

INTELIGÊNCIA ARTIFICIAL

T5 - Classical Search

Agente \rightarrow algo que percepciona o ambiente através de sensores e atua no ambiente através de actuadores.

- O termo percepção refere-se às percepções do agente num dado instante. Uma sequência de percepções do agente é o histórico total de tudo o que o agente percecionou.
- O comportamento do agente é descrito matematicamente pela função do agente que mapeia qualquer sequência de percepções numa ação.

Agente: arquitetura* + programa**
 * dispositivo computacional com sensores e actuadores
 ** materialização da função do agente

FORMULAÇÃO DE PROBLEMAS

* Problem Solving Agent \rightarrow procura encontrar a sequência de ações que leva a um estado desejável!

Formulação do Problema

- Quais as ações possíveis? (qual o seu efeito sobre o estado?)
- Quais os estados possíveis? (como representá-los)
- Como avaliar os estados?

Problema de Pesquisa

- Solução: sequência de ações

* Fase final corresponde à execução

* Formular \rightarrow Pesquisar \rightarrow Executar

* Muitos dos problemas em computação podem ser formulados como:

- Um conjunto S de estados
- Um estado inicial $s \in S$
- Uma relação de transição T no longo deste espaço de estados

• Um conjunto de estados finais (objetivos) : O ES

- ✗ Um problema pode ser definido formalmente em cinco componentes:
 1. Representação do Estado
 2. Estado inicial (atual)
 3. Estado objetivo (define os estados desejados)
 4. Operadores (move, pré-condições, efeitos, custo)
 5. Custo da solução

Resolução De Problemas

- ✗ O problema pode ser resolvido através de pesquisa e um caminho entre o estado inicial e um estado objetivo
- ✗ Em suma, a formulação do problema envolve decidir que ações e estados a considerar tendo em conta um objetivo

COMPONENTES DE UM PROBLEMA DE PROCURA

- ✗ Estado inicial
- ✗ Ações
- ✗ Modelo de transição
 - Que estado resulta da execução de uma determinada ação em um determinado estado?
- ✗ Estado objetivo
- ✗ Custo do caminho
 - Soma dos custos não negativos de cada etapa
- ✗ A solução ideal é a sequência de ações que forma o menor custo de caminho para alcançar a meta
- ✗ Uma solução para um problema é o caminho do estado inicial para o estado objetivo
- ✗ A qualidade da solução é medida pela função de custo caminho e uma solução "ideal" tem o menor custo possível entre todas as soluções.
- ✗ Estado (abstrato) = conjunto de estados reais
- ✗ Ação (abstrata) = combinação completa de ações reais

- Solução (abstrata) = conjunto de caminhos reais que são soluções no mundo real

Típos De Problemas

- Ambiente determinístico, totalmente observável → problema de estado único
 - O agente "sabe" exatamente o estado em que estará; a solução é uma sequência
- Ambiente determinístico, não acessível → problema de múltiplos estados
 - O agente "não sabe" onde está; a solução é uma sequência.
- Ambiente não determinístico e / ou parcialmente acessível → problema de contingência
 - Perceções fornecem novas informações sobre o estado atual
 - Frequentemente intercalam pesquisa e execução
- Espaço de estados desconhecido → problema de exploração

Pesquisa De Soluções

- Uma solução para um dado problema é definida como a sequência de ações desde o estado inicial até ao estado objetivo. A "qualidade" da solução é medida através da função do custo do caminho e pode ser:
 - Uma solução satisfatória se é uma qualquer solução
 - Uma solução semi-ótima é aquela que tem aproximadamente o menor custo entre todas as soluções
 - Uma solução ótima é aquela que tem o menor custo entre todas as soluções.

Procura

Ideia Básica

- Comegando no estado inicial (mundo) e expandi-lo, fazendo uma lista de todos os possíveis estados sucessores.
- Manter uma lista de estados (mundo) não expandidos
- Em cada etapa, escolher um estado da lista de estados não expandidos para expandir
- Continue até atingir o estado objetivo
- Tente expandir o maior número possível de estados

METODOLOGIA PARA REALIZAR A PESQUISA DA SOLUÇÃO

1. Começar com o estado inicial
2. Executar o teste do objetivo
3. Se não \hookrightarrow foi encontrada a solução, usar os operadores para expandir o estado atual gerando novos estados - sucessores (expansão)
4. Executar o teste do objetivo
5. Se não tivermos encontrada a solução, escolher qual o estado a expandir a seguir (estratégia de pesquisa) e realizar essa expansão
6. Voltar a 4.

ÁRVORE DE PESQUISA

- através do espaço de estados do problema podemos formar uma árvore de pesquisa que nos auxilia a encontrar a solução
- O estado inicial forma o nó raiz das árvores e os ramos de cada nó são as ações possíveis a partir do nó (estado) para as folhas (próximos estados)
- Nós folhar, ou não têm sucessores ou ainda não foram expandidos.

