 1

iv) 3

v) 4

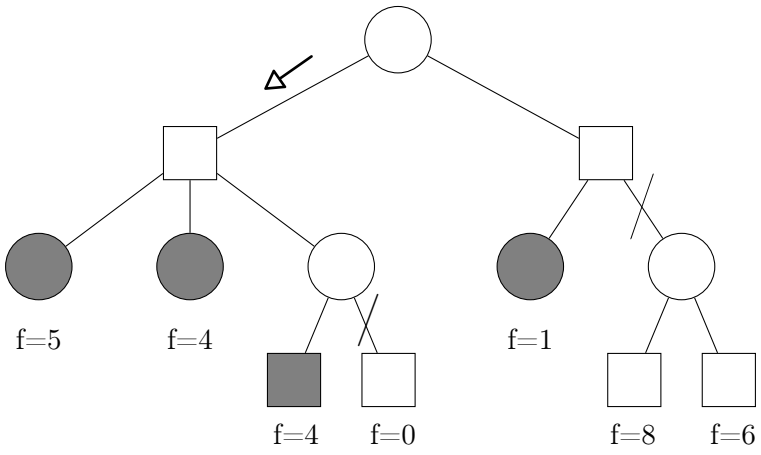
vi) 5

vii) 6

viii) 8

Resolução

v) 4.



Exercício 6

Este problema exercita os conceitos básicos de jogos, utilizando o Jogo do Galo [1]. Defina-se Xn como o número de linhas, colunas ou diagonais com exactamente nXs e nenhum O. Similarmente, On e o número de linhas, colunas ou diagonais com n Os. A função de utilidade atribui +1 a qualquer posição com X3=1 e -1 a qualquer posição com O3=1. Todas as outras posições terminais têm utilidade 0. Para posições não terminais, utilize a função de avaliação $Eval(s) = 3X2(s) + X1(s) - 3O2(s) - O1(s)$.

a) Aproximadamente, quantos jogos do galo diferentes existem?

b) Mostre toda a árvore de jogo começando com o tabuleiro vazio até a profundidade 2 (i.e., com uma jogada de X e outra de O), levando em consideração as simetrias.

c) Avalie todas as posições no nível 2.

d) Utilizando o algoritmo minimax, marque na sua árvore os valores retornados para as posições no nível 1 e 0, e utilize esses valores para escolher a melhor jogada inicial.

e) Assinale os nós à profundidade 2 que seriam avaliados se estivesse a ser executado o minimax com cortes alfa-beta, assumindo que os nós são gerados na ordem que optimiza os cortes.

Resolução

a) O Jogo do Galo pode acabar ao fim de cinco, seis, sete, oito ou nove jogadas: ou um jogador ganha ou o jogo termina em empate. Para acabar o jogo um jogador tem de colocar 3 X's/O's seguidos ou numa linha, coluna ou diagonal e há 8 possibilidades (3 linhas, 3 colunas e 2 diagonais). O primeiro (resp. segundo) jogador coloca X's (resp. O's) no tabuleiro. **Um jogo acaba nas seguintes possibilidades**:

   i) **Em 5 jogadas e o primeiro jogador ganha**. O jogador tem de colocar os X's seguidos nas 8 possibilidades de 3 X's seguidos não interessando a ordem (8*3!) e o segundo jogador coloca os O's nas outras casas de jogo não interessando a ordem (6*5). O que resulta em **8*3!*6*5 = 1440 possibilidades** do primeiro jogador ganhar ao fim de 5 jogadas.

   ii) **Em 6 jogadas e o segundo jogador ganha**. Tem de colocar os O's seguidos não interessando a ordem (8*3!) e o primeiro jogador coloca os X's nas outras casas de jogo não interessando a ordem (6*5*4). Logo existem **8*3!*6*5*4 = 5760 possibilidades**. Contudo temos de ter atenção e excluir o caso em que ficam 3 O's na mesma linha e os 3 X's também todos noutra linha uma vez que deste modo o jogo teria acabado ao fim de cinco jogadas. O mesmo se aplica às colunas. O caso de ficarem 3 O's seguidos numa diagonal impossibilita estarem 3 X's seguidos numa linha/coluna. Há 3 linhas e 3 colunas para colocar 3 O's seguidos não interessando a ordem (6*3!). Restam 2 colunas e duas linhas para os 3 X's seguidos não interessando a ordem (2*3!). Deste modo estas combinações em que o jogo teria acabado em 5 jogadas são **6*3!*2*3! = 432 casos**. Subtraindo estes casos ao número total de jogos com 6 jogadas obtemos **5760-432 = 5328 possibilidades** para o jogo acabar em 6 jogadas.

   iii) **Em 7 jogadas e o primeiro jogador ganha**. Com sete jogadas o jogo acaba com 4 X's no tabuleiro e 3 O's. Neste caso a ordem interessa pois o 4º X tem de ser colocado seguido a outros dois, daí resultam 8*3*6*3! possibilidades.

[1] https://pt.wikipedia.org/wiki/Jogo_do_Galo

Os 3 O's jogados pelo segundo jogador têm de ser colocados nos restantes 5 quadrados (5*4*3), logo existem **8*3*6*3!*5*4*3 = 51840 possibilidades**. Têm de ser retirados as combinações do jogo em que os 3 primeiros X's ou os 3 O's são colocados seguidos em jogo, caso contrário o jogo teria acabado antes de 7 jogadas **6*3*6*3!*3! = 3888 casos**. Deste modo existem **51840-3888 = 47952 possibilidades** para o jogo acabar em 7 jogadas.

iv) **Em 8 jogadas e o segundo jogador ganha**. Como dito anteriormente há 8 conjuntos de três posições seguida (3 linhas, 3 colunas e 2 diagonais). Interessa que o 4º O colocado pelo segundo jogador fique na sequência de 2 O's colocados anteriormente. Os X's não podem ficar seguidos. Portanto seguindo a lógica usada para quando são 7 jogadas, há um total de **8*3*6*3!*5*4*3*2 = 103680 possibilidades**. Excluindo os casos em que os 3 O's e 3 X's estão seguidos. Se uma linha tem O's só há duas possíveis linhas onde colocar um O e 2 possíveis lugares para um X. Deste modo excluimos **6*3*6*3!*2*4! = 31104 casos**. Chegamos a um total de **103680-31104 = 72576 possibilidades** para o jogo acabar em 8 jogadas.

v) **Em 9 jogadas, o primeiro jogador ganha ou o jogo acaba em empate**:

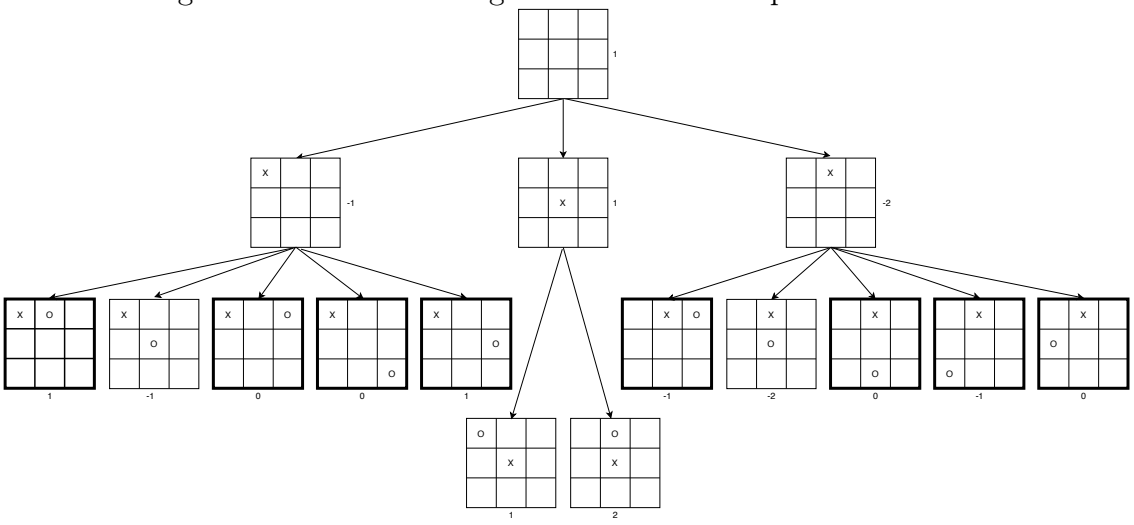
v.i) O primeiro ganha colocando o último X seguido de outros dois X's colocados anteriormente. Há três maneiras de colocar 3 X's seguidos no fim do jogo: numa diagonal, numa linha ou coluna. Ou numa casa que ao mesmo tempo completa uma linha e coluna. **No que toca às diagonais**, há duas diagonais e o quinto X tem de ser colocado na diagonal o que obriga os outros 2 X's que não fazem parte da solução a estarem nos 8 dos 15 pares de posições fora da diagonal, o que resulta em **2*3*8*4!*4! = 27648 possibilidades**. **Relativamente a uma linha ou coluna**, os outros 2 X's têm de ficar nos restantes 10 de 15 pares que não fazem parte dessa linha/coluna. Apenas 4 pares dos 10 impedem 3 O's de ficarem em fila, o que resulta em **6*3*4*4!*4! = 41472 possibilidades**. Por último relativamente a intersectar duas linhas/colunas/diagonais ao mesmo tempo há 22 pares onde há a possibilidade de uma intersecção com o 5º X. Desde modo temos **22*1*4!*4! = 12672 possibilidades**. Com um total de **27648+41472+12672 = 81792 possibilidade do primeiro jogador ganhar** ao fim de 9 jogadas.

v.ii) O jogo acaba em empate quando o último X é colocado de maneira a que não fica colocado em linha com outros 2 X's. Há 16 padrões em que colocados os 5 X's e os 4 O's não ficam 3 símbolos seguidos iguais em linha. Não interessando a ordem. Existem **16*5!*4!=46080 possibilidades de empate** ao fim de 9 jogadas.

