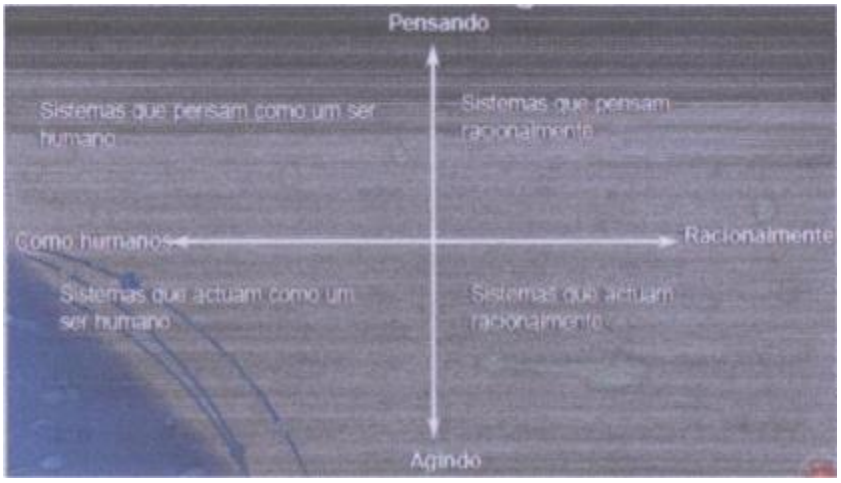
Inteligência Artificial

Slides 01 – Introdução

O que é a inteligência artificial?

Não existe uma definição única, mas podemos classificar inteligência artificial como sistemas que pensam e agem racionalmente, tentando imitar o comportamento e o pensamento de um ser humano, por outras palavras :



"Sistemas que actuam como um ser humano" – São os testes de turing, a arte de criar máquinas para fazerem coisas nas quais os seres humanos hoje em dia são mais eficientes

"Sistemas que pensam como um ser humano" – É o esforço que é feito para as máquinas pensarem, como mentes humanas, perante as tomadas de decisões.

"Sistemas que pensam racionalmente" – O estudo de modelos computacionais que fazem ser possível uma máqina perceber,raciocinar e agir.

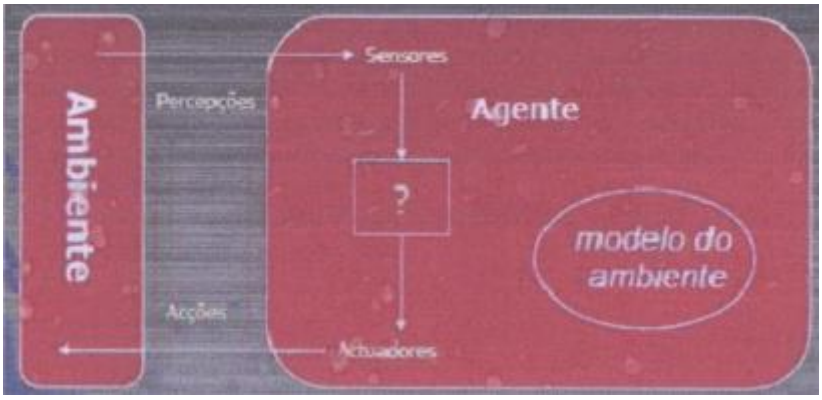
"Sistemas que actuam racionalmente" – O ramo da ciência da computação preocupado com a automação (processos automáticos) do comportamento inteligente.

Os fundamentos da inteligência artificial:

- Matemática
- Psicologia
- Computação
- Linguística
- Filosofia

Slides 02 – Agente

Um agente é qualquer entidade que perceba o seu ambiente através de sensores e que age através de atuadores.



Por exemplo :

- Agente humano
  - Sensores: olhos,ouvidos,pele
  - Atuadores : mãos,pés..
- Agente robô
  - Sensores:Câmara,infravermelhos;
  - Atuadores: Rodas,luzes…
- Agente Software
  - Sensores: Input
  - Atuadores: Output

Propriedades de um Agente:

- *Autonomia*: Operam sem a intervenção de humanos e possuem controlo sobre as suas ações e estado interno.
- *Pró actividade*: Não se limitam a agir em resposta ao seu ambiente, tomam iniciativa e exibem comportamento direccionado por objectivos.
- *Reactividade*: Têm a percepção do seu ambiente e respondem rapidamente às alterações que nele ocorrem
- *Habilidade social* : Conseguem interagir com outros agentes e humanos.
- *Mobilidade*: Têm capacidade de se movimentar de um local para outro.
- *Conhecimento*: Possuem informação dinâmica e capacidade de raciocínio sobre essa informação.
- *Intenções e obrigações*: Intenções são objectivos de longo prazo do agente. Obrigações são o que o agente assumiu fazer previamente.
- *Racionalidade*: Em cada instante, face ao conhecimento adquirido e capacidades, o agente tenta tomar a melhor decisão para completar seus objectivos.
- *Inteligência*: O estado de um agente é formalizado por conhecimento e por interação com outros agentes utilizando linguagem simbólica
- *Continuidade temporal*: o agente é um processo que é executado continuamente ao longo do tempo.

- *Carácter*: O agente possui uma personalidade credível e evnetualmente um estado emocional
- *Aprendizagem*: Adquire conhecimento e altera o seu comportamento baseado na experiência.

Estrutura de um agente

Agente = programa + arquitectura, em que o programa é uma função que transforma as percepções (inputs) em acções (outputs) e a arquitectura é o suporte dado para a execução dos programas do agente.

Definir Agentes

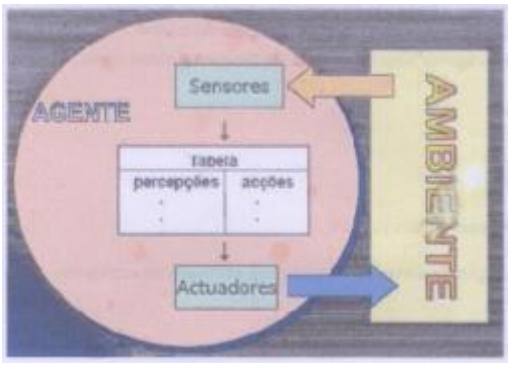
Os agentes são definidos em grande parte pelo projecto do agente que está associado ao ambiente em que este está inserido.

Propriedades do ambiente:

- *Acessível vs Inacessível*: é acessível quando os sensores percebem o estado completo do ambiente;
- *Estático vs dinâmico*: Estático é quando o ambiente não muda enquanto o agente decide a acção a realizar
- *Determinista vs não determinista*: É determinista quando o próximo estado do ambiente, pode ser determinado pelo atual
- *Discreto vs contínuo*: é continuo quando as perceções/ações mudam em um espectro contínuo de valores
- *Episódico vs não episódico:* É episódico quando a experiência do agente é dividida em episódios.

Arquitetura dos Agentes

Agente em tabela



Dada uma percepção, o agente reage procurando a resposta (ação) a essa percepção.

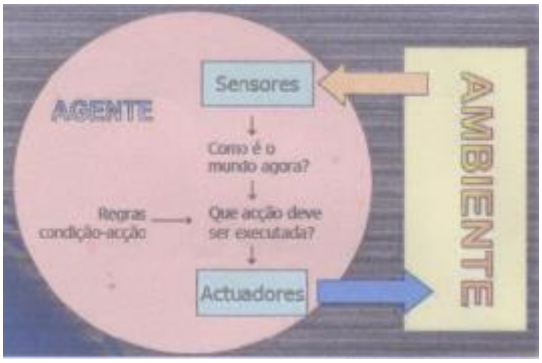
Os seus ambientes são: acessível,determinista,episódico,estático,discreto e minúsculo.
Exemplo: projecto IA

Desvantagens:

- Tabela pode ser muito grande;

- Agente não tem autonomia, é tudo pré-definido internamente.

Agente reactivo



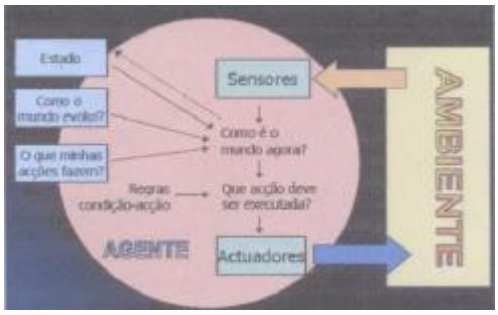
É um agente que usa a regra de condição->ação

Ambientes -> dinâmico,acessível,episódico e pequeno.

Problemas:

- A decisão do agente só depende da percepção atual
- Muitas soluções não são alcançadas porque o agente pode não saber como o mundo era antes
- Tem pouca autonomia
- Só funciona se o ambiente for completamente observável.

Agente reactivo com estado interno (autómato)



O estado actual é dado em função do estado anterior (histórico) e do que foi percebido no ambiente.