ESTADOS vs NÓS

- Um estado é a representação de uma configuração física
- Um nó é uma estrutura de dados constituída por parte da árvore de pesquisa que inclui: estado, nó pai*, ação**, custo do caminho $g(n)$ ***, profundidade.
- * nó que lhe deu origem
- ** operador aplicado para gerar
- *** custo do caminho desde o nó inicial

GRAFOS DE PROCURA COMO ÁRVORES DE PROCURA

- Árvores são grafos sem ciclos e em que os nodos têm ≤ 1 pai.
- Podemos transformar problemas de pesquisa gráfica (de $S \approx G$) em problemas de pesquisa em árvore:
 - substituindo links não direcionados por 2 links direcionados
 - evitando loops no caminho (ou monitorando os nós visitados)

ALGORITMO DE PESQUISA EM ÁRVORE

- ✗ Inicializar a lista de estados não expandidos usando o estado inicial
- ✗ Enquanto a lista de estados não expandidos não estiver vazia:
 - Escolher um nó da lista de estados não expandidos de acordo com a estratégia de pesquisa e removê-lo da lista.
 - Se o nó contiver o estado do objetivo, devolver a solução
 - Senão, expandir o nó e incluir os seus filhos na lista de estados não expandidos.
- ✗ Para lidar com estados expandidos:
 - Sempre que se expande um nó, adicionar esse estado ao conjunto explorado; não colocar estados explorados na lista de estados não expandidos novamente
 - Sempre que se adiciona um nó à lista de estados não expandidos, verificar se ele já existe na lista de estados não expandidos com um custo de caminho mais alto, se sim, substituir esse nó pelo novo.

ESTRATÉGIAS DE PESQUISA

- ✗ Avaliação através dos seguintes critérios:
 - Completude: Garantido que se encontra solução?
 - Complexidade no tempo: Quanto tempo demora a encontrar a solução?
 - Complexidade no espaço: Quanta memória necessita para fazer a pesquisa?
 - Optimalidade: Encontra a melhor solução?
- ✗ O tempo e a complexidade do espaço são sempre considerados tendo em conta a medição da dificuldade do problema (tamanho do gráfico e espaço dos estados)
- NOTA:
- Tempo e a complexidade do espaço são medidos em termos de:
 - ✗ b : o máximo fator de ramificação (o número máximo de sucessores de um nó da árvore de pesquisa)
 - ✗ d : a profundidade da melhor solução
 - ✗ m : a máxima profundidade do espaço de estados
- ✗ Uma estratégia de pesquisa é definida escolhendo a ordem da expansão do nó
- ✗ Estratégias de pesquisa não-informadas usam apenas as informações disponíveis na definição do problema.

- Nas estratégias de pesquisa informadas dá-se ao algoritmo "dicas" sobre a adequação de diferentes estados.

TIPOS DE ESTRATEGIAS DE PESQUISAS

PESQUISA NÃO INFORMADA (CEBA)

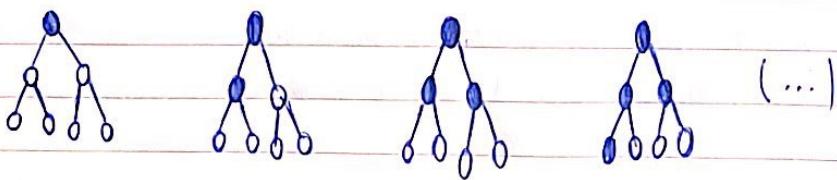
- X Primeira em Largura (BFS)
 - X Primeira em Profundidade (DFS)
 - X Custo Uniforme
 - X Pesquisa iterativa
 - X Pesquisa bidirecional

PESQUISA INFORMADA (HEURÍSTICA)

- X Pesquisa gulosa
 - X Algoritmo A'

PESQUISA PRIMEIRO EM LARGURA (BREATH-FIRST SEARCH)

- Estratégia: Todos os nós de menor profundidade são expandidos primeiro
 - Bom: Pesquisa muito sistemática
 - MAU: Normalmente demora muito tempo e sobretudo ocupa muito espaço
 - Propriedades
 - Completa: Sim, se bloco (atramentação) for finito
 - Tempo: $O(b^d)$ exponencial na profundidade
 - Espaço: Guarda cada nó em memória $O(b^d)$
 - Ottimal: Sim, se o custo de cada passo for 1.
 - Escolhe o 1º elemento de lista de estados não expandidos
 - Adiciona extensões de caminho ao final da lista de estados não expandidos.



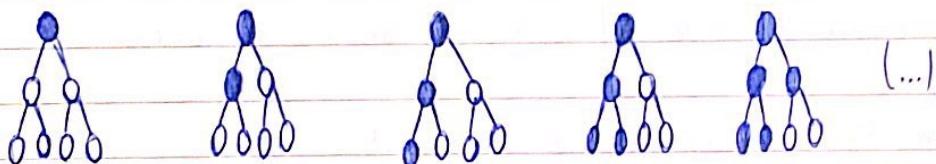
PESQUISA PRIMEIRO EM PROFUNDIDADE (DEPTH-FIRST SEARCH)

- ✗ Estratégia: Expandir sempre um dos nós mais profundos da árvore
- ✗ Bom: Necessita pouca memória, bom para problemas com muitas soluções
- ✗ Ruim: Não pode ser usada para árvores com profundidade infinita, pode ficar presa em ramos errados

✗ Propriedades:

- Completa: Não, faltam espaços de profundidade infinita, com repetição (loops)
 - Modificar para evitar estados repetidos ao longo do caminho
- Tempo: $O(b^m)$, mau se $m > d$ ($m = \text{máxima profundidade}$)
- Espaço: $O(bm)$, espaço linear
- Ótimal: Não (em princípio devolve a 1ª solução que encontra)
- ✗ Por vezes é definida uma profundidade limite e transforma-se em Pesquisa com Profundidade Limitada:

- Escolhe o primeiro elemento da lista de estados não expandidos
- Adicionar extensões de caminho à frente da lista de estados não expandidos.