Logo existem **81792+46080 = 127872 possibilidades** para o jogo acabar em 9 jogadas.

Concluindo, somando todas as possibilidades apresentadas anteriormente (com 5, 6, 7, 8 e 9 jogadas) sabemos que existe **um total de 1440+5328+47952+72576+127872 = 255168 Jogos do Galo possíveis**.

Figura 5.1: Árvore do Jogo do Galo até uma profundidade 2.



b) A árvore na Figura 5.1 mostra a árvore de jogo até uma profundidade de 2.

c) As avaliações estão anotadas em baixo das folhas da árvore na Figura 5.1.

d) Usando a função de avaliação fornecida no enunciado. Os valores estão anotados do lado direito dos nós da árvore na Figura 5.1. A melhor jogada será colocar o X na casa do meio.

e) Nós da árvore na Figura 5.1 que têm limites destacados não seriam avaliados usando cortes alpha-beta e uma ordenação óptima dos nós.

Exercício 7

Considere o jogo de dois jogadores, A e B, descrito abaixo.

A			**B**
1	2	3	4

Na figura os dois jogadores A e B encontram-se nas suas posições iniciais. O jogador A move-se primeiro. Os dois jogadores jogam alternadamente e cada jogador move a sua peça para uma posição adjacente livre em qualquer direcção. Se o oponente ocupar uma posição adjacente, o jogador pode pular para a próxima posição livre, se a houver. Por exemplo, se A estiver em 3 e B em 2, então A pode pular para a posição 1. O jogo acaba quando um jogador alcançar a posição oposta do tabuleiro. Se o jogador A atingir a posição 4 primeiro, então o valor para A é 1. Se B atingir a posição 1 primeiro, então o valor para A é -1.

a) Desenhe a árvore do jogo completa, utilizando a seguinte convenção:

   i) Escreva cada estado como (sA, sB), onde sA e sB representam as localizações de A e B.

   ii) Ponha cada estado terminal numa caixa rectangular e escreva o seu valor no jogo num círculo.

iii) Ponha os estados repetidos (estados que aparecem no caminho para a raiz) numa caixa rectangular dupla. Já que não é claro o valor a atribuir a valores de estados repetidos, marque um ponto de interrogação (?) dentro de um círculo.

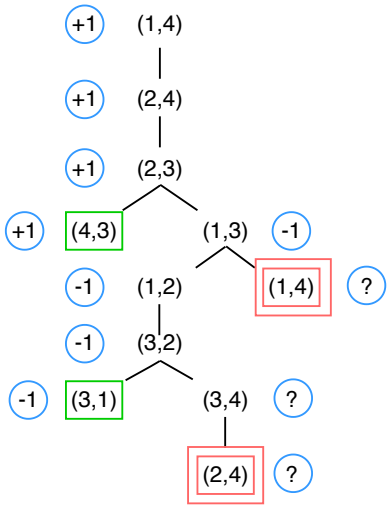
b) Marque cada nó com o valor minimax (também num circulo). Explique como tratou o ponto de interrogação e porquê.

c) Explique porque o algoritmo minimax falharia neste jogo e sugira alterações para corrigir. O seu algoritmo modificado dá as decisões óptimas para todos os jogos com estados repetidos?

d) Este jogo pode ser generalizado para n quadrados com n>2. Prove que A ganha se n for ímpar e perde se n for par.

Resolução

Figura 5.2: Árvore de Jogo para o jogo de dois jogadores A e B e quatro quadrados.



a) A Figura 5.2 mostra a árvore de jogo. Cada estado terminal numa caixa rectangular verde com o respectivo valor do jogo no circulo azul ao lado. Os estados repetidos estão em caixas rectangulares duplas vermelhas com um valor de (?).

b) Na Figura 5.2 todos os nós têm anotado o valor minimax num circulo azul ao lado de cada nó. Tendo duas escolhas entre um valor (-1 ou +1) e (?) o agente escolhe sempre acabar o jogo (-1 ou +1).

c) O algoritmo de minimax falharia neste jogo pois trata-se de uma procura em profundidade (DFS) e entraria num loop infinito. Uma possibilidade seria guardar os estados já visitados e ao encontrar um estado repetido atribuímos o valor de (?) e tratamos este valor como explicado na alínea anterior.

Para este caso em especifico funciona mas para todos os jogos com casos repetidos não funcionaria pois não é claro como comparar o valor de (?) com uma posição de empate. Ou se existirem vitórias com diferentes valores como no jogo Backgammon.

Outra possibilidade para uma árvore de jogo com ciclos é usar um método de programação dinâmica.

d) Provemos por indução no tamanho do tabuleiro do jogo (n), $[1, \ldots, n]$. Claramente no tamanho mínimo ($n = 3$) o jogador A perde sempre. No caso de $n = 4$ é uma

vitória para A. Generalizando para $n > 4$, A e B jogam alternadamente na direcção oposta à sua casa inicial. Deste modo ao fim de duas jogadas, 1ª de A e 1ª de B, podemos ver o jogo como um subjogo de dimensão $n - 2$ com um tabuleiro de $[2, \ldots, n-1]$. Com a diferença que agora há a opção de voltar para trás. Ignorando por agora esta opção, é fácil de verificar que se A ganha o jogo "$n-2$" significa por definição do jogo que A chega à posição $n - 1$ antes de B chegar à posição 2. E por este motivo A chega a n antes de B chegar a 1. Seguindo a mesma lógica caso seja B a ganha o jogo $n - 2$ significa que B ganha o jogo n. Se n for um número par o jogador A ganha, caso contrário o jogador B ganha. Tendo em conta agora as jogadas de voltar para trás. O jogador vencedor nunca volta para trás. Caso o jogador perdedor volte para trás é fácil de verificar que continua a perder o jogo. O jogador vencedor nunca volta para trás, mesmo que o jogador perdedor salte por cima do jogador vencedor, o jogador perdedor não ganha duas jogadas visto que perdeu uma jogada a andar para trás.

Capítulo 6

Planeamento Automático

6.1 Modelação

Exercício 1 - Compras

Considere o domínio das Compras. O objetivo é executar uma sequência de ações que permitam que uma pessoa compre uma lista de produtos. Considere as seguintes ações:

A1: Comprar($item$, $loja$)

- Pré-condição: Em($loja$) $\wedge$ Vende($item$, $loja$)
- Efeito: Tem($item$)

A2: Ir(x, y)

- Pré-condição: Em(x)
- Efeito: _ _ _ _ _ _ _

a) Explique o funcionamento de A1 e A2. Qual deve ser o efeito de A2?

Resolução

A1: Se a pessoa está num lugar que vende um dado produto, é possível comprá-lo, resultando em que a pessoa tenha o produto.
A2: Para ir do lugar x ao y, a pessoa necessita de estar em x. Como resultado, a pessoa não está mais em x, mas sim em y. Formalmente, o efeito será Em(y) $\wedge$ not(Em(x)).

b) Suponha que uma pessoa está em casa. Um supermercado vende leite e bananas, e uma loja de ferramentas vende berbequins. O objetivo é que a pessoa tenha leite, uma banana, e um berbequim. Formalize os estados inicial e objetivo.