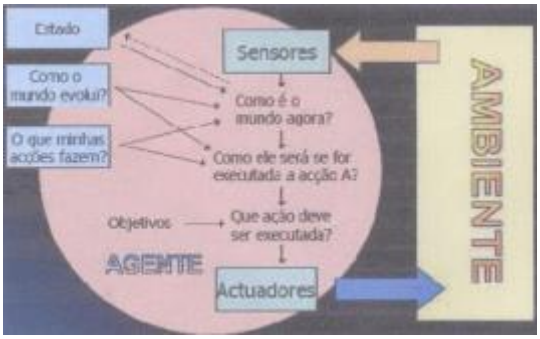
Agente guarda as informações recolhidas do ambiente num estado interno.

Para atualizar a memória do agente é necessário ter conhecimento de como o ambiente evolui independentemente do agente e como as suas ações afectam o mundo.

Problemas:

- Conhecer os estados do ambiente não é suficiente para tomar uma boa decisão. Ex: um taxista chega a um cruzamento com 3 ruas, qual deve seguir? Ter passado lá não ajuda a decidir, porque o que decide é o destino do táxi.

Agente cognitivo (baseado em objectivos)



O agente precisa de algum tipo de informação sobre os seus objectivos Combinando as informações sobre o seu objectivo e resultado das suas ações, torna-se mais fácil escolher a ação mais adequada que alcance seu objectivo.

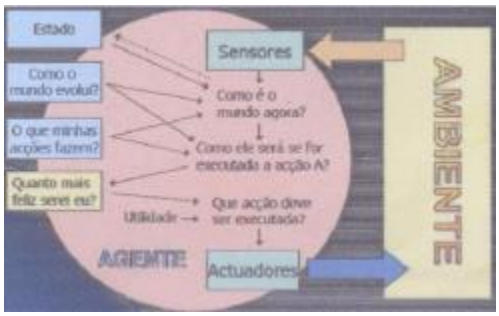
Para encontrar sequências de ações que alcançam os objectivos temos : algortimos de procura e planeamento.

Ambiente: Determinista

Problemas:

- Agentes que funcionam orientado a objectos são mais flexíveis
- Mais flexivel é sinal qe a representação do conhecimento permite modificações
- Objectivo não garante o melhor comportamento para o agente.

Agente optimizador



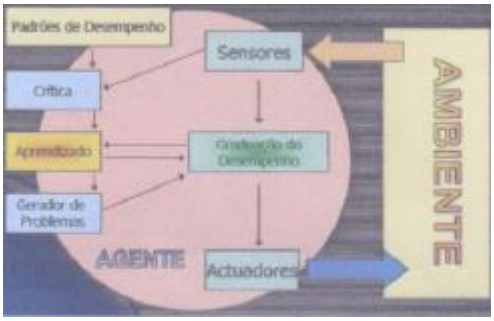
Agente baseado em utilidade, ou seja, um estado do mundo é mais desejável que outro, então esse estado terá mais utilidade para o agente.

A utilidade relaciona um estado para um número real (corresponde ao grau de satisfação). Esta é importante para determinar o peso de de cada objectivo e definir prioridades.

Ambiente: sem restrição

Problema: Não tem adaptalidade.

Agente que aprende



È dividido em quatro componentes conceituais:

- Elementos de aprendizagem: responsável pela execução dos aperfeiçoamentos
- Crítico – Determina de que maneira o elemento de desempenho deve ser modificado
- Elemento de desempenho – responsável pela selecção de acções externas
- Gerador de problemas – responsável por sugerir acções que levarão a experiências novas e informativas

Slides 03 – Procura Cega

Parte I

Resolução de problemas (Acções)

Agente: deve escolher uma sequência de ações que conduzam-lhe ao objectivo

Procura é quando o agente seleciona uma sequência de ações, que conduzem-no à meta desejada.

Ideia de custo é a determinação do agente escolher entre as várias metas possíveis.

Agente solucionador de problemas

É quando o agente exibe um comportamento orientado a atingir metas particulares e deve:

- Ter uma *representação* adequada ao seu ambiente
- Conhecer as *ações* que pode efectuar
- Poder *raciocinar* sobre efeito das suas ações tomadas sobre o ambiente

O agente reactivo escolhe as ações baseado nas percepções atuais e o agente solucionador de problemas procura uma sequência de ações que leve a objectivos desejáveis.

Representação de problemas/modelação

É a relação existente entre as diferentes formas de formular um problema e a eficiência de encontrar uma solução para o mesmo e tem uma grande influência no esforço que é requerido para resolvê-lo.

Modelar é um conjunto de convenções para representar a informação. Por outras palavras são os modelos e servem para visualizar um problema e controlar o processo de resolução de um problema.

Características de uma boa representação/modelo

- *Clareza:* deve ser evidente a relação entre o modelo e o problema real.
- *Exactidão:* o modelo deve ser fiel á realidade nos aspectos relevantes para a resolução do problema
- *Completude:* o modelo deve representar todos os aspectos relevantes para a resolução do problema
- *Eficiência:* a representação/modelo deve poder ser utilizado eficientemente
- *Conciso:* as caracteristicas irrelevantes devem ser omitidas e os detalhes suprimidos
- *Utilidade*: devemos avaliar se o modelo sugere um bom método para resolver o problema

Exemplo de boa representação:

O primeiro mapa do metro de Londres seguia fielmente a geografia das linhas (curvas, e distâncias entre estâncias).

Entretanto o propósito do mapa era mostrar aos passageiros as estações e conexão das mesmas, por isso a fidelidade à realidade dificultava.

Por isso substituiram o mapa por uma representação mais clara e concisa.

Como podemos ver, para modelar é preciso perceber bem o contexto do problema para criarmos uma solução adequada ao mesmo.

Parte II

Procura em espaço de estados

Devemos procurar o estado final depois do problema estar bem formulado, ou seja, deveos utilizar um método de procura para saber a ordem correcta que nos leva do estado inicial ao final. Depois da procura na árvore estar completa, é só executar a solução (ordem correcta devolvida pelo método de procura.

Funcionamento do algoritmo:

1. Selecionar o primeiro nó da fronteira da árvore
2. Testar se o nó é um estado final (solução)
3. Gerar novo conjunto de estados, aplicando os operadores ao estado selecion.
4. Inserir os nós gerados na fronteira, dependendo da estratégia de procura usada e volta ao passo 1.

*Fronteira – é o conjunto de nós que podemos seguir caminho

Cada nó pode guardar a seguinte *informação*:

- Estado correspondente
- Nó pai
- Operador aplicado para gerar o nó
- Profundidade do nó
- Custo do nó (desde a raiz)

Métodos de procura

- *Procura exaustiva – cega*
    - Não sabe qual é o próximo melhor nó -> menor custo de caminho a
    - Direção da procura
        - Do estado inicial para o objectvio
        - Do objectivo para o estado inicial
        - Procura bidireccional
- *Procura heurística – informada*
    - Sabe qual é o próximo melhor nó com base em funções heurísticas (conhecimento)
    - Direção da procura
        - Do estado inicial para o objectvio
        - Do objectivo para o estado inicial
        - Procura bidireccional

Critérios de avaliação das estratégias de procura:

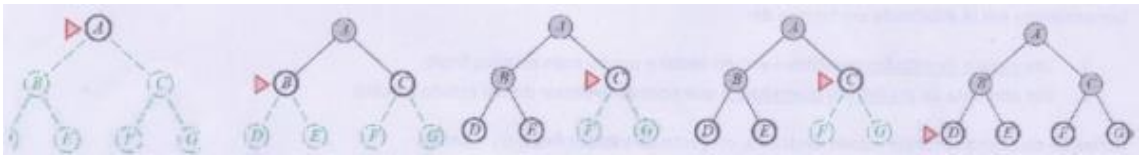
- Completude – Encontra sempre solução a estratégia?
- Custo do tempo – Quanto tempo gasta para chegar à solução?
- Custo de memória – Quantidade de memória necessária para realizar a procura?
- Otimização – Encontra a melhor solução quando existe diversas?

Procura cega

Encontra soluções para problemas pela geração sistemática de novos estados que são comparados com o objectivo. São ineficientes na maioria dos casos porque realizam a procura (expanção de novos estados) sem conhecimento e de forma aleatória. Por isso chama-se procura cega

Notação : b – factor de ramificação; d= profundidade da solução; m = profundidade máxima da arvore; t = limite de profundidade~

1.Em largura primeiro (breath-first)



O nó de menor profundidade é escolhido para gerar sucessores. O nó raiz é expandido e todos os nós gerados são explorados. Só depois de explorar todos os nós de uma profundidade, é que é criado mais um grau de pofundidade,começando sempre na esquerda.

A solução encontrada primeiro será a de menor profundidade.

Caracteristicas : Completo e ótimo.