PESQUISA DE CUSTO UNIFORME

✗ ESTRATÉGIA:

- Para cada nó da lista de estados, guardar o custo total do caminho do estado inicial para esse nó.
 - Expandir sempre o nó com menor custo da lista de estados não expandidos (medido pela função de custo da solução)
 - ✗ Pesquisa Primo em Largura é igual se $g(N) = \text{depth}(N)$
 - ✗ Equivalente a Pesquisa Primo em Largura se todos os custos forem iguais
 - ✗ IMPLEMENTAÇÃO: de listas de estados não expandidos é uma fila prioritária ordenada pelo custo do caminho
 - ✗ Temos de garantir que $g(\text{sucessor}) \geq g(N)$
 - ✗ Equivalente ao algoritmo de Dijkstra em geral.

X PROPRIEDADES:

- Completa: Sim, se o custo da etapa for maior que alguma constante positiva
- E (não queremos sequências infinitas de etapas com um custo total finito)
- Tempo:
 - Número de nós com custo de caminho \leq custo da solução ideal (C^*), $O(b^{C^*})$
 - Pode ser maior que $O(bd)$, a pesquisa pode explorar caminhos mais longos que consistem em pequenos passos antes de explorar caminhos mais curtos que consistem em passos maiores
 - Espaço: $O(b^{C^*})$
 - Ottimal: Sim

PESQUISA ITERATIVA - APROFUNDAMENTO PROGRESSIVO

- Se não conhecermos o valor limite máximo, estaremos condenados a uma estratégia de procura em profundidade primeiro e teremos que lidar com o problema de eventuais caminhos infinitos. A resposta passa pela alteração do princípio da procura limitada fazendo varrer esse limite entre zero e infinito.
- Usar uma pesquisa Primeiro em Profundidade (DFS) como uma sub-rotina:
 - Verificar raiz
 - Fazer um DFS procurando um caminho de comprimento 1
 - Se não houver um caminho de comprimento 1, fazer um DFS para procurar um caminho de comprimento 2
 - Se não houver um caminho de comprimento 2, fazer um DFS para procurar um caminho de comprimento 3
 - (...)

• ESTRATEGIA: Executar pesquisa em profundidade limitada, iterativamente, aumentando sempre o limite da profundidade.

X PROPRIEDADES:

- Completo: Sim
 - Tempo: $O(b^d)$
 - Espaço: $O(b^d)$
 - Ottimal: Sim, se o custo for 1
- Em geral, é a melhor estratégia não-informada para problemas com um grande espaço de pesquisa

e em que a profundidade da solução não é conhecida.

PESQUISA BI-DIRECIONAL

✗ **ESTRÉGIA:** Executar pesquisa para a frente desde o estado inicial e para trás desde o objetivo, simultaneamente

- **BEN:** Pode reduzir enormemente a complexidade no tempo $O(b^{d/2})$

• **PROBLEMAS:** Será possível gerar os predecessores? E se existirem muitos estados objetivo? Como fazer o "matching" entre as duas pesquisas? Que tipo de pesquisa fazer nas duas metades?

HEURÍSTICAS

✗ Como técnica de procura para a obtenção de metas em problemas não algorítmicos que geram "explosões" combinatoriais;

✗ Como um método aproximado de resolução de problemas utilizando funções de avaliação de tipo heurística

✗ Como um método de poda (corte) para estratégias de programas de jogos

✗ Numa procura, podemos aplicar dois tipos genéricos de heurísticas, sobre a decisão sobre qual nó será feita a expansão e sobre a decisão sobre quais os nós que devem ser descartados.

• Se o universo é totalmente conhecido, a heurística será realizada através da atribuição de números

• Se o universo não é totalmente conhecido, no qual a heurística será realizada através da aplicação de regras

✗ As heurísticas são específicas para cada problema

PESQUISA GULOSA - GREEDY-SEARCH

✗ **ESTRÉGIA:**

- Expandir o nó que parece estar mais perto da solução

✗ $h(n)$ = custo estimado ao caminho mais curto do estado n para o objetivo (função heurística)

✗ **PROPRIEDADES:**

- Completa: Não, pode entrar em ciclos (susceptível a falsos caminhos)

• Complexidade no tempo: $O(b^m)$ (com uma boa função heurística pode diminuir consideravelmente)

• Complexidade no espaço: $O(b^m)$ (mantém todos os nós na memória)

X ÓTIMA: Não

X Necessário detectar estados repetidos

PESQUISA A*

X ESTRATÉGIA:

- evitam expandir caminhos que são caros
 - o algoritmo A* combina a pesquisa gulos com a uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até à solução
- $$f(m) = g(m) + h(m)$$
- \downarrow \downarrow
custo estimado para chegar ao objetivo
custo do percurso até agora

PESQUISA COM MEMÓRIA LIMITADA - IDA*/SMA*

X IDA* - Pesquisa com Profundidade Iterativa

- Estratégia: utilização de um custo limite em cada iteração e realização de pesquisa em profundidade iterativa.
- Problemas em alguns problemas reais com jumps de custo com muitos valores

X SMA* - Pesquisa Simplificada com memória limitada

- IDA* de uma iteração para a seguinte só se lembra do custo limite, SMA* utiliza toda a memória disponível para evitar repetições.
- Estratégia: Quando necessita de gerar um sucessor e não tem memória, esquece um nó da fila que pareça pouco prometedor (com custo alto)