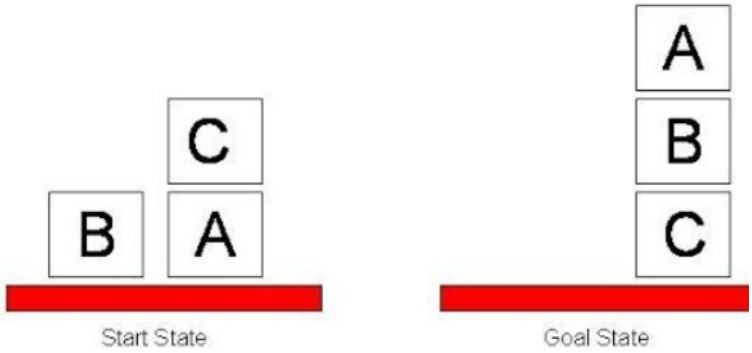
Resolução

Inicial: Em(casa) $\wedge$ Vende(SM, leite) $\wedge$ Vende(SM, banana) $\wedge$ Vende(LF, berbequim)
Objetivo: Tem(leite) $\wedge$ Tem(banana) $\wedge$ Tem(berbequim)

Exercício 2 - Mundo dos Blocos

Considere o Mundo dos Blocos. O objetivo é construir uma ou mais torres verticais de blocos. Apenas um bloco pode ser movido de cada vez: pode ser posto na mesa, ou em cima de outro

bloco. Desta forma, qualquer bloco que esteja debaixo de outro, não pode ser movido nessas condições.
a) Formalize este domínio, e crie uma instância para os estados inicial e objetivo da figura seguinte:



Resolução

Planning.Domains

b) Considere a seguinte formalização:

OP : (ACTION : Move(b,x,y),
PRECOND : On(b,x) ∧ Clear(b) ∧ Clear(y),
EFFECT: On(b,y) ∧ Clear(x) ∧ ¬On(b,x) ∧ ¬Clear(y))

OP : (ACTION : MoveToTable(b,x),
PRECOND : On(b,x) ∧ Clear(b),
EFFECT: On(b,TABLE) ∧ Clear(x) ∧ ¬On(b,x))

Explique o propósito de cada uma das ações. A ação *MoveToTable* é necessária?

Resolução

A ação Move, pega no bloco b que está em cima de x e põe-no em cima de y. Assim, a pré-condição On(b,x) diz que b está em cima de x, e as pré condições Clear dizem que nem b nem y têm nada em cima, para ser possível pegar em b e por b em cima de y, respetivamente. O efeito é então ter b em cima de y (On(b,y)) e b deixa de estar em cima de x (not On(b,x)), logo y deixa de estar sem nada em cima (not Clear(y)) e x passa a não ter nada em cima (clear(x)) Esta ação Move não pode ser a única ação, porque se for temos problemas com a definição da mesa (table) estar ou não vazia. Se dissermos que a mesa está sempre vazia, já que queremos por lá blocos para chegar ao objetivo, da primeira vez que utilizarmos a ação Move para por um bloco em cima da mesa, então vamos ter que not Clear(table), e já não podemos por lá mais nada. Assim, temos a ação MoveToTable, que dado um bloco b que esteja em cima de x, o move para a mesa. Para a aplicar é necessário que o bloco b esteja em cima de x (On(b,x)) e que b não tenha nada em cima dele para o podermos mover (Clear(b)). Como efeito, b passa a estar em cima da mesa (On(b, TABLE), x deixa de ter algo em cima dele pois b saiu, logo b deixa de estar em x (Clear(x) e not On(b,x)).

Exercício 3 - Pasta

Considere o mundo da Pasta. O objetivo é transportar um dado número de objetos da sua posição inicial para a posição objetivo. Cada localização é acessível por qualquer uma das

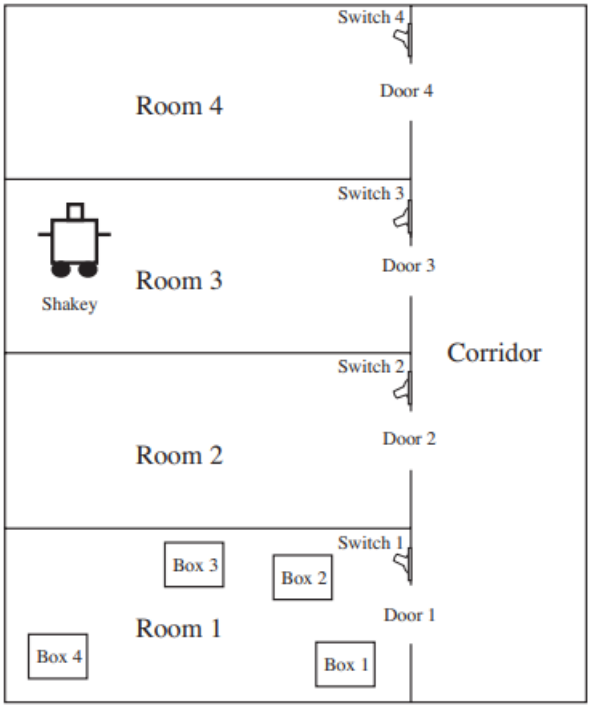
outras. Os objetos podem ser postos ou retirados de uma pasta. Quando a pasta se move entre posições, todos os objetos dentro da pasta são também movidos. Modele este domínio, e crie uma instância para 5 objetos $(o_0, ..., o_4)$ e 6 localizações $(l_0, ..., l5)$. A posição inicial dos objetos é: (o_0, l_1), (o_1, l_2), (o_2, l_3), (o_3, l_0), (o_4, l_4), sendo o objetivo: (o_0, l_0), (o_1, l_1), (o_2, l_4), (o_3, l_3), (o_4, l_5). A posição final da pasta deve ser l_0.

Resolução

Planning.Domains

Exercício 4 - Mundo do Shakey

Considere o Mundo do Shakey, representado na figura seguinte:



Consiste em 4 salas alinhadas ao longo de um corredor, em que cada sala tem uma porta e um interruptor de luz. As ações neste mundo consistem em mover entre lugares, arrastar objetos móveis (e.g., caixas), subir para cima de objetos rigidos (e.g., caixas), e ligar ou desligar os interruptores;

- $Go(x, y, r)$, que requer que o Shakey esteja em x $(At(x))$, e que x e y sejam lugares na mesma sala $(In(., r))$. Considere que uma porta entre duas salas está em ambas.
- Arrastar a caixa b do lugar x para y dentro do mesmo lugar: $Push(b, x, y, z)$. O predicado Box pode ser utilizado para ditar se uma constante é ou não uma caixa.
- Subir para uma caixa da posição x: $ClimbUp(x, b)$; Descer de uma caixa para a posição x: $ClimbDown(b, x)$. Sugestão: considerar o predicado On e a constante $floor$.
- Ligar ou desligar um interruptor: $TurnOn(s, b)$, $TurnOff(s, b)$. Para ligar ou desligar, o Shakey deve estar em cima de uma caixa na posição do interruptor.

  Formalize estas 6 ações, e o estado inicial mostrado na figura acima. Escreva um plano para o Shakey por a Box_2 em $Room_2$.

Resolução

Ações:

$Action(\text{ACTION:}Go(x,y), \text{PRECOND:}At(Shakey,x) \wedge In(x,r) \wedge In(y,r),$
$\quad \text{EFFECT:}At(Shakey,y) \wedge \neg(At(Shakey,x)))$
$Action(\text{ACTION:}Push(b,x,y), \text{PRECOND:}At(Shakey,x) \wedge Pushable(b),$
$\quad \text{EFFECT:}At(b,y) \wedge At(Shakey,y) \wedge \neg At(b,x) \wedge \neg At(Shakey,x))$
$Action(\text{ACTION:}ClimbUp(b), \text{PRECOND:}At(Shakey,x) \wedge At(b,x) \wedge Climbable(b),$
$\quad \text{EFFECT:}On(Shakey,b) \wedge \neg On(Shakey,Floor))$
$Action(\text{ACTION:}ClimbDown(b), \text{PRECOND:}On(Shakey,b),$
$\quad \text{EFFECT:}On(Shakey,Floor) \wedge \neg On(Shakey,b))$
$Action(\text{ACTION:}TurnOn(l), \text{PRECOND:}On(Shakey,b) \wedge At(Shakey,x) \wedge At(l,x),$
$\quad \text{EFFECT:}TurnedOn(l))$
$Action(\text{ACTION:}TurnOff(l), \text{PRECOND:}On(Shakey,b) \wedge At(Shakey,x) \wedge At(l,x),$
$\quad \text{EFFECT:}\neg TurnedOn(l))$

Estado Inicial:

$In(Switch_1,Room_1) \wedge In(Door_1,Room_1) \wedge In(Door_1,Corridor)$
$In(Switch_1,Room_2) \wedge In(Door_2,Room_2) \wedge In(Door_2,Corridor)$
$In(Switch_1,Room_3) \wedge In(Door_3,Room_3) \wedge In(Door_3,Corridor)$
$In(Switch_1,Room_4) \wedge In(Door_4,Room_4) \wedge In(Door_4,Corridor)$
$In(Shakey,Room_3) \wedge At(Shakey,X_S)$
$In(Box_1,Room_1) \wedge In(Box_2,Room_1) \wedge In(Box_3,Room_1) \wedge In(Box_4,Room_1)$
$Climbable(Box_1) \wedge Climbable(Box_2) \wedge Climbable(Box_3) \wedge Climbable(Box_4)$
$Pushable(Box_1) \wedge Pushable(Box_2) \wedge Pushable(Box_3) \wedge Pushable(Box_4)$
$At(Box_1,X_1) \wedge At(Box_2,X_2) \wedge At(Box_3,X_3) \wedge At(Box_4,X_4)$
$TurnwdOn(Switch_1) \wedge TurnedOn(Switch_4)$