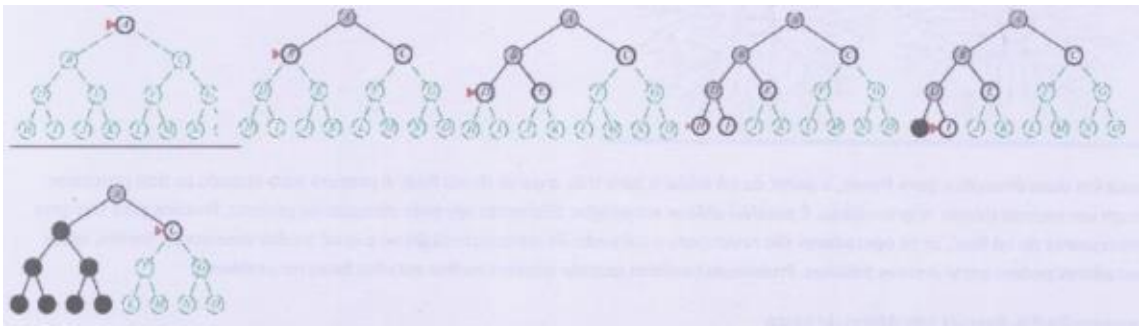
2.Custo uniforme (Uniform-cost)

Tem o mesmo conceito que a procura em lagura, que expande todos os nós de cada profundidade, Neste caso, os nós escolhidos para serem expandidos da fronteira são aquelas que possui menos custo (g(n)), e não é necessário visitar todos os estados de cada profundidade d antes de visitar os estados da profundidade d+1.

Isto garante sempre que a primeira solução será sempre a mais barata.

Características : Completo e ótimo, contudo utiliza demasiada memória

3.Profundidade Primeiro



Expande sempre o nó mais profundo da árvore. 1º o nó raiz, 2º o primeiro nó da profundidade 2 e depois é sempre o que tiver mais profundo.

Quando um nó final não é solução, o algoritmo volta ao nó mais profundo que ainda não tenha sido expandido, e escolhe-o para gerar sucessores.

Caracteristicas :

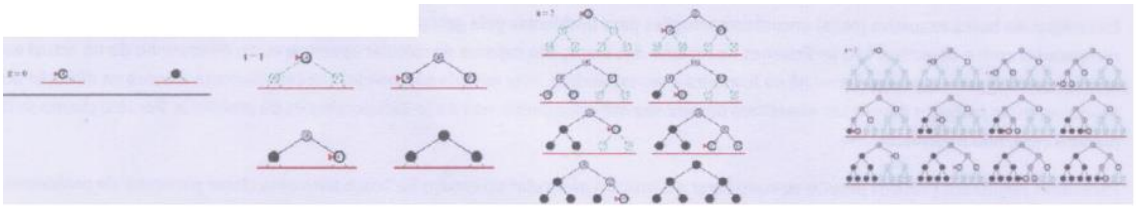
- Nem é completo, nem é ótimo.
- Torna-se mais rápido do que procura em largura quando tem mais que uma solução
- Deve ser evitado em arvores demasiado compridas
- Vantagem: Requer pouca memória
- Desvantagem: Se a profundidade da árvore for infinita, o algoritmo desce indefinidamente.

4.Profundidade Limitada

Um dos grandes problemas da procura em profundidade é caminhos infinitos. Este algoritmo de profundidade limitada, vem resolver esse problema fixando o nível máximo da procura.

Neste caso, cada vez que é para gerar um sucessor aplicamos um operador antes. Depois de alcançar a profundidade limite, a procura pára o processo de gerar sucessores. Este limite serve para evitar as partes do grafo em que se supõe que não encontramos um nó objectivo o suficente perto do nó inicial.

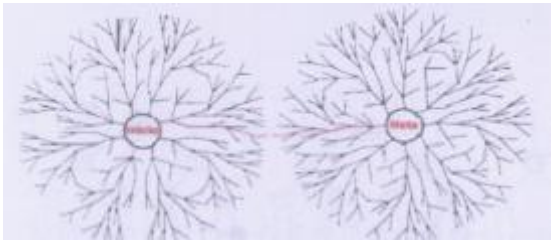
5.Aprofundamento progressivo (progressive depth)



Para evitar os problemas de caminhos infinitos, este algoritmo é uma espécie de procura em profundidade com limite variável. Isto é, o limite máximo inicialmente pode ser 3, e realiza a procura usando o método de profundidade até o nivel 3 mas percorrendo todos os estados tal como no método em largura e caso não encontre a solução, define um nivel novo e volta a fazer o processo.

É uma solução que mistura a procura em profundidade com procura em largura, recolhendo os aspectos positivos de ambas.

6.Bidereccional



É um método que procura em duas direções, para frente do estado inicial e para trás do estado final.A procura pára quando geram um mesmo estado intermediário, sendo possível utilizar diferentes estratégias em cada direção (mas corremos o risco de entrar em loops infinitos).

Comparação das diversas estatégias

Critério	Largura	Custo Uniforme	Profun-didade	Aprofun-damento Iterativo
Tempo	b^d	b^d	b^m	b^d
Espaço	b^d	b^d	bm	bd
Otima?	Sim	Sim*	Não	Sim
Completa?	Sim	Sim	Não	Sim

Slides 04 – Procura Informada

Parte I

Procura heurísitca ou procura informada- é quando utilizamos características próprias do problema em particular para ajudar no processo de busca.

Exemplo: Para encontrar um barco perdido, não podemos procurar no oceano inteiro, então utilizamos as correntes marítimas,ventos para ter uma estimativa da sua localização.

Por outras palavras, é uma procura que introduz *métricas* que permitem ao agente de busca escolher o próximo nó da fronteira a ser expandido.

Heurística – é uma técnica que permite melhorar a eficiência da procura, estimando o custo ou longetitude (na árvore) de um estado até o objectivo.

Notação dado um nó n

- g(n) – custo desdo o nó inicial até n
- h(n) – função heurisitica aplicada ao nó n
- h*h(n) – custo real de um caminho ótimo desde n até uma solução
- f(n) = g(n) + h(n) custo estimado de uma solução que passa pelo nó n

<u>Best-first search</u>

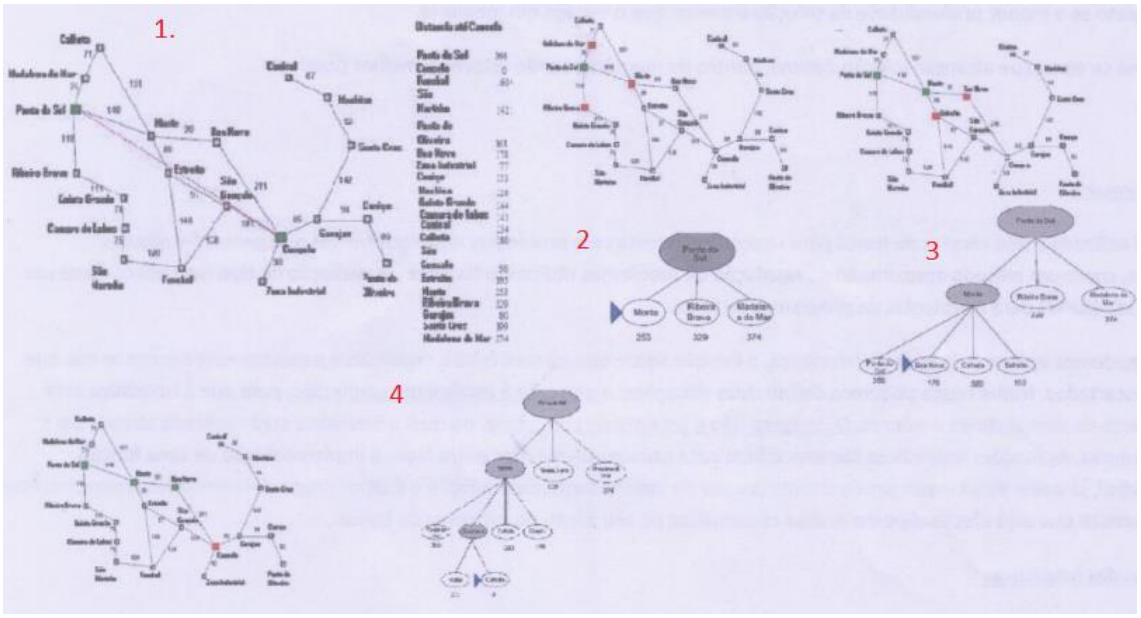
É a buscar genérica onde o nó de menor custo,aparentemente, na fronteira é expandido primeiro, e tem duas abordagens básicas : pesquisa gulosa e algoritmo A*

1.Pequisa gulosa

Tem como objectivo expandir sempre o nó cujo estado é previsto como o mais perto do nó final, com base na função heurística.

Função heuristica -> h(n) -> é o custo estimado do caminho mais barato desde o estado n até o estado final, e deve ser escolhida tendo em conta que o seu custo de aplicação não seja fora dos atingíveis.

Exemplo : Queremos pequisar qual e o caminho mais rapido numa rota na madeira, neste caso Ponta de sol -> Cancela.



Inicialmente, todas as distâncias são calculados pela função heurística, que consiste em calcular a distância que um certo lugar está para com a cancela.