ALGORITMOS DE PESQUISA LOCAL E PROBLEMAS DE ORGANIZAÇÃO

- Nem todos os ambientes permitem que a sua solução seja uma sequência de ações
- Estes algoritmos efectuam pesquisa puramente local no espaço de estados, avaliando e modificando um ou mais estados atuais, em vez de explorar sistematicamente caminhos a partir de um estado inicial
- São adequados para problemas nos quais tudo o que importa é o estado da solução, não o custo do caminho para alcançá-lo

ALGORITMOS DE MELHORIA ITERATIVA

- X Em muitos problemas de otimização, o caminho para o objetivo é irrelevante
- X Espaço de estados = conjunto de configurações completas
- X Algoritmos iterativos mantêm um único estado (currente) e tentam melhorá-lo

- ESTRATÉGIA: Começar como uma solução inicial do problema e fazer alterações de forma a melhorar a sua qualidade

PESQUISA SUBIDA DA COLINA (HILL-CLIMB SEARCH)

- X Algoritmo clássico, bastante eficiente na tarefa de encontrar máximos ou mínimos locais, pela exploração.
- X ESTRATÉGIA:
 - iniciar num ponto aleatório X e fazer a sua avaliação
 - mover do ponto X para um seu vizinho Y
 - se Y for uma solução melhor que X, ficar com ele e repetir o processo, caso contrário, tentar outro vizinho de X.
- X Dificilmente encontrará a solução ideal (a menos que se tenha sorte no ponto de inicialização)

ARREFECIMENTO SIMULADO (SIMULATED ANNEALING)

- X Inspirado pela natureza.

X ESTRATÉGIA:

- iniciar num ponto aleatório X e fazer a sua avaliação
- fazer o movimento até um vizinho Y e avaliar esse novo ponto.
- Se os resultados melhoraram no ponto Y, mover até ele e refazer o processo, caso contrário mover apenas se a probabilidade de ir para um ponto negativo seja superior a um número aleatório.

$$\text{probabilidade (p)} = \text{Exp}(-\Delta E / T) \quad (T \equiv \text{temperatura})$$

- Assim sendo, nas primeiras iterações, quando T está elevado, a probabilidade de aceitar valores negativos é maior. Com o passar do tempo, este algoritmo passa a ter um comportamento como o Hill-Climb.

PESQUISA TABU (TABU SEARCH)

- ✗ Meta-heurística é um procedimento adaptivo auxiliar que usa um algoritmo de pesquisa local na exploração contínua dentro de um espaço de pesquisa
- ✗ A ideia básica da Pesquisa Tabu é penalizar movimentos que levam a solução para espaços de pesquisa já visitados. Aceita de forma determinística soluções que não melhoraram para evitar ficar preso em mínimos locais.
- ✗ ESTRATÉGIA:
 - Idéia chave: manter a sequência de nós visitados (Lista tabu)
 - Partindo de uma solução inicial, a pesquisa move-se, a cada iteração, para a melhor solução na vizinhança, não aceitando movimentos que levam a soluções já visitadas, esses movimentos conhecidos ficam armazenados numa lista tabu.
 - A lista permanece na memória guardando soluções já visitadas durante um determinado espaço de tempo ou certo número de iterações. Como resultado final é esperado que se encontre um ótimo global ou próximo disso.

T6 - PESQUISA EM CONTEXTOS COMPETITIVOS (JOGOS)

Jogos Como Problemas De Pesquisa

- ✗ Podem existir situações em que um agente na procura de soluções num espaço de procura (o que ocorre habitualmente em jogos)
- ✗ Em ambientes multiagentes como estes, cada agente necessita considerar as ações dos outros agentes e como estas os afetam.
- ✗ A imprevisibilidade destes agentes pode colocar contingências no processo de resolução de problemas por pesquisa do agente.
- ✗ Estes ambientes competitivos, nos quais os objetivos dos agentes estão em conflito, dão lugar a problemas de procura com adversários (adversarial search problem)
- ✗ A Teoria de Jogos considera qualquer ambiente multiagente como um jogo, em que o impacto de um agente nos outros é "significativo", independentemente se os agentes são competitivos ou cooperativos.

Jogo vs Pesquisa De Agente Unico

X Não se sabe como o adversário vai agir

- A solução não é uma sequência fixa de ações do estado inicial para o estado do objetivo, mas uma estratégia ou política

X Eficiência é fundamental para jogar bem

- O tempo para fazer uma mudança é limitado.

• O fator de ramificação, a profundidade da pesquisa e o número de configurações do término são enormes.

X Tipos De Jogos:

• Informação:

→ Perfeita: Xadrez, Damas, Go, Otelo, Gomão, Monopólio

→ Imperfeita: Poker, Scrabble, Bridge, King

• Sorte / Determinístico:

→ Determinístico: Xadrez, Damas, Go, Otelo

→ Jogo de Sorte: Gomão, Monopólio, Poker, Scrabble, Bridge, King

X PLANO DE "ATAQUE":

• Algoritmo para o jogo perfeito

• Horizonte finito, avaliação aproximada

• Cortes nas árvores para reduzir custos (pruning)

X CARACTERÍSTICAS:

• Agente Hostil (adversário) incluído no mundo

• Oponente Imprevisível → Solução é um Plano de Contingência

• Tempo Limite → Pouco provável encontrar o objetivo (necessária uma aproximação)