Plano:

$Go(X_S,Door_3)$
$Go(Door_3,Door_1)$
$Go(Door_1,X_2)$
$Push(Box_2,X_2,Door_1)$
$Push(Box_2,Door_1,Door_2)$
$Push(Box_2,Door_2,Switch_2)$

6.2 Algoritmos

Exercício 1

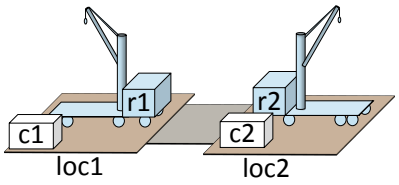
Considere um problema de planeamento envolvendo dois robots, cujas ações são controladas por um único agente. Na formalização seguinte, r é um robot, l e m são locais, e c é um contentor. Os dois robots podem ter a mesma localização.

take(r, l, c)
  pre: loc$(r) = l$, pos$(c) = l$,
    cargo$(r) =$ nil
  eff: cargo$(r) = c$, pos$(c) \leftarrow r$

put(r, l, c)
  pre: loc$(r) = l$, pos$(c) = r$
  eff: cargo$(r) \leftarrow$ nil, pos$(c) \leftarrow l$

move(r, l, m)
  pre: loc$(r) = l$
  eff: loc$(r) \leftarrow m$

(a) action templates



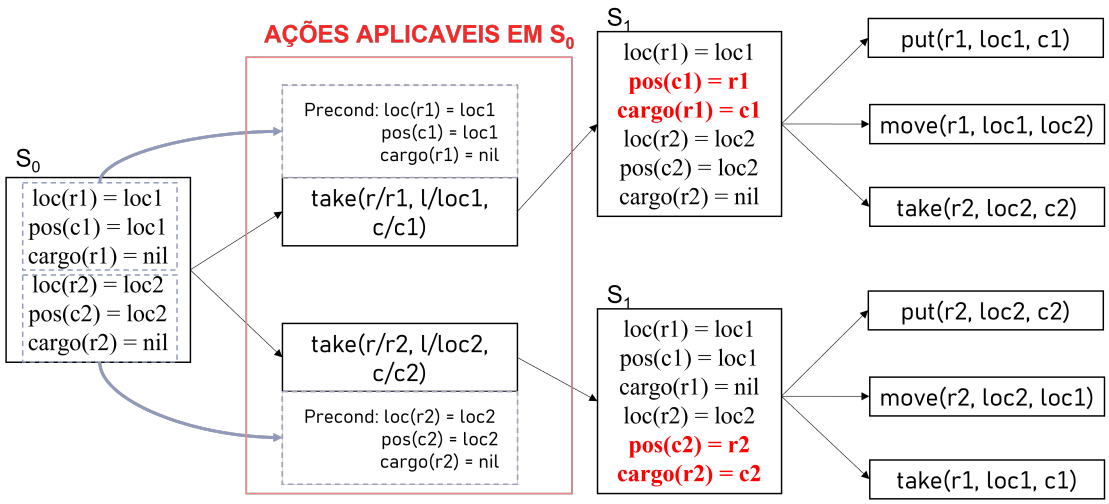
$s_0 = \{$loc(r1) $=$ loc1, loc(r2) $=$ loc2,
cargo(r1) $=$ nil, cargo(r2) $=$ nil,
pos(c1) $=$ loc1, pos(c2) $=$ loc2$\}$

$g = \{$pos(c1) $=$ loc2, pos(c2) $=$ loc1$\}$

(b) initial state and goal

a) Execute *forward-search* neste problema.Quantas iterações são necessárias para o tempo de execução mais curto? E quais o planos gerados?

Resolução



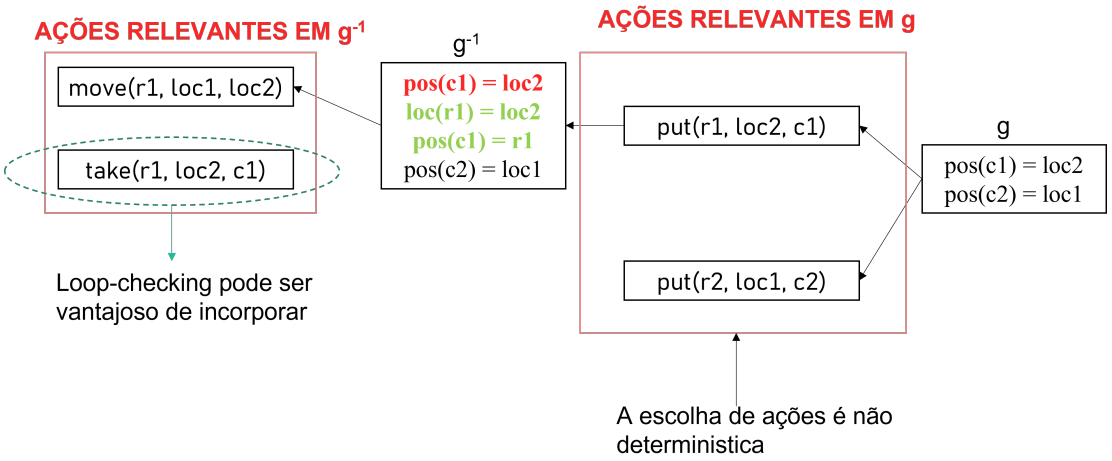
O estado $S_1 = S_0 - DEL(a) \cup ADD(a)$, em que a é a ação aplicavel em S_0. Na imagem os efeitos da ação estão a vermelho. Note que este é apenas o passo 1 do algoritmo. O algoritmo continua até que o S_n seja o estado objectivo.

A solução pode ser encontrada em 6 passos.

Exemplo plano possivel em 6 passos: [take(r1, loc1, c1), move(r1, loc1, loc2), put(r1, loc2, c1), take(r2, loc2, c2), move(r2, loc2, loc1), put(r2, loc1, c2)]

b) Execute *backward-search* neste problema. Quantas iterações são necessárias para o tempo de execução mais curto? E quais o planos gerados?

Resolução



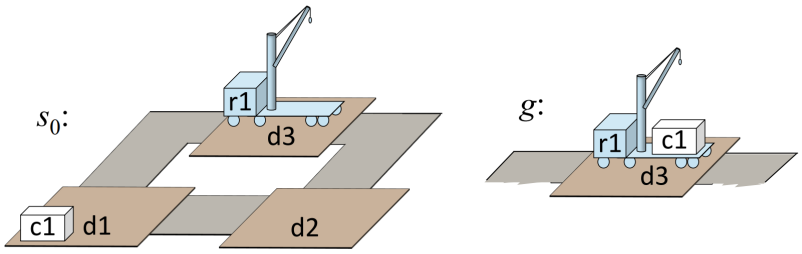
Uma regressão de g através da ação a, put(r1, loc2, c1), dá-nos um estado $g^{-}1$ em que os literais positivos e negativos de $g^{-}1$ são obtidos da seguinte forma:

POS($g^{-}1$) = (POS(g) - ADD(a)) ∪ POS(precond(a))
NEG($g^{-}1$) = (NEG(g) - DEL(a)) ∪ NEG(precond(a))

A vermelho o literal que é RETIRADO a verde os literais ADICIONADOS.
O algoritmo termina quando o estado inicial é atingido. A solução pode ser encontrada em 6 passos, 7 iterações do algoritmo.

Exercício 2

Considere uma simplificação do problema anterior em que existe apenas um robot, um contentor, 3 locais de descarga e não existem pilhas de blocos. Na formalização seguinte, r é um robot, l e m são locais, e c é um contentor.

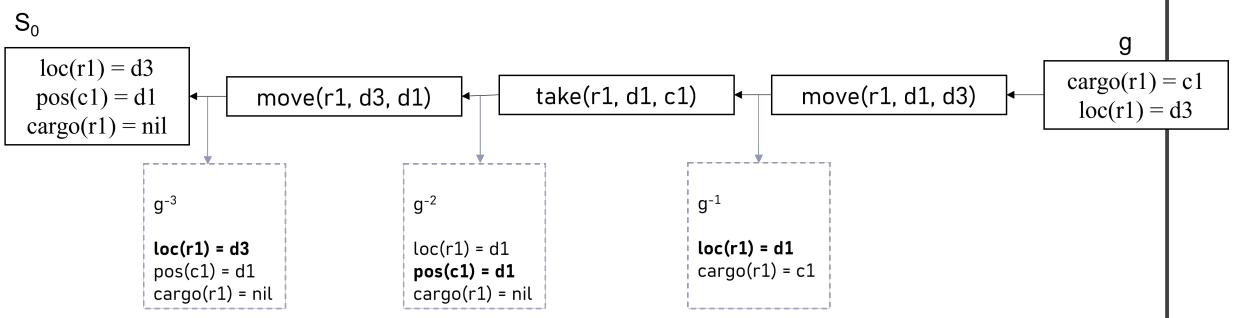


a) Qual o plano de acordo com o algoritmo *backward-search* que satisfaz o objectivo em 4 iterações?