Começamos na ponta de sol e queremos ir para a cancela, temos como hipótese ir pelo Monte (distância de 253 km ate cancela), R.Brava (329km) e Madalenas (374km), por isso optamos por ir pelo Monte.

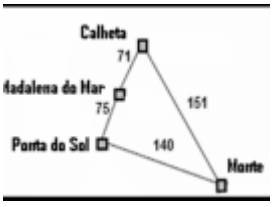
De seguida quando estamos no monte temos as hipoteses de ir pela Ponta de sol (366km), (boa nova 176km), calheta (380m) e estreito (193 km) e, optamos por ir pela boa nova que é mais perto.

Por fim ,quando estamos na boa nova temos a hipótese de ir pelo monte (253 km) e para cancela que é nosso destino final. Chegamos entao ao estado final.

Características :

- Custo de busca mínimos
- Complexidade temporal : O (bn) , b é o fator ramificação e n nivel de solução
- Complexidade espacial : O (bn) , b é o fator ramificação e n nivel de solução
- Não é completa – Pode ter nós repetidos que originam caminhos infinitos.
- Não é ótima – escolhe o caminho mais económico à primeira vista
    - Exemplo:

      Estamos na ponta de sol e queremos ir para o monte, esta busca vai pelas madalenas quando seria mais perto ir diretamente para o monte.



*2.Algoritmo A**

É uma combinação dos métodos de procura Custo Unifora e Busca Gulosa.

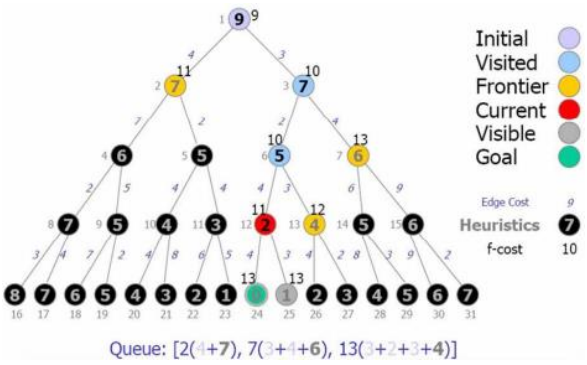
A avaliação de cada nó é uma combinação de f(n) = g(n) + h(n) em que :

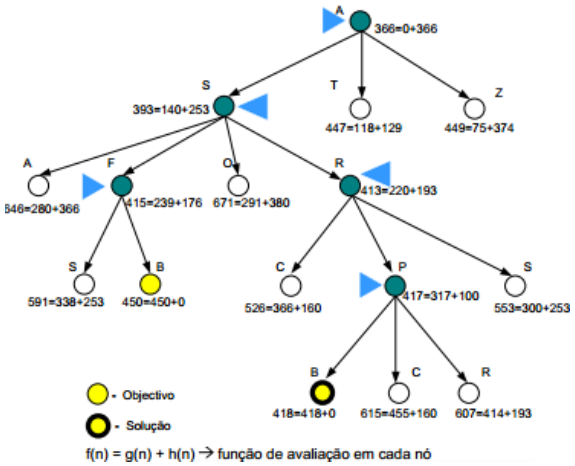
- G(n) – Custo do caminho desde o nó inicial até o n
- H (n) – Estimativa do custo do caminho desde n até nó objectivo
- F (n) – custo estimado de uma solução que passa por n

Funcionamento:

O objectivo é em cada nó que seja expandido tenha associado um valo f(n), e a procura decide sempre por ir pelo menor valor dos nós expandidos

Exemplos:





Função admíssivel - > A pesquisa A* só é completa e ótima, se e só se, escolher uma função heurística admíssivel, ou seja, que nunca sobrestima o custo real do caminho que passa por n (h(n) <= h*(n))

Características :

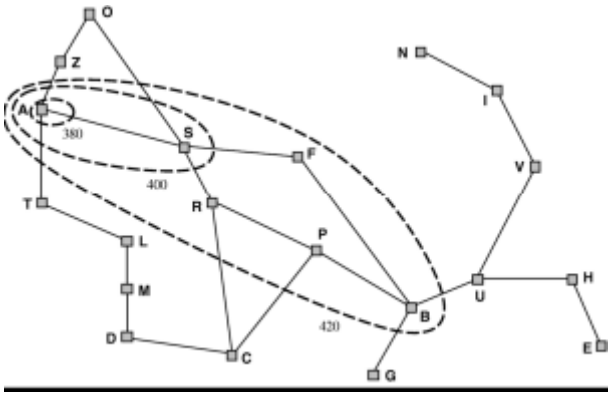
- Complexidade do tempo : exponencial com o comprimento da solução
- Custo de espaço : O (b^d)
    - Guarda todos os nós expandidos na memória e possibilita backtracking
    - Eficiência ótima: entre as várias soluções, encontra sempre a melhor primeiro

*3. Algoritmo IDA **

É igual ao anterior mas tem um limite de profundidade imposto pelo valor de f.

Esse limite é incrementado para o menor valor da iteração anterior.

O corte é feito pelo menor valor de f(N) da iteração corrente, que excede o limite.



- Características:
- Completo e ótimo apenas se o caminho for mais curto, pois cabe na memória dísponivel

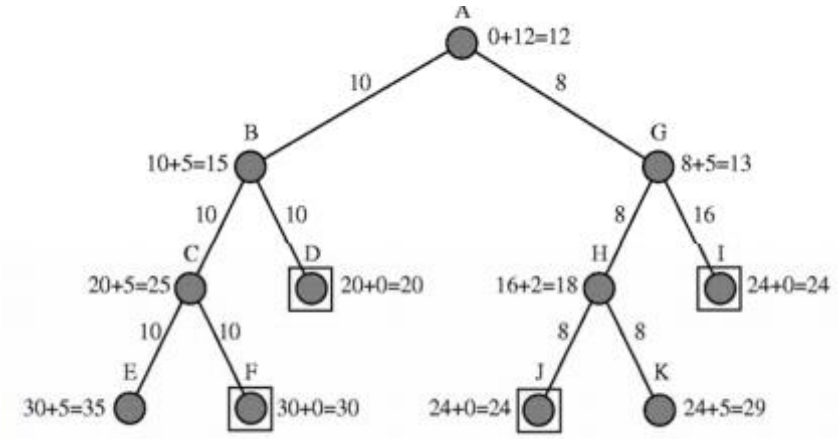
*4. Algoritmo SMA **

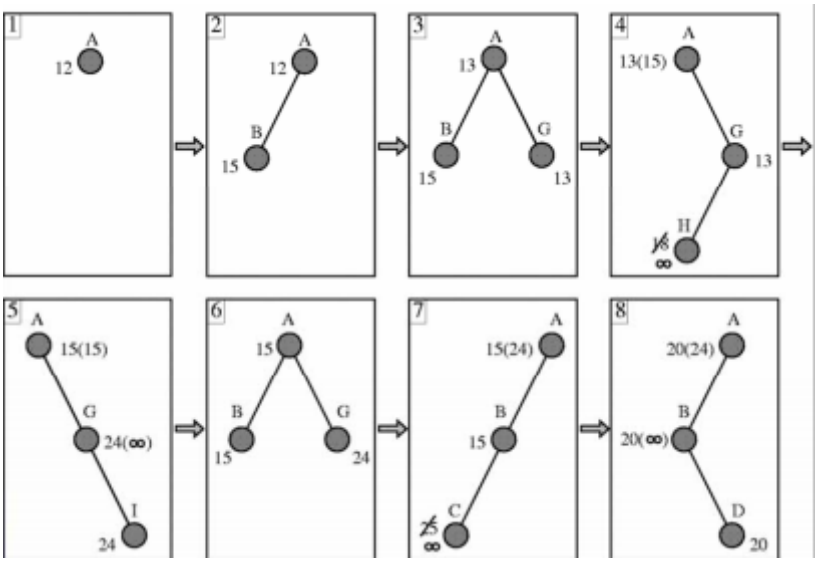
É uma versão simplificada do A*

Faz uso de toda a memória dísponivel

Estatégia:

- Ir expandindo o melhor nó até preencher a memória
- Se ainda não encontrou, descarta o nó com pior valor de f(n) e substitui pelo seguinte melhor nó





Neste caso, começamos no estado inicial A e passamos para o B e para o C.

Quando chegamos neste ponto, a memória fica cheia (máximo de 3 nós), e temos que descartar um e fazemos:

- Comparamos o f(n) ,ou seja, valor do estado e o que tiver menor valor fica.

Portanto, neste caso descartamos o nó B, e ficamos com o G e dexemos mais um nivel de profundidade, ficando de momento com A->G-lugar vazio.

Para substituir o lugar vazio vemos os nós possíveis para ir quando estamos no G ou seja, podemos ir para H e para I.

Neste momento, verificamos que o H não é um estado final, tendo nós filhos, portanto descartamos (fica simbolo infinito) e passamos para o nó seguinte o I, ficando com A->G->I chegando então a um estado final.

É importante referir, que a cada vez que acrescentamos um nó à memória, o valor de A aumenta.