X Considerando um jogo de informação perfeita com dois adversários podemos defini-lo formalmente como um problema de pesquisa através dos seguintes elementos:

• Estado INICIAL (posição do tabuleiro e qual o próximo jogador a jogar)

• CONJUNTO DE OPERADORES (definem os movimentos legais)

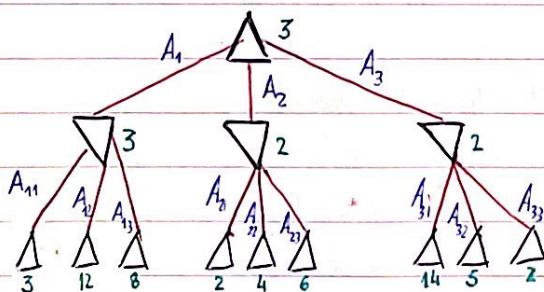
• TESTE TERMINAL (que determina se o jogo acabou)

• FUNÇÃO DE UTILIDADE (dá um valor ao resultado; por exemplo $1 \rightarrow V / 0 \rightarrow E / -1 \rightarrow D$)

ESTRÉGIAS IDEIAS

- Depende do MIN e, portanto, o MAX deve encontrar uma estratégia contingente que especifique:
 - o movimento do MAX no estado inicial
 - em seguida, o movimento do MAX nos estados resultantes de todas as respostas possíveis do MIN
 - então os movimentos de MAX nos estados resultantes de toda a resposta possível de MIN a esses movimentos.
- Uma estratégia ideal leva a resultados pelo menos tão bons quanto qualquer outra estratégia quando se joga com um oponente inviolável
- O algoritmo Minimax pode ser aplicado como estratégia de resolução neste tipo de jogos (determinísticos, com dois adversários e informação perfeita) e consiste em:
 - Gerar a árvore completa até aos estados terminais
 - Aplicar a função utilidade a esses estados
 - Calcular os valores da utilidade até à raiz da árvore, uma camada de cada vez.
 - Escolher o movimento com valor mais elevado

ÁRVORE DO JOGO



- VALOR MINIMAX DE UM NÓ: a utilidade (para MAX) de estar no estado correspondente, assumindo o desempenho perfeito dos dois lados

ESTRÉGIA MINIMAX: escolher a jogada que oferece o melhor retorno do pior caso.

MINIMAX (nó) = \rightarrow utilidade (nó) se o nó for terminal

\rightarrow Maxaction Minimax (Sucessor (nó, ação)) se player = MAX

\rightarrow minaction Minimax (Sucessor (nó, ação)) se player = MIN

MINIMAX VALOR

- ✗ Dada uma árvore de jogo, a estratégia ideal pode ser determinada examinando o valor minimax de cada nó (MINIMAX-VALUE (m))
- ✗ O valor de minimax de um nó é a utilidade de estar no estado correspondente, supondo que os dois jogadores joguem da melhor maneira dali para o final do jogo.
- ✗ Com uma opção, MAX prefere passar para um estado de valor máximo, enquanto MIN prefere um estado de valor mínimo.

PROPRIEDADES DO MINIMAX

✗ PROPRIEDADES:

- Completo? Sim, se a árvore for finita.
- Ótimo? Sim, contra um adversário ótimo.
- Complexidade no tempo? $O(b^m)$
- Complexidade no espaço? $O(b^m)$

✗ PROBLEMA:

- Invável para qualquer jogo minimamente complexo.

ALPHA-BETA PRUNING

- ✗ É possível calcular a decisão minimax sem expandir todos os nós na árvore do jogo.
- ✗ α é o melhor valor para MAX encontrado até ao momento.
- ✗ Se V for pior que α , MAX deve evitá-lo \rightarrow cortar o ramo.
- ✗ β é definido da mesma forma para MIN.

DECISÕES IMPERFEITAS EM TEMPO REAL

- ✗ O algoritmo minimax gera todo o espaço de pesquisa do jogo, enquanto que o algoritmo alpha-beta permite remover grande parte dele. No entanto, este ainda precisa pesquisar até aos estados terminais em pelo menos uma parte do espaço de pesquisa.
- ✗ Essa profundidade geralmente não é prática, porque as mudanças devem ser feitas em um período de tempo razoável.
 - Função de avaliação: Utilidade (interesse) estimada para a posição
 - Teste de corte: profundidade limite
- ✗ Surge eventualmente outro problema: Problema do horizonte

- Tabelas de transposição para armazenar estados expandidos anteriormente
- Pruning direto para evitar considerar todos os movimentos possíveis
- Tabelas de pesquisa para movimentos de abertura e jogos finais

MONTE CARLO SEARCH TREE

- ✗ Jogos com árvores profundas, grande fator de ramificação e sem boas heurísticas
- ✗ Em vez de pesquisar com profundidade limitada com uma função de avaliação, usar simulações aleatórias
- ✗ Começando no estado atual (raiz da árvore de pesquisa), iterar:
 - Selecionar um nó folha para expansão usando uma política de árvore (combinação de descoberta e exploração)
 - executar uma simulação usando uma política padrão (por exemplo, movimentos aleatórios) até que um estado terminal seja alcançado
 - propagar movimente o resultado para atualizar as estimativas de valor dos nós internos da árvore.