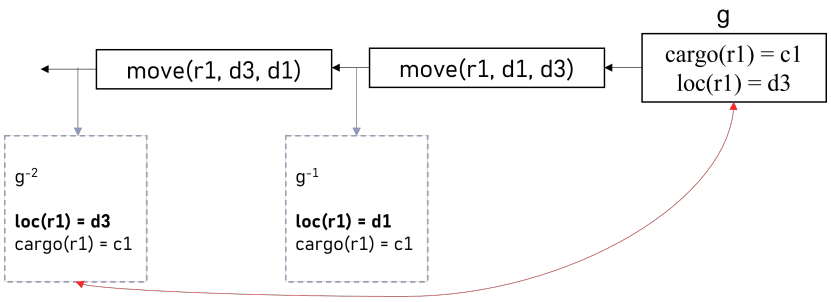
b) Suponha que incorporamos a deteção de loops no algoritmo de *backward-search*, que verifica se o estado que uma ação consegue atingir já foi atingido anteriormente e em caso afirmativo o algoritmo retorna *Falha*. Dado o estado objectivo g = {cargo(r1) = c1, loc(r1) = d3}, considere que move(r1, d1, d3), move(r1, d2, d3) e load(r1, c1, d3) são as 3 acções relevantes que precedem o estado g. Suponha que a escolha não deterministica do algoritmo *backward-search* é move(r1, d1, d3). No loop seguinte que acção retorna *Falha*?

Resolução

a)



b) Após a aplicação da move(r1, d1, d3) existem duas ações relevantes: move(r1,d3,d1) e take(r1, d1, c1). Vamos assumir que o algoritmo escolhe move(r1,d3,d1). Nesse caso vai retornar *Falha*, pois o estado resultante já foi atingido anteriormente (ver imagem em baixo).



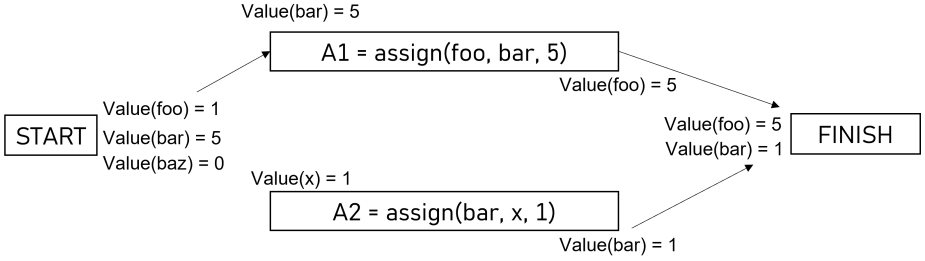
Exercício 3

Considere o problema da troca de valores entre duas variáveis. Existe o seguinte conjunto de objetos: B = Variáveis ∪ Números, onde Variáveis = {foo, bar, baz}, e Números = {0, 1, 2, 3, 4, 5}. Existe apenas uma ação:

assign(x_1, x_2, n)

- Pré-condição: value(x_2) = n
- Efeito: value(x_1) ← n

Os estados inicial e objetivo são:
s_0 ={value(foo)=1,value(bar)=5,value(baz)=0};
g = {value(foo) = 5, value(bar) = 1}.
A próxima figura mostra um plano parcial para este problema.



a) O algoritmo POP permite encontrar um plano parcial que resolva um dado problema de planeamento. Este algoritmo consiste em realizar uma procura dentro de um espaço de planos. Que operadores estão disponíveis neste tipo de procura?

b) Mostre a sequencia de resolventes que atinge o plano parcial acima.

c) Começando neste plano parcial, encontre um plano parcialmente ordenado que seja solução para o problema.

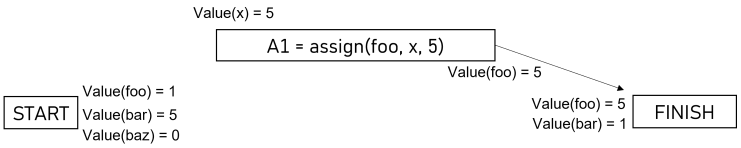
Resolução

a)

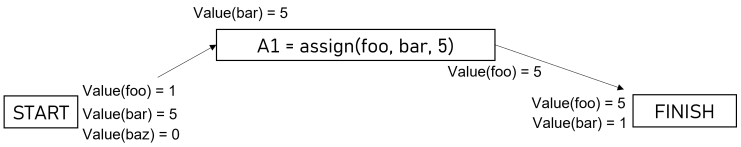
- Adicionar novas ações ao plano;
- Introduzir restrições de ordem;
- Atribuir valores a variáveis.

b) A procura começa com o plano inicial que contém uma ação Start e os seus efeitos. A ação Finish tem um conjunto de pré-condições que são adicionadas à lista de pré-condições abertas. O algoritmo acaba quando as lista de pré-condições abertas é vazia

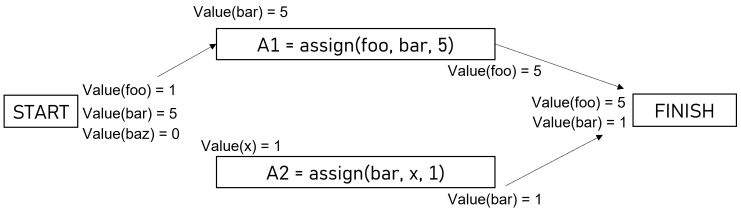
Escolher a primeira pré-condição da lista: Value(foo)=5 e aplicar a ação a1. pré-condições abertas = Value(x) = 5, Value(bar)=1



Escolher a pré-condiçao da lista Value(x)=5. Fazer atribuição x=bar, ligação causal de start. pré-condições abertas = Value(bar)=1

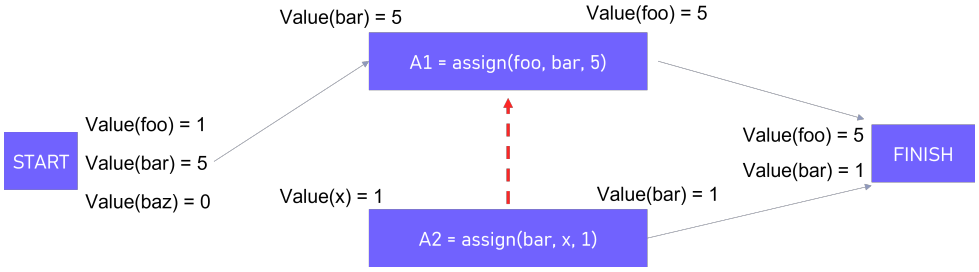


Escolher a pré-condição Value(bar)=1 e aplicar ação a2. pré-condições abertas =

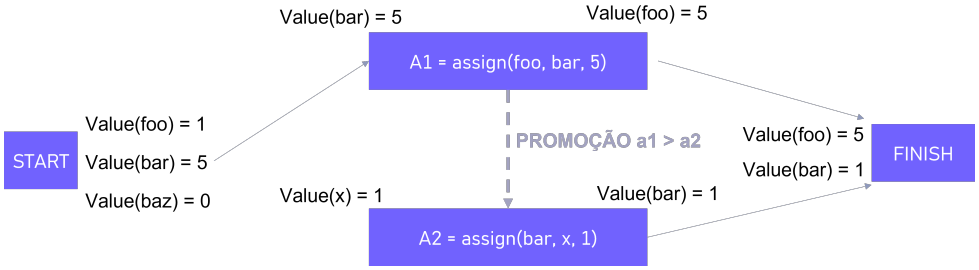


Chegou-se ao plano parcial do enunciado, como pedido.

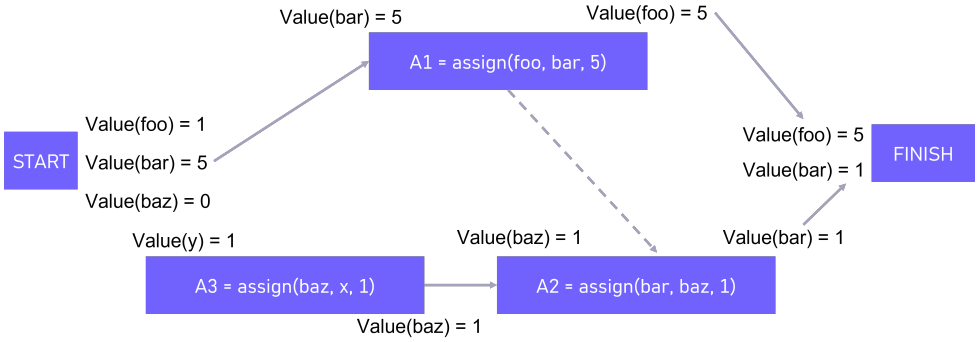
c)
pré-condições abertas = Value(x)=1
Conflito: Se A2 acontecer antes de A1, efeito de a2 põe em causa a ligação causal entre start e A1. Conflito representado pela seta vermelha.