Este algoritmo é:

- Completo se a menor profundidade da solução for menor que o espaço em memória
- Ótimo se consegue alcançar a solução, senão retorna a melhor possível.

Parte II

Processo de Procura

Heurística é utilizada como um método aproximado de resolução de problemas, utilizando funções de avaliação de tipo heurístico.

Numa procura,podemos aplicar a decisão sobre qual nó será feita a expansão e a decisão sobre quais os nós que devem ser descartardos e dependem de:

- Se o universo é conhecido -> heuristica realizada através de atribuição de números
- Se o universo é desconhecido -> heuristica realizada através de aplicação de regras

Função heuristica é específica para cada problema e são difíceis de implementar porque é complicado medir o valor de uma solução e medir os conhecimentos de forma matemática para ser utilizado no processo de busca.

Características das heurísticas:

- Admissibilidade – ou seja, nunca subestimar o custo real da solução
- Consistência – ou seja, cada nó n e cada sucessor de n gerado por qualquer ação x, o custo estimado par atingir o estado final nunca é superior ao custo em cada expansão obter o sucessor de n e o custo desse atingir o estado final.
- Dominância – quando é possível definir mais que uma função heurística, devemos usar aquela que envolve todas as outras. Caso não seja possível, devemos aplicar cada função a um nó diferente.
- Qualidade – é medida através de effective branch -> N = 1+ b* + (b*)2 +…+ (b*)d, e mede quanto um processo de procura está concentrado em atingir seu objectivo.
- Análise de performance – é definida através da precisão da função heurística e a simplicidade do processo.

Algortimos de procura local

Utilizam um único estado, o actual, e procuram melhora-lo sem guardar os caminhos seguidos pelo algoritmo.

Tem como vantagem:

- Utilizar muito pouca memória
- Conseguem encontrar boas soluções em espaço de estados infinitos

Algoritmo trepa colinas

Problema das n rainhas

- Temos que colucar n rainhas num tabuleiro n x n , tal que exista apenas uma rainha por linha,coluna ou diagonal.
- Exemplo n = 4:

No primeiro quadrado, a primeira rainha pode ser atacada pela direita e diagonal em baixo, a segunda pode ser atacada pelas outras 3, a terceira também…

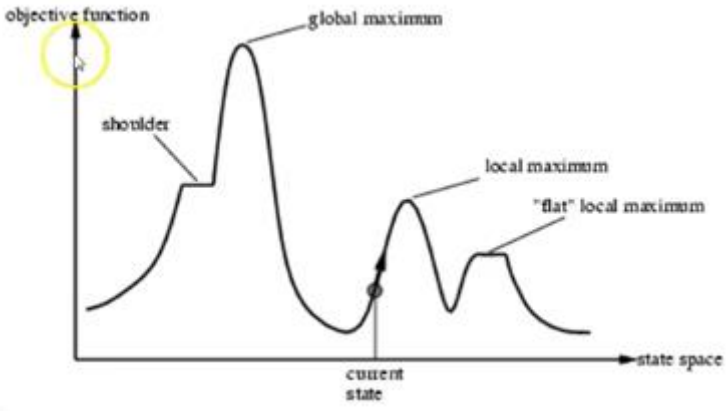
No segundo quadrado ao movermos a rainha um quadrado abaixo, faz com que não possa ser atacada por nenhuma

No terceiro, ao mover a outra para o fundo, ficam todas em segurança.

O algoritmo utilizado foi :