	Determinístico	Estocástico
Informação perfeita (totalmente observável)	Xadrez, Go, Damas	Gamão, Monopólio
Informação imperfeita (parcialmente observável)	Batalha Naval	Scrabble, Poker, Bridge

Jogos Estocásticos

- ✗ Tipicamente combinam habilidade e sorte
- ✗ Árvore de pesquisa deve incluir nós de probabilidade
- ✗ Decisão é efetuada com base no valor esperado
- ✗ Algoritmo Expect-Max

EXPECTIMINIMAX

X Todos os métodos anteriores são ótimos:

- Pruning alpha-beta
- Função de avaliação
- etc...

X A complexidade computacional é muito complicada

- Fator de ramificação da escolha aleatória pode ser alto.
- O dobro dos "níveis" da árvore

X Sorte (chance) soma os valores dos estados sucessores ponderados pela probabilidade de cada sucessor

- Fator de ramificação pode ser muito desagradável, definindo funções de avaliação e algoritmos de pruning mais difíceis.

X Eventualmente usar Simulação Monte Carlo: quando se chega a um mês sorte, simula-se um grande número de jogos com jogadas aleatórias de dados e usar a percentagem de ganhos como uma função de avaliação

JOGOS ESTOCÁSTICOS COM INFORMAÇÃO IMPERFEITA (PARCIAL)

X Os jogos de cartas fornecem um dos melhores exemplos de jogos estocásticos com informação imperfeita, onde as informações não conhecidas advêm de aleatoriedade.

TÉCNICAS

X Se conhecemos as probabilidades de diferentes configurações e desejarmos maximizar os ganhos médios (por exemplo, podemos jogar o jogo muitas vezes): expectiminimax

X Se não temos ideia das probabilidades de diferentes configurações, ou, se podemos apenas jogar uma vez e não podemos (nem queremos) perder: minimax

X Se a informação desconhecida for selecionada intencionalmente pelo oponente: teoria dos jogos

T7 - Representação do Conhecimento e Raciocínio

CONHECIMENTO E RACIOCÍNIO

- ✗ O que é o conhecimento?
 - Pode ser definido como informação sobre ambiente (pode ser expresso na forma de proposições)
- ✗ O que é representação do conhecimento?
 - Símbolos usados para representar informação sobre ambiente (as proposições)
- ✗ O que é representação e raciocínio do conhecimento?
 - Manipulação de símbolos (codificam proposições para produzir representações de novas proposições)

AGENTES BASEADOS EM CONHECIMENTO (KNOWLEDGE-BASED AGENTS)

- ✗ Os humanos sabem "coisas", o que os ajuda a fazer "coisas"
 - Processos de raciocínio que operam em representações intérpretes de conhecimento.
- ✗ Lógica: uma classe geral de representações para apoiar agentes baseados em conhecimento.
 - Combinase e recombinase informações para atender a uma infinidade de finalidades
- ✗ Os agentes baseados em conhecimento podem aceitar novas tarefas na forma de objetivos explicitamente descritos
 - "Ouvir" ou aprender novos conhecimentos sobre o meio ambiente
 - Adaptando-se às mudanças no ambiente, atualizando conhecimento relevante.

CONHECIMENTO vs IMPLEMENTAÇÃO

- ✗ Um agente baseado em conhecimento pode ser descrito no nível do conhecimento
 - É preciso especificar o que o agente sabe e quais são os seus objetivos
- ✗ **ABORDAGEM DECLARATIVA** para construção do sistema: TELL ao agente o que ele precisa de saber.
- ✗ **IMPLEMENTAÇÃO:** estruturas de dados dentro da KB e algoritmos
 - Abordagem procedural: codificar comportamentos diretamente como código do programa

REPRESENTAÇÃO, RACIOCÍNIO E LÓGICA

- ✗ Representação do conhecimento procura representar o conhecimento de forma a que seja manipulável pelo computador.
- ✗ Lógicas são linguagens formais para representar informação de forma a que conclusões possam ser tiradas
- ✗ Sintaxe define as possíveis frases de uma linguagem
- ✗ Semântica define os factos do mundo a que as frases se referem (significado)

REPRESENTAÇÃO DO CONHECIMENTO

- ✗ Procura responder a questões tais como:
 - Como representar conhecimento?
 - Qual é a natureza do conhecimento e como o representarmos?
 - Será que representarmos esse conhecimento todos de igual forma?
 - Um esquema de representação deve lidar com um domínio específico ou deve ser de uso geral?
 - Quão expressivo é um esquema de representação?
 - O esquema deve ser declarativo ou procedural?
 - Como devem os programas ditos "inteligentes" representar e usar esse conhecimento?
 - Seremos capazes de representar todo o tipo de conhecimento?
- ✗ Características desejáveis, entre outras:
 - Definir explicitamente os objetos e as suas relações;
 - Exibir as limitações e restrições (expressar a forma como um objeto ou relação os define)
 - Transparente
 - Rápida
 - Computável

PROPRIEDADES

- ✗ Adequação da representação
 - capacidade de representar o conhecimento necessário
- ✗ Adequação da inferência
 - Capacidade de manipular conhecimento e "produzir" novos conhecimentos
- ✗ Eficiência da inferência
 - capacidade de direcionar a inferência para direções produtivas

- capacidade de responder com recursos limitados
- ✗ Esciências na aquisição de novo conhecimento
- Capacidade de "adquirir" novo conhecimento
- de forma automática (se possível)

Lógica

- ✗ Linguagem com regras concretas e consistentes
 - Nenhuma ambiguidade na representação
 - Permite comunicação e processamento inequívocos
 - Muito diferente das línguas
- ✗ Muitas maneiras de traduzir entre línguas
 - Uma declaração pode ser representada em diferentes lógicas
- ✗ A expressividade de uma lógica
 - Quanto podemos dizer neste língua?
- ✗ Não confundir com raciocínio lógico.