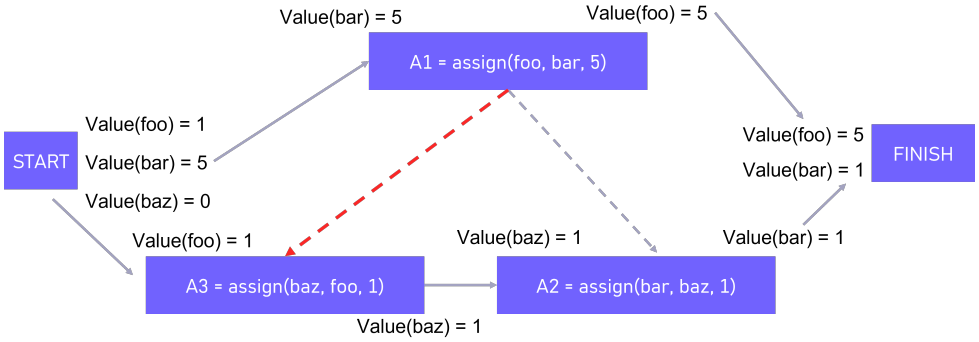
Conflitos são resolvidos com restrições de ordem.



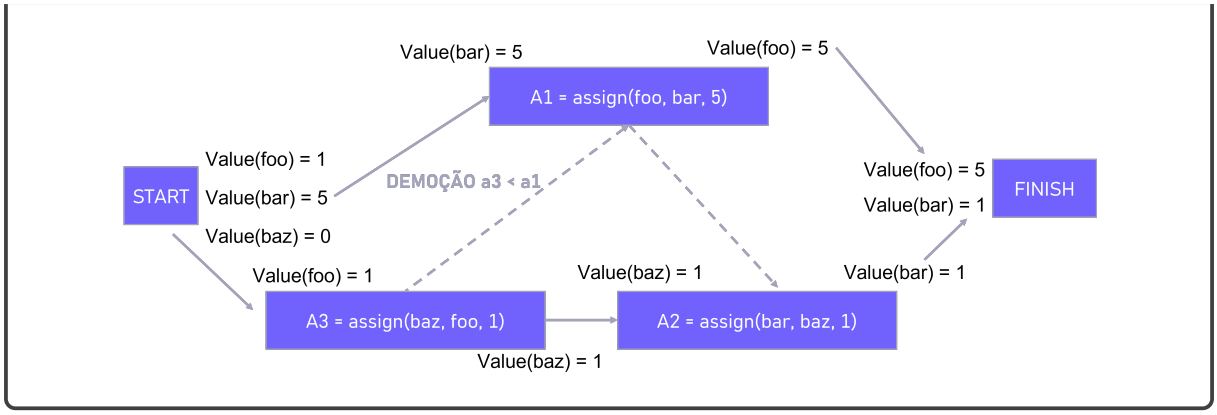
Escolher a pré-condição da lista. Aplicar ação a3 + atribuição.



Escolher pré-condição aberta Value(y)=1. Criar ligação ligação causal ao estado Start (pois esta pre-condição pode ser atingida através da ação Start) e fazer atribuição. Esta atribuição resulta em conflito entre of efeitos de A1 e a pre-condição de A3.



Conflitos são resolvidos com restrições de ordem.



Exercício 4

Considere o algoritmo POP e o problema de planeamento com a formalização:
A1:

- Pré-condição: A ∧ C
- Efeito: not(A)∧ B∧ not(C)∧ not(D)

A2:

- Pré-condição: A
- Efeito: not(A)∧ B∧ not(D)

A3:

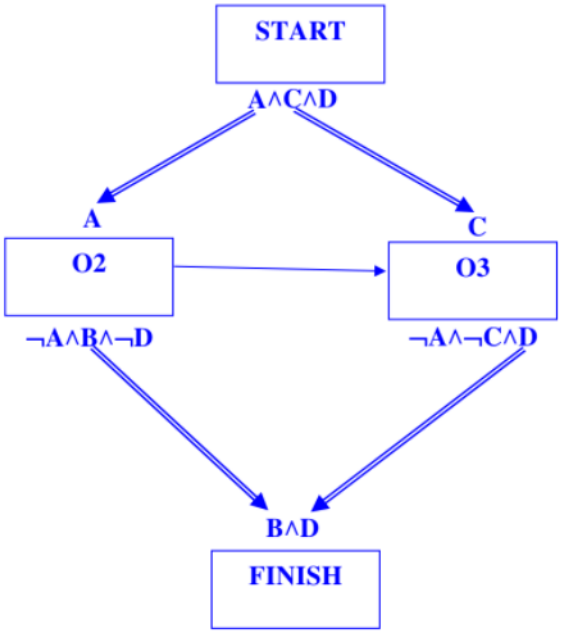
- Pré-condição: C
- Efeito: not(A)∧ not(C)∧ D

Estado Inicial: A ∧ C ∧ D
Estado Objetivo: B ∧ D

Encontre o plano parcialmente ordenado que é solução para este problema.

Resolução

Resposta:



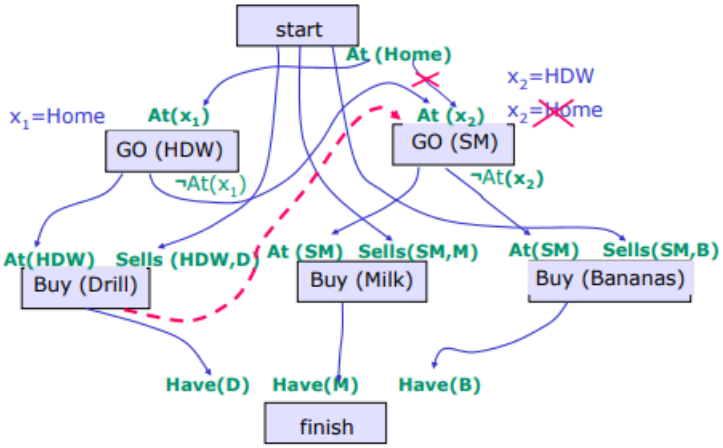
Depois de introduzir a ligação causal START$\xrightarrow{A}$O2, se se introduzir a acção O3 o efeito ¬A de O3 é uma ameaça para esta ligação causal. Para resolver esta ameça promove-se O3 introduzindo a restrição de ordem que obriga a que O2 se realize antes de O3 (seta simples entre O2 e O3).

Nota: setas duplas - ligação causal; setas simples - ordenação

Exercício 5

Considere o Domínio das Compras, formalizado no exercício 1 da Secção 1.1.
a) Encontre um plano parcialmente ordenado que resolva o problema.

Resolução



Setas azuis são ligações causais que também implicam restrições de ordem. Setas vermelhas indicam apenas restrições de ordem.
Resolução completa: https://ocw.mit.edu/courses/6-825-techniques-in-artificial-intelligence-sma-5504-fall-2002/resources/lecture11finalpart1/ ou

https://people.cs.pitt.edu/ wiebe/courses/CS2710/lectures/curpartialOrderPlanning.pdf

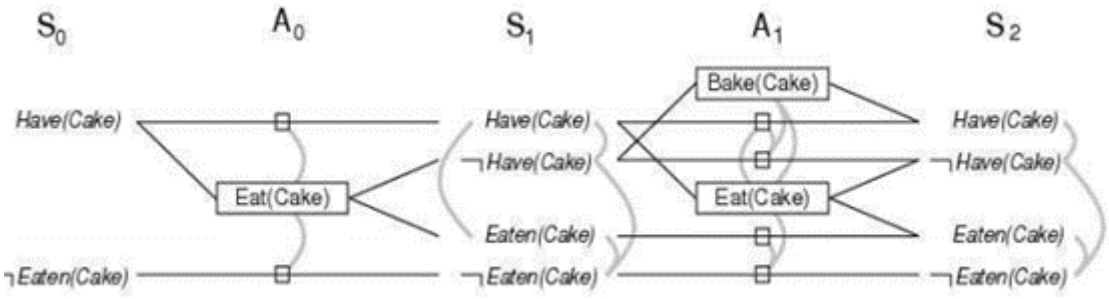
b)Indique, justificando, o número de planos mínimos.

Resolução

É possível escolher ir primeiro ao supermercado e depois à loja de ferramentas, ou o contrário. Estando no supermercado, é também possível comprar primeiro as bananas e depois o leite, ou o contrário. Todas estas hipóteses retornam um número mínimo de ações, pelo que há quatro planos mínimos.

Exercício 6

Considere o seguinte *planning graph*:



a) Explique o que são ligações mutex.

Resolução

Num grafo de ações, em cada secção Ai podemos ter representadas ações que, ao tempo i, são capazes de ter as suas pré-condições satisfeitas, isto é, que talvez ocorram em Si. Ligações mutex entre ações indicam que essas ações não podem ocorrer simultaneamente, pois têm efeitos inconsistentes.

b) Explique em que circunstancias é que uma ligação mutex é imposta entre duas ações. Para cada caso, identifique uma ligação mutex correspondente no grafo acima.