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node
  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
     neighbor ← a highest-valued successor of current
     if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
     current ← neighbor
```

No entanto este algoritmo tem problemas tais como:



- Máximos locais – O algortimo pára porque só se move com uma taxa crescente de variação, ou seja, quando o proximo estado é melhor que o atual. Chegando ao pico, os seguintes são de menor valor e então fica preso.

- Planaltos – Uma região de estados em que o valor da função é igual, fazendo com que o algoritmo pare de procurar outros estados após algumas tentativas.

Problema das 8 rainhas

- ***h*** = number of pairs of queens that are attacking each other, either directly or indirectly
- ***h*** = *17* for the above state
- Here h is the heuristics used, and can define your objective function.

Nesta caso depois do algoritmo ser utilizado, pára quando o maximo local tem h =1, ou seja, quando apenas há um caso em que as rainhas podem-se atacar, sendo o melhor caso possível com 8 rainhas no tabuleiro.

Caracteristicas do algoritmo Trepa Colinas:

- Completo apenas quando todos os nó tratados são estados completos
- Ótimo apenas quando o número de interações possíveis forem suficientes.

Anelamento Simulado

É uma especie de trepa colinasm mas oferece a possibilidade de escapar aos máximos locais, e evitar que o algoritmo fique preso nessas situações. Para tal é aumentado o tempo de procura.

Slides 05 – Jogos

Parte I

Introdução à procura adversária

Procura adversária é aplicar algoritmos de procura mas considerando a existência de inimigos onde ambos querem ganhar o jogo.

Os jogos são ideias para estudar o uso de estratégias adversárias, dado que são aplicaçoes atrativas para métodos de IA :

- Formulação simples do problema (ações bem definidas)
- Ambiente acessível
- Representação simplificada de problemas reais
- Sinónimo de inteligência
- Boa medida de desempenho : vitória
- Contêm os seguintes desafios:
    - Tamanho e limitação do tempo da árvore
    - Incerteza devido a imprevisibilidade do adversário

Características dos jogos:

- Ambientes competitivos, onde os agentes têm objectivos que estão em conflito
- Tipos de jogos:
    - Informativo – pefeitos (xadrez) e imperfeitos (poker)
    - Sorte - Poker e monopolio

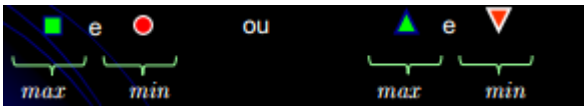
O jogo define-se como problema de procura com as seguintes características:

- Estado inicial -> inclui posicao do tabuleiro e indicação de quem joga
- Conjunto de operadores -> definem os movimentos válidos para cada jogador (ações)
- Teste de terminação -> estabelece quando jogo acaba (estados finais)
- Função utilidade -> atribui valor numerico ao estado final do jogo, que singnifica quem pode ganhar : Max = 1 , empatar = 0 , min = -1

A diferença deste tipo de procura, é que já não basta realizar uma simples procura na árvore até encontrar estado final, temos que delinear uma estratégia de forma a ganhar o adversário, e então chegar ao estado final primeiro.

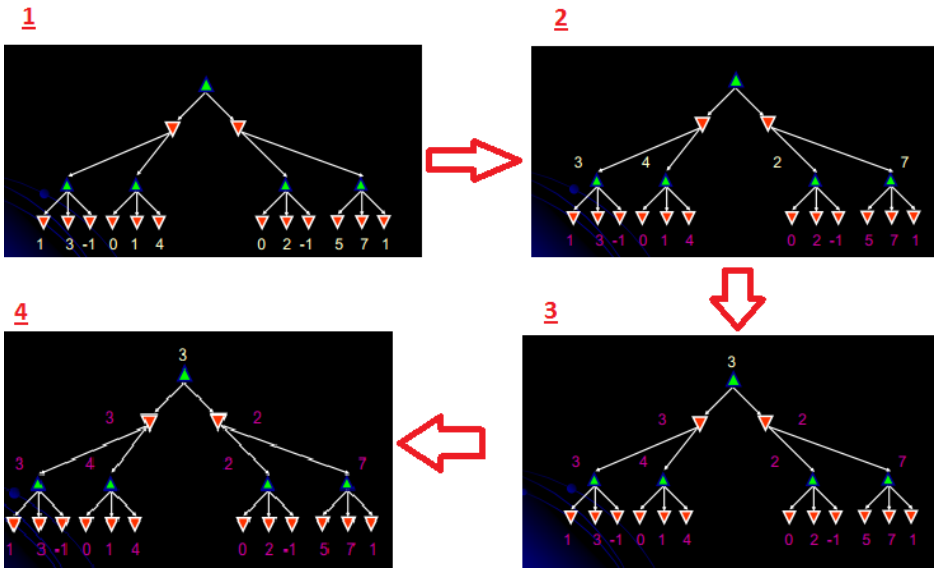
Algoritmo Min-Max

Neste algoritmo é importante referir que :

- Max vai realizar sempre a sua melhor jogada
- Min vai realizar sempre a pior jogada para max
- Devemos :
    - Utilizar a procura cega em profundidade em que :
        - 1. Geramos a arvore completa ate aos estados terminais
        - 2. Aplicamos a função de utilidade (associamos valores 1,0,-1) aos nós
        - 3. Escolher movimentos conforme quem joga
        - 4. Valor que fica no nó raiz, é o valor máximo que o Max pode obter
        - 

Exemplo:

Neste exemplo, o Max tem que escolher o maior número possível, e o min tem que fazer com que o max fique com o número mais baixo possivel (https://www.youtube.com/watch?v=KU9Ch59-4vw).



O primeiro a jogar é o max, e como escolhe sempre os numeros mais altos , escolhe o 3, 4, 2 e 7 para os estados que dizem respeito à profunidade 2

Depois joga o min que deve escolher sempre o pior numero mais baixo, ficando escolhido o 3 e o 2 respectivamente à profundidade 1

Depois por fim, joga o Max (estado inicial) que escolhe o maior numero ficando com o estado inicial = 3.

Algoritmo Alpha-beta

Poda (corte) -> É o processo de eliminar uma ramificação da árvore de pesquisa de forma a que não seja examinada.

- Serve para eliminar nós desnecessários à pesquisa, para acelerar o processo de pesquisa.
- Tem como vantagem tornar a ordenação melhor logo mais eficiente a pesquisa na árvore
- Retorna as mesmas escolhas que o minimax, mas examina menos nós

Alpha -> Representa o valor mais alto encontrado no caminho do max

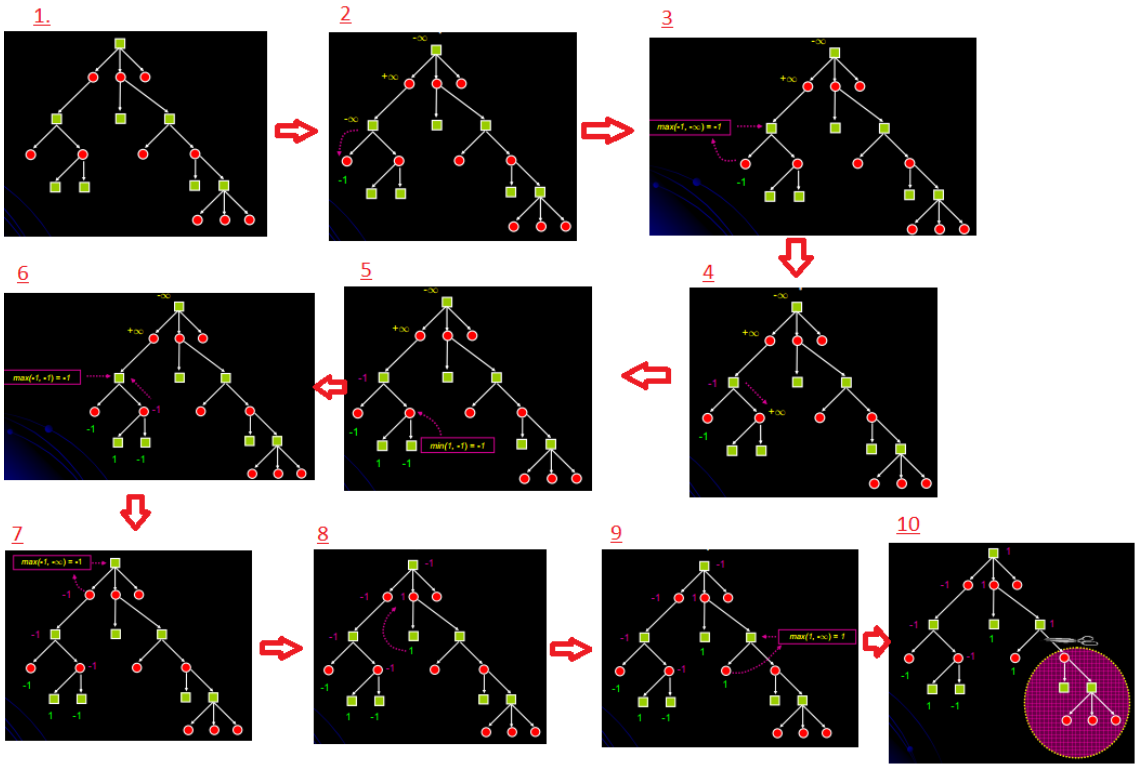
Beta -> Representa o valor mais baixo encontrado no valor do min

Como funciona:

O nó vai ter duas caracteristas : Alpha e beta associadas ao mesmo, em que são atualizadas a cada profundidade. Então o que faz :

- Se estivermos num nó Min -> se o alpha > beta, podemos descartar o resto do nó
- Se estivermos num nó Max -> se o alpha for > beta , podemos descartar o resto do nó

Exemplo:



Funcionamento (https://www.youtube.com/watch?v=xBXHtz4Gbdo):

1. Fig 1 até fig 2 – como ainda não tem valores associados aos nós, associa o infinito e dexe sempre até encontrar um nó com um valor
2. Fig 2 até fig 3 – Como trata-se de um nó Max, verifica qual é o maior valor e coloca estado a - 1
3. Fig 3 até fig 5 – volta a dexer novamente, porque ainda não tem valores associados aos nós
4. Fig 5 até fig 6 – É um nó Min, logo verifica qual é o valor minimo dos estados possiveis e associa ao estado, o valor do nó mais baixo que é -1
5. Fig 6 até 7 – Faz os nós Max e Min abaixo, e depois quando chega ao estado inicial verifica o valor máximo dos nós filhos (neste caso é -1) e associa ao estado inicial. Neste momento temos alpha = - 1.
6. Fig 7 até 8 – percorre o nó do meio e como é o único valor ainda lido, associa estado o valor 1
7. Fig 9 – ele associa ao estado o valor 1, e compara com o valor do alpha (que é -1 ainda). Por isso como o valor 1 é > a -1, ele já nem verifica o resto dos ramos, e guarda esse valor, e volta ao estado inicial.

Eficiente do algoritmo alpha beta :

- Depende da ordem como e feita a prgoressao de pesquisa.
- Comparado com o mini max, examina sempre menos estados

Problemas do algoritmo alpha beta :

- Penalzia movimentos que penalizam algo no inicio para obter frutos mais tarde
- Avaliação de utilidade não é exacta
- Assume que o oponente escolherá sempre o melhor movimento possível

Resumo:

- Em jogos com informação perfeita entre dois jogadores, o minimax pode determinar a melhor jogada
- O algoritmo alpha beta é mais eficiente que o minimax por causa dos cortes

Parte II

Decisões imperfeitas

A maioria dos jogos, o espaço de procura é muito grande e há restrições de tempo.

Ex: o xadrez tem factor de ramificação = 35, supondo que nosso agente procura 1000 estados por segundo e tem tempo limite de 150 seguntos, então dá para procurar 150.000 estados,ou seja, equivale a pensar 4 jogadas a frente e isso não é um bom jogador de xadrez.

Criticas ao minimax: o tempo gasto é impraticavel mas serve como base para outros algoritmos mais realísticos.

Para melhorar, criaram o alpha-beta porém não é suficiente por isso têm que ser alterados, para utilizar funções de avaliação (ser possivel avaliar nós intermedios sem chegar ao fim da arvore).