REGRAS DE PRODUÇÃO

- ✗ Conjunto de regras de pares < condição, ação>
- ✗ "SE Condição ENTÃO Ação"
 - Modulares
 - Mais fácil expansão
 - Ativação pelo estado do sistema
 - Próximas do modelo cognitivo
- ✗ VANTAGENS:
 - MODULARIDADE - cada regra define uma parte do conhecimento, sendo independente
 - INCREMENTABILIDADE - novas regras podem ser acrescentadas em qualquer momento
 - ALTERABILIDADE - as regras podem ser alteradas em qualquer momento
 - INDEPENDÊNCIA do sistema de inferência utilizado
 - Facilidade em gerar explicações para uma dada resposta:
 - Como se chegou a uma dada conclusão? (questões "como?")
 - Porque é que estamos interessados nesta informação? (questões "porque?")

UTILIZAÇÃO DE REGRAS DE PRODUÇÃO

- ✗ A base de conhecimento é feita de regras e factos.
- ✗ Considera-se a sintaxe lógica:
 - se Condigão então Conclusão
- ✗ Um or condigão pode ser:
 - um predicado lógico
 - uma conjunção de duas condigões
 - uma disjunção de duas condigões
- ✗ Representar factos (dados): facto(X)

SISTEMAS DE INFÉRÉNCIA

- ✗ BACKWARD CHAINING
 - de uma questão (hipótese) o raciocínio retrocede na cadeia de inferência até aos factos que a suportam
 - das conclusões às condigões
- ✗ FORWARD CHAINING
 - todas as conclusões (exaustivamente) possíveis de se provar (derivadas) são inseridas na base de conhecimento como factos
 - com todos os factos representados na base de conhecimento, "basta" provar (confirmar) a existência da conclusão

INCERTEZA EM REGRAS DE PRODUÇÃO

- ✗ Até agora não são considerados tipos de informação de valor intermédio (exemplo: pouco provável, provável, altamente provável, ...)
- ✗ No mundo real isto é irrealista
- ✗ Os sistemas têm de saber lidar com a incerteza | grau de risco, probabilidade, confiança, ...)
- ✗ Associa-se a cada proposição um grau de confiança
 - Factos: facto(Proposição)::C.
 - Regras: se Condigão então Ação :: C
 - sendo C um número entre 0 e 1. (0 - probabilidade 0%; 1 - probabilidade 100%)

PROGRAMAÇÃO DIRIGIDA AOS PADRÕES

- Arquitetura de programação baseada em padrões de dados que ativam um ou mais módulos.
- Um programa orientado a padrões é um conjunto de módulos.
- Cada um deles é definido por uma pré-condição e uma ação a ser executada sempre que os dados do problema formarem essa pré-condição verdadeira.
- Desta forma, a execução dos módulos é ativada por padrões existentes nos dados, não havendo, como acontece nos sistemas convencionais, um esquema pré-definido de invocação.

ESTRUTURAS HIERÁRQUICAS

- O objetivo é fazer uma compactação dos factos a representar num dado sistema.
- As entidades podem agrupar-se em classes, partilhando valores para os mesmos atributos.
- Os factos associados a um dado objeto podem não estar representados ao seu nível, mas sim serem reconstruídos através de um processo de inferência por herança, sobre as classes superiores.

• REDES SEMÂNTICAS

→ correspondem a um grafo, onde os nodos definem as entidades (objetos, classes) do sistema e os ramos definem relações entre as mesmas. Algumas destas relações são chamadas relação é-um (isa) e permitem a herança do conhecimento definido numa dada entidade.

• ENQUADRAMENTOS (FRAMES)

→ definem objetos (ou classes), cada um deles com uma designação e um conjunto de slots, correspondentes a atributos, onde é colocado um valor. Alguns destes atributos são utilizados para representar as relações entre um objeto e uma classe à qual este pertence e as relações entre classes e superclasses, relações estas que possibilitam a herança de um valor de um dado slot não preenchido.

REDES SEMÂNTICAS

- Redes de entidades e de relacionamento entre elas.
- Um grafo:
 - cada nodo corresponde a uma entidade
 - os ramos correspondem às relações e são etiquetados com o nome da relação.

X Tipos de relações: é-um, desloca-se, aberto, etc...

FRAMES

- X Técnica de representação do conhecimento que tenta organizar conceitos de uma forma que explora interações e crendices comuns
- X Análogo à POO
- X Estruturas de dados cujos componentes se chamam slots
- X Os Slots são identificados por nomes e denotam informações de vários tipos: valores simples, referências de outras frames, procedimentos, ...
- X is-a: relação de Classe / Superclasse
- X instance-of: relação de membro de uma classe
- X Representação: frame (Frame, Slot, Valor).

SCRIPT

- X Um script é uma estrutura de dados usado para representar uma sequência de eventos
- X São usados para interpretar histórias
- X Caracteriza-se:
 1. Uma cena
 2. Adereços (objetos manipulados no script)
 3. Atores (agentes que podem mudar o estado do mundo)
 4. Eventos
 5. Atos: conjunto de ações dos atores
- X Em cada cena um ou mais atores realizam ações. Os atores agem com os adereços. O script pode ser representado como uma árvore ou rede de estados, impulsionada por eventos
- X Assim como os Frames, orientam a interpretação, dizendo ao sistema o que procurar e onde procurar. Pode prever eventos.