Resolução

Acontece em 3 cenários: 1) Efeitos inconsistentes: uma ação nega um dos literais no efeito da outra. Na figura, em A0, EatCake e a persistência de HaveCake são um exemplo. 2) Interferencia: Um dos efeitos de uma ação é a negação de um literal da precondição da outra. Na figura, em A0, EatCake nega HaveCake que é a precondição à persistência de HaveCake. 3) Necessidades em competição: Uma das precondições de uma ação é mutualmente exclusiva com uma precondição da outra. Na figura, em A1, EatCake e BakeCake competem pelo valor de HaveCake na precondição.

c) Explique em que circunstancias é que uma ligação mutex é imposta entre dois literais. Para cada caso, identifique uma ligação mutex correspondente no grafo acima.

Resolução

Acontece em cenários de suporte inconsistente: Se um literal é negação do outro ou se cada possível par de ações que consiga chegar aos dois literais é mutualmente exclusivo. Na figura, em S1, a única maneira de chegar a HaveCake tem relação mutex com a única maneira de chegar a EatenCake, logo os dois literais têm relação mutex nesta secção.

Exercício 7

Considere o seguinte problema de planeamento:

Predicados: Descansado(x), Satisfeito(x)
Estado Inicial: Descansado(João)
Estado Objetivo: Descansado(João) ∧ Satisfeito(João)
Ação: Corre(x)

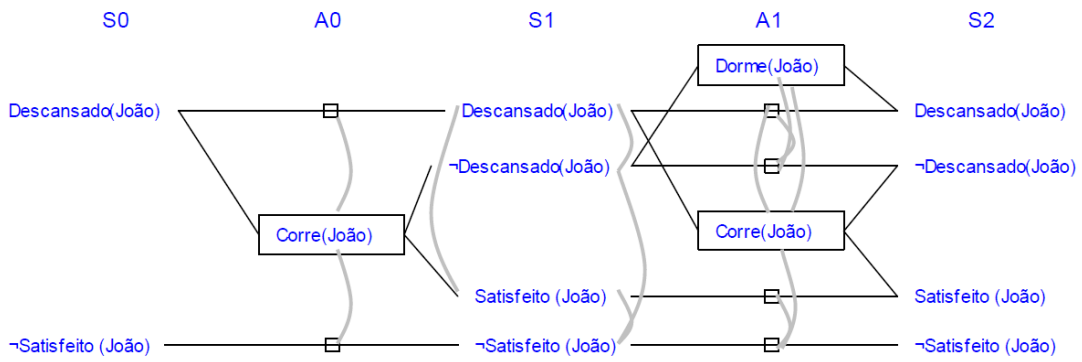
- Pré-condição: Descansado(x)
- Efeito: not(Descansado(x) ∧ Satisfeito(x)

Ação: Dorme(x)

- Pré-condição: not(Descansado(x))
- Efeito: Descansado(x)

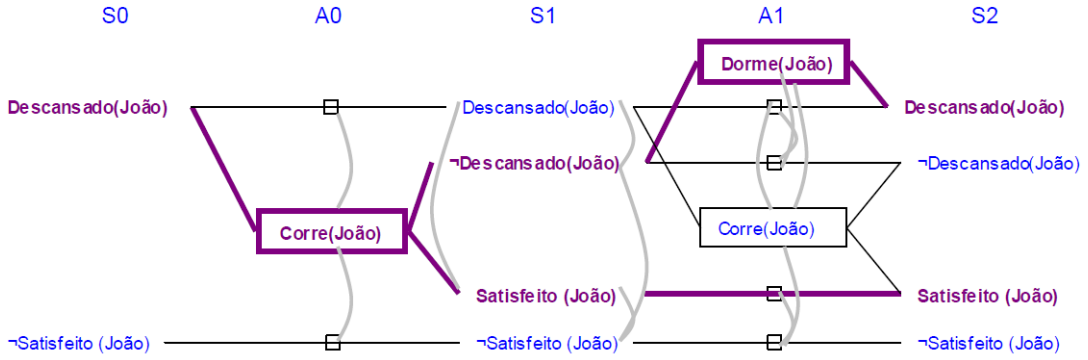
a) Desenhe o *planning graph* para este problema.

Resolução



b) Utilizando GRAPHPLAN e o *planning graph*, construa um plano que atinja o objetivo.

Resolução


S0
A0
S1
A1
S2
Descansado(João)
Corre(João)
¬Satisfeito (João)
Descansado(João)
¬Descansado(João)
Satisfeito (João)
¬Satisfeito (João)
Dorme(João)
Corre(João)
Descansado(João)
¬Descansado(João)
Satisfeito (João)
¬Satisfeito (João)

Capítulo 7

Aprendizagem por Reforço

O tipo de aprendizagem por reforço que abordamos na cadeira, assume que o problema se pode modelar através dos seguintes valores:

- Espaço de estados: $X = x_1, x_2, ..., x_n$
- Espaço de acções: $A = a_1, a_2, ..., a_m$
- Função de *reward*: $R : X \times A \to \mathbb{R}$

A função $V(x)$ calcula a *reward* obtida após uma trajectória com T passos que começa no estado x. Se assumirmos γ como o factor de desconto, podemos calcular $V(x)$ da seguinte forma:

$$V(x) = \sum_{t=0}^{T-1} \gamma^t r_t \tag{7.1}$$

A função $Q : X \times A \to \mathbb{R}$, calcula o mesmo que V mas para um par (estado, acção inicial).

Uma política $\pi(a|x) : X \times A \to [0, 1]$ define a probabilidade de se executar cada acção a para cada estado x. À política que maximiza a reward total chamamos política óptima π^*. Esta pode ser extraída da função Q da seguinte forma:

$$\pi^*(x) = argmax_{a \in A} Q^*(x, a) \tag{7.2}$$

Assim, tendo em conta que num problema de aprendizagem por reforço queremos aprender a política óptima para um dado ambiente, apenas precisamos de aprender a função Q^*. Utilizando Q-learning podemos apróximar a função Q^* a partir de trajectórias de exemplo no ambiente.

Se considerarmos um passo de uma trajectória como o conjunto (x, a, y, r), onde:

- x: estado antes de fazer a acção
- a: acção utilizada
- y: estado após fazer a acção
- r: *reward* obtida por fazer a acção a no estado x

Definimos a seguinte regra de update, assumindo *alpha* como *learning rate*:

$$Q(x, a) \leftarrow Q(x, a) + \alpha(R(x, a) + \gamma \max_b Q(y, b) - Q(x, a)) \tag{7.3}$$

Exercício 1

Considere o seguinte ambiente:

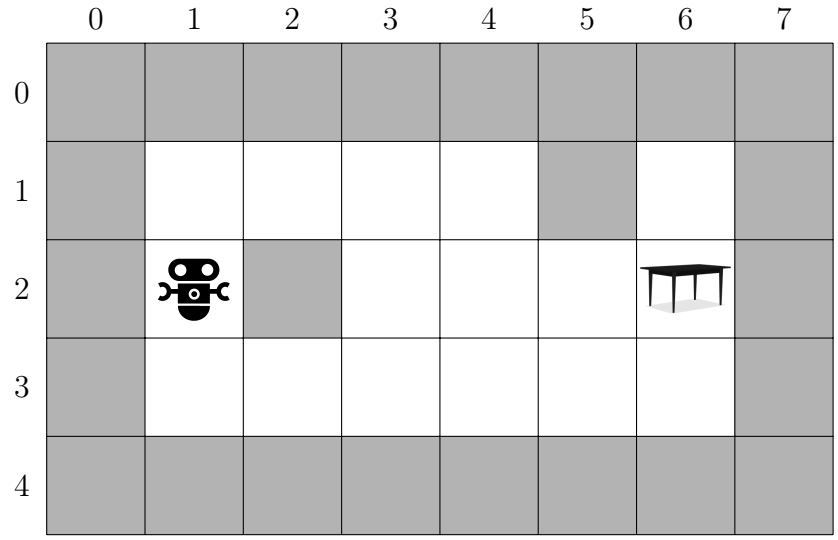


Figura 7.1: Um ambiente, um agente e uma mesa.

Assuma que as acções possíveis são: cima (C), baixo (B), esquerda (E), direita (D). Sempre que o agente faz uma acção no estado da mesa recebe *reward* 0. Em todos os outros casos recebe -1. Se bater numa parede, fica no mesmo estado.

(a) Descreva em conjuntos (x, a, y, r) os passos de uma trajectória óptima até à mesa.

(b) Assumindo que começa com todos os Q a zero, que $\alpha = 0.1$ e que $\gamma = 0.9$, faça os updates do Q-learning para os vários passos da trajectória anterior.

(c) Se fizer uma nova ronda de updates com a mesma trajectória, como ficariam os valores de Q?