Jogos não determinísticos

Existem elementos externos que afectam o jogo, tal como um lançamento do dado, lançar uma carta ao acaso,etc tornando-se impossivel de criar uma árvore standarizada.

Jogos de azar com informação perfeita: o factor sorte faz parte do jogo, por isso começamos a ponderar dois elementos fundamentais num jogo: sorte e perícia.

Nós de probabilidades : como temos agora o factor sorte nas árvores, passa a haver nós Max, Min e nós Probabilidadesem que :

- Um nó de probabilidade tem associado nós sucessores, nomeadamente Min e Max.
- Cada nó sucessor tem associado uma probabilidade de ser pesquisado
- Resumo : Muda o funcionamento todo até agora visto

Como tomar decisões correctas?

Não é possível obter o conjunto de todos os movimentos válidos que o adversário tem ao seu dispor, portanto não podemos ter a certeza de como evitar certas jogadas por parte do adversário. Deixamos de calcular valores exactos e passamos a calcular valores esperados baseados em probabilidades.

Como escolher melhor jogada que nos leva à melhor posição dísponivel?

A solução passa por fazer uma generalização do algoritmo minimax para jogos que incluam nós de probabilidades, conhecido por expectminimax.

Expectminimax

- Nós terminais terminai , Min e Max são avaliados como anteriormente
- Nós probabilidades avaliados com base na média de todas as posições possíveis

Custo Extra

Torna irrealista olhar muito em frente na árvore, porque a nossa habilidade de olhar em frente depende de uma quantidade de enventos aleatórios que possam ocorrer.

Como não existe sequências exactas de movimentos, torna-se impossível aplicar os cortes alpha-beta. Para isso é utilizado o *-minimax.

Jogo de azar com informação imperfeita

Tipicamente é possível calcular uma probabilidade para cada mão possivel.

Para isso calculamos o valor do minimax para cada acção em cada mão, e depois escolhemos o estado com o valor esperado maior.

Resumo:

- Jogos são extermamente engraçados e perigosos
- Jogos ilustram pontos interessantes de IA
    - Perfeição é inatingível -> melhor possível é aproximar
    - A incerteza restrigne a atribuição de valores aos estados.

Slides 06 – Conhecimento e inferência

Parte I

Todos os seres humanos têm raciocinio de bom senso. Os componentes que sobressaem desse bom senso é a representação e o raciocínio. Representação para poder aplicar o raciocinio num determinado cenário, e o raciocinio serve para realizar inferências a partir do mesmo.

Abstração baseada em posturas

Quando explicamos o comportamento de um objecto, podemos fazer através de diferentes níveis de abstração.

Quanto mais concreto o nivel, mais correcto esta nossas predições, no ento quanto mais abstracto mais fácil é a capacidade computacional.

No entanto esta abstração está dividida em diferentes tipos de posturas:

- *Postura física* – Trata-se de nos preocuparmos com calculos físicos e matemáticos tais como : trajectórias,velcodiades,energias,etc
- *Postura de desenho* – Trata-se de nos preocuparmos com detalhes tais como função , desenho. Tomamos uma postura de desenho por exemplo : quando predizemos que um pássaro voa ao mexer as asas, baseamos-nos em que as asas foram feitas para voar.
- *Postura intencional* – Trata-se de considerar crenças, desejos e intenção: Por exemplo, quando predizemos que um pássaro voa porque um gato aproxima-se, estamos utilizando a postura intencional.

Representação e raciocínio

Para poder racicionar sobre algo é necessário representá-lo. Raciocinamos para poder predizer o comportamento do cenário. Estes dois processos estão intimimamente ligados e dependem um do outro. Juntos são dos aspectos mais importantes na ciência da computação.

Representação do conhecimento

É o estudo de simbolos formais utilizados para representar uma colecção de proposições que são acreditadas por um agente.

O raciocínio é o encarregado de obter as proposições que são acreditadas a partir das que são representadas.

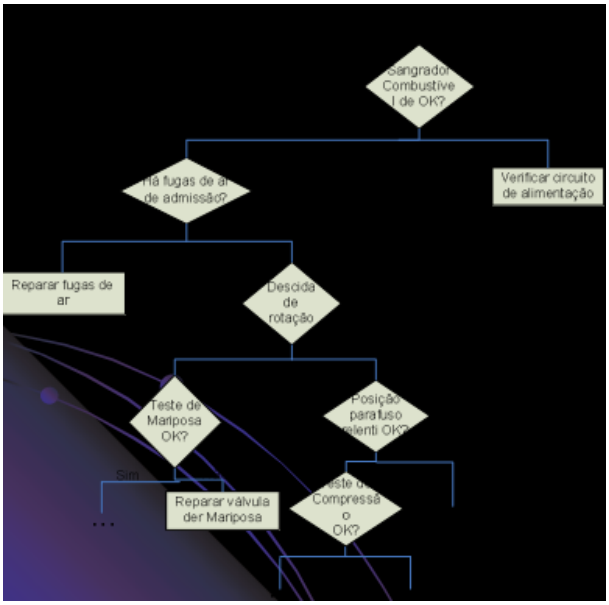
Tipos de conhecimento :

- *Causal x diagnóstico*
  - Casual – prevê resultados de ações e eventos . Ex : Olho para norte e viro à direita, passo a olhar para leste
  - Diagnóstico – Forma hipóteses sobre causas de efeitos observados. Ex: Cheira a queimado, não devo ter desligado o fogão.
- *Terminológico x Dedutivo*
  - Terminológico – Aspectos estruturais e estacionários. Ex: Se joão é filho do pedro e pedro é filho do carlos, então joão é neto do carlos
  - Dedutivo – Aspectos comportamentais e temporais. Ex: sigolismos
- *Certo x incerto*
  - Certo – Epistemologicamente booleano. Ex: Pedro é português
  - Incerto – Epistemologicamente probabilista . Ex: Se jogador um dado, vai sair um 6 com 176 de probabilidade
- *Preciso x vago*
  - Preciso – Epistemologicamente booleano. Ex: João é português
  - Vago – Epistemologicamente possibilista. Ex: João é alto

Conhecimento declarativo – Descreve o que é conhecido acerca de um determinado problema (inclui declarações que são assumidas como verdadeiras/falsas que descrevem um objecto/conceito). Ex: Fumar pode provocar cancro no pulmão

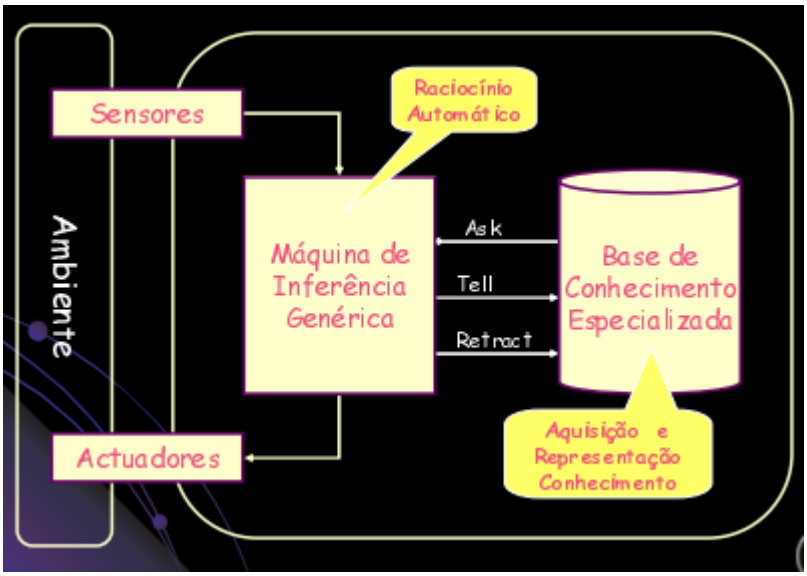
Conhecimento Procedimental – Descreve como um problema é resolvido ou como agir perante uma dada situação (regras, estratégias, etc).

Ex:



Conhecimento especialista e bom senso - > reflecte o conhecimento obtido com toda a experiência que se detém ao lidar/resolver com um determinado tipo de problema.

Agente baseado em conhecimento



Ontologia -> define um vocabulário comum para profissionais que necessitam partilhar informações num domínio.

Parte II

Para representar um raciocinio destinguem-se as seguintes componentes:

- Linguagem forma de representação – utilizada para descrever as fórmulas válidas para expressar conhecimento acerca do dominio
- Semântica- significado das sentenças da linguagem, ou seja, relaciona os elementos da linguagem e o dominio
- Teoria de raciocinio – Conjuntos de regras de inferencia
    - Teoria correcta – se infere respostas correctas de acordo com a semântic
    - Teoria completa – se gera todas as respostas correctas

O que é raciocinio inteligente?

É uma forma de calculo forma, que tem a lógica como ponto inicial em que esses cálculos envolvem formas de dedução . Ou seja, é caracterizado por um comportamento de estimulo/resposta emergente da interacção de uma enorme quantidade de processadores conectados em paralelo.

O que é raciocinio dedutivo?

É uma classe de raciocinio onde a conclusão é alcançada a partir das premissas, de factos previamente conhecidos.

Ex: Todos os homens são mortais. Sócrates é homem. Então sócrates é mortal.

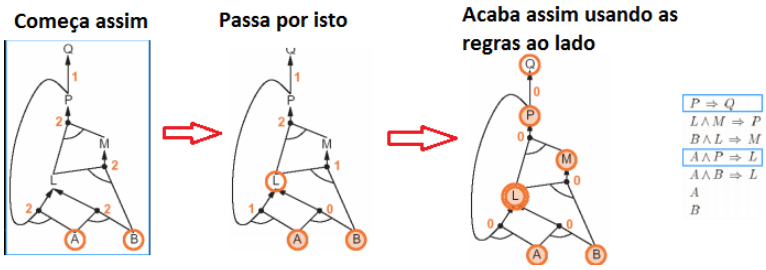
Encadeamento para a frente

Parte de um conjunto de factos e usa as regras da KB para deduzir novos factos.

Estratégia

- Os factos básicos originam o disparo de regras.
- As regras conduzem à obtenção de conclusões intermédias
- Conclusões intermédicas juntamente com os factos básicos conduzem a disparos de novas regras
- O processo continua até que se chegue à conclusão final..

Exemplo

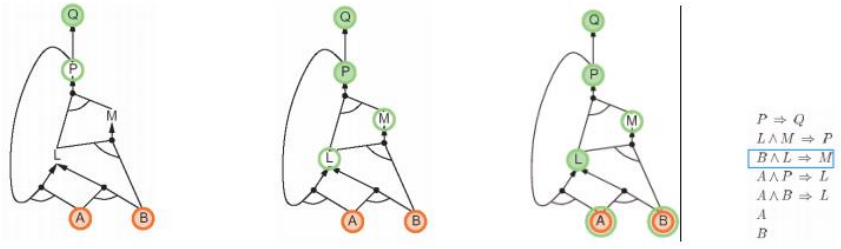


Encadeamento para a trás

Estratégia

- Provar as conclusões finais que aparecem ao lado direito das regras
- As conclusões são provadas, provando as condições que aparece do lado esquerdo da regra
- As condições do lado esquerdo pode mser suportadas por conclusões intermédias de outras regras ou por factos básicos.

Ex:



Encadeamento para a frente vs para trás

Encadeamento para frente	Encadeamento para trás
Planeamento, monitorização e controlo	Diagnósticos
Presente para o futuro	Presente para o passado
Antecedente para consequente	Consequente para antecedente
Raciocínio *bottom-up* baseado nos dados	Raciocínio *top-down* baseado no alvo
Trabalhar para a frente para encontrar soluções que derivam dos factos	Trabalhar para trás para encontrar os factos que suportam a hipótese
Antecedentes determinam a procura	Consequências determinam a procura

Parte II

Lógicas não monótonas

O que queremos dizer com esta imagem é que quando não possuímos informação completa, utilizamos o nosso bom senso e pensamos nas hipóteses mais viáveis. Ou seja, nova informação pode fazer-nos mudar de opinião

Mundo fechado

Esta suposição baseia-se em assumir que toda a informação positiva que é preciso saber, está na nossa kb.

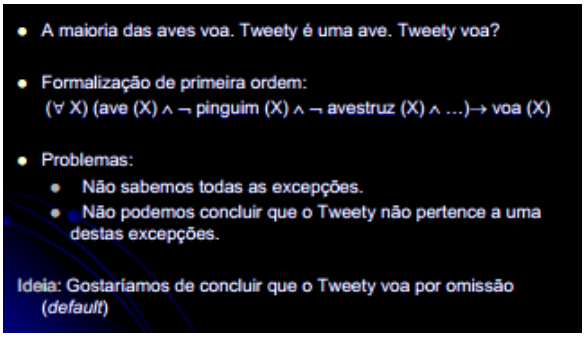
Esta suposição permite obter informação negativa. A informação negativa não esta representada frequentemente de uma forma explícita.

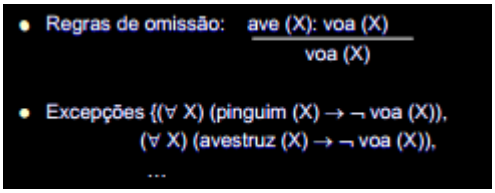
A informação que não é mencionada na kb assume-se como falsa.

Lógica de omissões (default logic)

- Diferencia-se de outras abordagens pela maneira como representa omissões, nomeadamente regras de inferencia com uma verificação adicional de consistência
- Regra de inferencia : (F : G) / H
    - Significa que se F +e conhecida e não há qualquer prova de que G seja falsa, então H pode ser inferida.
- F , G, e H são frases de linguagem

Exemplo:





Consistência

- Aplicação de uma regra de omissão requer que uma condição consistente seja satisfeita
- O que torna esta condição complicada é a consistência depender da aplicação ou não aplicação de todas as regras
- Para resolver esta dificuldade, é utilizado a noção de extensão

Parte IV

Cálculo situacional

Serve para guardar as mudanças na base de conhecimento , por exemplo um agente ir de (1,1) ate (1,2).

Descrição das ações

- As acções são descritas pelos seus efeitos
    - Especificando as propriedades da situação resultante da realização da acção
- Axiomas de efeito descreve como o mundo muda
- Problemas:
    - Axioma do efeito diz o que muda mas não diz o que fica igual (frame problem)
        - Para resolver foi inventado os axiomas de frame, que descrevem como o mundo fica igual

Slide 08 – Sistemas Multi Agentes

Parte I

Porquê sistemas multi-agentes na resolução de problemas?

Porque possuem caracteristcas que permitem resolver os problemas de forma diferente da tradicional, adequando-se a problemas complexos e de natureza descentralizada e porque os problemas reais são demasiado grandes para serem resolvidos apenas por um agente.

Exemplos

- Entidades distinas (controlo áreo)
- Problemas distribuidos no espaço (supervisão rede informática)
- Peritos são muitos (concepção de um avião)

SMA – Sistemas multi agente

São utilizados quando:

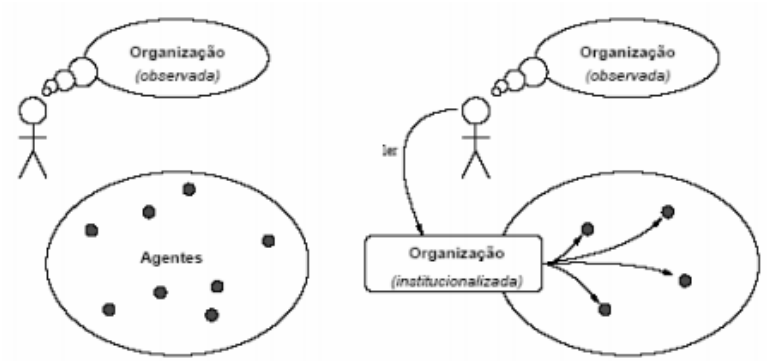
- Necessidade de manter a autonomia de subsecções, sem perdad e estrutura organizacional
- Complexidade nas interacções, partilha de informação e coordenação
- Impossibilidade de descrição da solução do problema à priori, devido a problemas em tempo real no ambiente (falhasd e equipamento)

Caracteristicas dos SMA's :

- Possui os seguintes elementos
  - Um ambiente
  - Conjunto de objectos (O)
  - Conjunto de agentes (A)
  - Conjunto de relações, (R) que liga os objectos
  - Conjunto de operações (OP)
- Objectivos : o agente têm sempre um objectivo a ter em conta
- Pró actividade : são capazes de agir activamente de acordo com o objectivo
- Habilidade socia : é a capacidade de interagir com outros agentes

Prespectivas dos SMA's :

- Prespectiva do agente :
  - Foca elementos que caracterizam o agente envolvido num SMA (Categorias, Capacidade de raciocinio, capacidades de adaptaçao,etc)
- Prespectiva do grupo :
  - Reúne aspectos de grupo, tais como organização, coordenação,interação, comunicação,etc

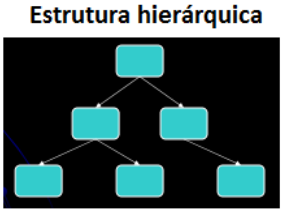


Prespectiva do agente

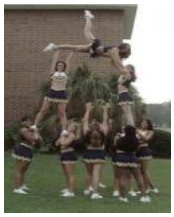
- *Estutura e manutenção de conhecimento*
  - Um agente é constituido por representações que possui sobre mundo real e problema a resolver. Este conhecimento é combinação do conhecimento adquirido anteriormente com o conhecimento adquriido durante a fase de resolução do problema
- *Capacidade raciocinio*
  - Devem explorar várias hipóteses antes de tomar uma decisão e ter em conta o conhecimento e comportamento de outros agentes
- *Capacidade aprendizagem*
  - Necessita de se adaptar ao seu ambiente e comportamento de outros agentes
- *Arquitecturas de agentes*
  - São os diferentes tipos de agentes existentes ara operar em diferentes ambientes.

Prespectiva do grupo

- *Estrutura*
  - Padrão de relações de informação e controle entre agente, bem como distribuição de habilidades e responsabilidades entre eles
- *Organização*
  - Conjunto de compromissos globais, crenças mútuas e comuns aos agentes com o intuito de atingir certo objectivos .
  - Exemplos estruturas:



- *Comunicação*
  - Capacudade de um agente intercambiar informações com outros agentes
- *Interação*
  - Conjunto de comportamentos que resulta do agrupamento de agentes que devem actuar no ambiente para atingir os objectivos comuns
- *Cooperação*
  - Quando vários agentes planeiam e executam suas ações de forma coordenada . *Ex:*



- *Negociação*
  - Fundamental em actividades cooperativas dentro de sociedades humanas, permite que pessoas resolvam conflitos que possam interferir com o comportamento cooperativo

Parte II

Aspectos arquitecturais dos SMA's

Estrutura hierárquica

- Agentes podem comunicar apenas com os agentes por ele supervisionados e com seu supervisor
- Vantagens a comunicação ser mais eficaz e rápida
- Desvantagens é não ser reorganizável quando necessário para atender às necessidades de uma determinada tarefa

Estrutura nivelada

- Cada agente pode também falar com os agentes do mesmo nivel
- Vantagens aumentar quantidade de comunicação existente e haver maior possibilidade de reorganizaçãó

Estrutura Composta

- Quando existe agentes que são componentes de outros agentes

Diversos níveis de abertura

Abertura dinâmica

- Agentes podem sair e entrar do sistema a qualquer momento
- Não é preciso notificar outros agentes
- Vantagem : facilita adaptação do sistema em relação a mudanças do ambiente
- Desvantagem : precisa de serviços adicionais para localizar os agentes

Abertura estática

- Agentes podem sair e entrar do sistema mas com menos frequência
- É preciso notificar outros agentes
- Geralmente utilizado quandos agentes vão interagir com o sistema (ex: jogo fut)

Abertura off-line

- Para entrar ou remover agentes deve ser desativado o sistema e depois de entrar/sair é reativado o sistema

Resumo

O futuro da IA passa pelos SMA's devido às suas diversas caracteristicas. Apesar de ainda estarem em desenvolvimento já fornecem uma boa solução para os diferentes problemas existentes entre os vários agentes que habitam e evoluem em múltiplos ambientes, nomeadamente no da robótica.