Resolução

(a) Escolhendo a trajectória: B,D,D,D,D,D,C,D

1) $([1,2], B, [1,3], -1)$
2) $([1,3], D, [2,3], -1)$
3) $([2,3], D, [3,3], -1)$
4) $([3,3], D, [4,3], -1)$
5) $([4,3], D, [5,3], -1)$
6) $([5,3], D, [6,3], -1)$
7) $([6,3], C, [6,2], -1)$
8) $([6,2], D, [6,2], 0)$

(b) Assumindo que a estimativa inicial de Q tem tudo a zero, os updates de Q-learning ficam:

1) $Q([1,2], B) = 0 + \alpha(-1 + \gamma * 0 - 0) = -0.1$
2) $Q([1,3], D) = 0 + \alpha(-1 + \gamma * 0 - 0) = -0.1$
3) $Q([2,3], D) = 0 + \alpha(-1 + \gamma * 0 - 0) = -0.1$
4) $Q([3,3], D) = 0 + \alpha(-1 + \gamma * 0 - 0) = -0.1$
5) $Q([4,3], D) = 0 + \alpha(-1 + \gamma * 0 - 0) = -0.1$

6) $Q([5,3],D) = 0 + \alpha(-1 + \gamma * 0 - 0) = -0.1$

7) $Q([6,3],C) = 0 + \alpha(-1 + \gamma * 0 - 0) = -0.1$

8) $Q([6,2],D) = 0 + \alpha(0 + \gamma * 0 - 0) = 0$

(c) Repetindo a lógica anterior mas utilizando os Qs actualizados, obtemos:

1) $Q([1,2],B) = -0.1 + \alpha(-1 + \gamma * 0 - (-0.1)) = -0.19$

2) $Q([1,3],D) = -0.1 + \alpha(-1 + \gamma * 0 - (-0.1)) = -0.19$

3) $Q([2,3],D) = -0.1 + \alpha(-1 + \gamma * 0 - (-0.1)) = -0.19$

4) $Q([3,3],D) = -0.1 + \alpha(-1 + \gamma * 0 - (-0.1)) = -0.19$

5) $Q([4,3],D) = -0.1 + \alpha(-1 + \gamma * 0 - (-0.1)) = -0.19$

6) $Q([5,3],D) = -0.1 + \alpha(-1 + \gamma * 0 - (-0.1)) = -0.19$

7) $Q([6,3],C) = -0.1 + \alpha(-1 + \gamma * 0 - (-0.1)) = -0.19$

8) $Q([6,2],D) = 0 + \alpha(0 + \gamma * 0 - 0) = 0$

Exercício 2

Considere o seguinte ambiente:

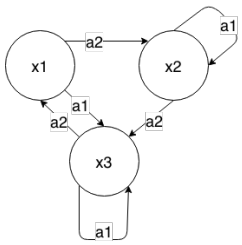


Figura 7.2: Um ambiente com três estados e duas acções.

com espaço de estados $X = \{x_1, x_2, x_3\}$, espaço de acções $A = \{a_1, a_2\}$ e função de *reward*:

$$\begin{bmatrix} R(x_1,a_1) & R(x_1,a_2) \\ R(x_2,a_1) & R(x_2,a_2) \\ R(x_3,a_1) & R(x_3,a_2) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 1 & 0 \end{bmatrix}$$

(a) Calcule o valor de V para o conjunto de acções $\{a_1, a_2, a_2, a_1, a_2, a_1\}$, assumindo que começa em cada um dos estados possíveis (assuma γ arbitrário).

(b) Descreva em conjuntos (x, a, y, r) os passos de uma trajectória que, começando em x_0, segue a política $\{a_1, a_2, a_2, a_1, a_2, a_1\}$.

(c) Assumindo que começa com todos os Q a zero, que $\alpha = 0.1$ e que $\gamma = 0.9$, faça os updates do Q-learning para os vários passos da trajectória da alínea anterior.

(d) Tendo em conta a fase de aprendizagem descrita, escreva uma política óptima.

Resolução

(a) Considerando a trajectória: $\{a_1, a_2, a_2, a_1, a_2, a_1\}$

$$V(x_1) = \sum_{t=0}^{5} \gamma^t r_t = 1 * \gamma^0 + 0 * \gamma^1 + 0 * \gamma^2 + -1 * \gamma^3 + 1 * \gamma^4 + 1 * \gamma^5 = 1 - \gamma^3 + \gamma^4 + \gamma^5$$

$$V(x_2) = \sum_{t=0}^{5} \gamma^t r_t = -1 * \gamma^0 + 1 * \gamma^1 + 0 * \gamma^2 + 1 * \gamma^3 + 0 * \gamma^4 + 1 * \gamma^5 = -1 + \gamma + \gamma^3 + \gamma^5$$

$$V(x_3) = \sum_{t=0}^{5} \gamma^t r_t = 1 * \gamma^0 + 0 * \gamma^1 + 0 * \gamma^2 - 1 * \gamma^3 + 1 * \gamma^4 + 1 * \gamma^5 = 1 - \gamma^3 + \gamma^4 + \gamma^5$$

(b) Considerando a trajectória: $\{a_1, a_2, a_2, a_1, a_2, a_1\}$

1) $(x_1, a_1, x_3, 1)$
2) $(x_3, a_2, x_1, 0)$
3) $(x_1, a_2, x_2, 0)$
4) $(x_2, a_1, x_2, -1)$
5) $(x_2, a_2, x_3, 1)$
6) $(x_3, a_1, x_3, 1)$

(c) Assumindo que a estimativa inicial de Q tem tudo a zero, os updates de Q-learning ficam:

1) $Q(x_1, a_1) = 0 + \alpha(1 + \gamma * 0 - 0) = 0.1$
2) $Q(x_3, a_2) = 0 + \alpha(0 + \gamma * 0.1 - 0) = 0.009$
3) $Q(x_1, a_2) = 0 + \alpha(0 + \gamma * 0 - 0) = 0$
4) $Q(x_2, a_1) = 0 + \alpha(-1 + \gamma * 0 - 0) = -0.1$
5) $Q(x_2, a_2) = 0 + \alpha(1 + \gamma * 0.009 - 0) = 0.10081$
6) $Q(x_3, a_1) = 0 + \alpha(1 + \gamma * 0.009 - 0) = 0.10081$

(d) Após aprendizagem ficamos a estimativa: Daqui podemos extrair a política óptima:

$$\pi^* = argmax_{a \in A} Q^* = argmax_{a \in A} \begin{bmatrix} 0.1 & 0 \\ -0.1 & 0.10081 \\ 0.10081 & 0.009 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Exercício 3

Considere o seguinte ambiente com nove estados e quatro acções: Cima (C), Baixo (B), Esquerda (E), Direita (D):

1	2	3
4	5	6
7	8	9

Figura 7.3: Um ambiente com 9 estados e 4 acções.

A função de *reward* é definida como -1 para todas as acções em todos os estados, excepto

para as acções B e D no estado nove onde é 0. Considere α e γ arbitrários.

(a) Descreva em conjuntos (x, a, y, r) os passos de uma trajectória que começa no estado 1 e segue as acções {B,B,D,D,D}.

(b) Assumindo que começa com todos os Q a zero, faça os updates do Q-learning para os vários passos da trajectória anterior.

(c) Calcule V para a trajectória anterior.

(d) Com base no seu entendimento do ambiente, escreva a política óptima (se para um estado houver mais do que uma acção liste ambas)?

Resolução

(a) Considerando a trajectória que começa em 1 e segue as acções (B,B,D,D,D), temos a trajectória:

1) (1,B,4,-1)
2) (4,B,7,-1)
3) (7,D,8,-1)
4) (8,D,9,-1)
5) (9,D,9,0)

(b) Assumindo que a estimativa inicial de Q tem tudo a zero, os updates de Q-learning ficam:

1) $Q(1, B) = 0 + \alpha(-1 + \gamma * 0 - 0) = -\alpha$
2) $Q(4, B) = 0 + \alpha(-1 + \gamma * 0 - 0) = -\alpha$
3) $Q(7, D) = 0 + \alpha(-1 + \gamma * 0 - 0) = -\alpha$
4) $Q(8, D) = 0 + \alpha(-1 + \gamma * 0 - 0) = -\alpha$
5) $Q(9, D) = 0 + \alpha(0 + \gamma * 0 - 0) = 0$

(c) Considerando a trajectória descrita em (a):

$$V(1) = \sum_{t=0}^{4} \gamma^t r_t = -1 * \gamma^0 + -1 * \gamma^1 + -1 * \gamma^2 + -1 * \gamma^3 + 0 * \gamma^4 = -1 - \gamma - \gamma^2 - \gamma^3$$

(d) O objectivo é chegar ao estado 9 e bater contra a parede para sempre (*reward* é zero), só assim maximizamos o valor. Por isso, temos que:

$$\pi^* = \begin{bmatrix} B/D \\ B/D \\ B \\ B/D \\ B/D \\ B \\ D \\ D \\ B/D \end{bmatrix}$$

↓→	↓→	↓
↓→	↓→	↓
→	→	↓→