SISTEMAS BASEADOS EM CONHECIMENTO

- X Programas de computador que utilizam o conhecimento representado explicitamente para resolver problemas.
- X Manipulam conhecimento e informação de forma inteligente
- X São desenvolvidos para resolverem problemas que requerem grandes porções de conhecimento humano

e especialização (perso:a).

X CONHECIMENTO + RACIOCÍNIO = RESOLUÇÃO PROBLEMA

DEFINIÇÃO

X Perspetiva do Conhecimento processável pelo homem

- A análise e modelação do método de resolução do problema

X Perspetiva simbólica processável pelo computador

- A actividade de representar este método através de um formalismo computacionalmente eficiente.

X Capacidade de Raciocínio / Inferência

- É a capacidade de definir um conjunto de passos para a resolução eficiente e rápida de um problema

- O próprio mecanismo de inferência é conhecimento

DIFERENÇAS PARA OUTROS SISTEMAS

SISTEMAS CONVENCIONAIS

Estruturas de Dados

Dados e relação entre eles

Algoritmos Determinísticos

Explicação de raciocínio difícil

Conhecimento Embutido no código

SISTEMAS BASEADOS EM CONHECIMENTO

Representação de Conhecimento

Conceptos, relações entre conceitos e regras

Pesquisa com Heurística

Podem e devem explicar o raciocínio

Conhecimento representado explicitamente e separado do programa

TB - Conhecimento Imperfeito

X Uma entidade que consegue todo o conhecimento (factos e regras) acerca do ambiente que o rodeia, usando uma qualquer aproximação (lógica) tem a capacidade de derivar planos (cujo sucesso é garantido)

X Mas:

- Muito raramente uma qualquer entidade tem acesso a todos os factos (verdades) acerca do seu ambiente.

- O que implica que são obrigados a actuar constantemente numa base de

incerteza / imprecisão.

FONTES

- ✗ Dados incertos
- ✗ Conhecimento incerto (mão confiável)
- ✗ Representação incerta do conhecimento
- ✗ Processo de inferências

INCERTEZA vs IMPRECISÃO

✗ INCERTEZA:

- O predicado está bem definido, mas não se conhece o seu valor verdadeiro
- Raciocínio estatístico - Redes Bayesianas - que são utilizadas para tratar da incerteza por probabilidade.

✗ IMPRECISÃO:

- O predicado é vago por si
- Raciocínio dos Conjuntos Vagos - Fuzzy - estes tratam da imprecisão (possibilidade)

✗ MISTOS:

- Em alguns domínios coexistem os dois tipos de incerteza: Imprecisão e Probabilidade

MÉTODOS DE TRATAMENTO DA IMPERFEIÇÃO DO CONHECIMENTO

MÉTODOS QUANTITATIVOS

- ✗ Hipótese de mundo fechado
- ✗ Circumscrição
- ✗ Lógica e raciocínio Default
- ✗ Truth maintenance system
- ✗ Modelos Combinacionistas
- ✗ ...

Destacam-se em ambientes de existência de conhecimento.

MÉTODOS QUANTITATIVOS

- ✗ Teoria das probabilidades
- ✗ Fatores de certeza
- ✗ Redes Bayesianas
- ✗ Fuzzy Sets and Fuzzy Logic

Destacam-se em ambientes de incerteza / imprecisão de conhecimento.

Bases De Conhecimento

- ✗ Pressuposto dos Nomes Únicos
- ✗ Pressuposto do Mundo Aberto
- ✗ Pressuposto do Domínio Aberto

Monotonia vs Não Monotonia

X Monotonia

- Não admite contradição com conclusões anteriores

X Não Monotonia

- Justifica-se pela:
 - Consideração de pressupostos temporários
 - Obtenção de conclusões plausíveis
 - Flexibilização da evolução do conhecimento
 - Dificuldade na representação completa do conhecimento

Programação Em Lógica Estendida

- ✗ A programação em lógica determina a veracidade ou falsidade de questões:

- $\text{voa}(X) \leftarrow \text{ave}(X)$
- $\text{não-voa}(X) \leftarrow \text{avestruz}(X)$

- ✗ A extensão à programação em lógica permite representar explicitamente a informação falsa:

- $\neg \text{voa}(X) \leftarrow \text{avestruz}(X)$

Negação Da Falha Na Prova vs Negação Clássica

- ✗ Negação por falha na prova:

- atravessar \leftarrow não combóio
- atravessar $\leftarrow \neg$ combóio

INFÉRNCIA / Raciocínio

- ✗ Geralmente, ao responder a uma questão $q(X)$ é:

- verdadeira se $\exists x : q(x)$
- falsa se $\exists x : \neg q(x)$
- desconhecida se $\neg \exists x : q(x) \vee \neg \neg q(x)$

REPRESENTAÇÃO DE CONHECIMENTO IMPERFECTO

X VALORES NULOS:

- Incerto

(I)

Desconhecido, genericamente

- Impreciso

(II)

Desconhecido, mas de um conjunto determinado de hipóteses

- Interditado

(III)

Desconhecido e não permitido conhecer.

BASE DE CONHECIMENTO

FILHOS	PAIS
Jáio	José
José	Manuel
Carlos	José
Belém	Alguém (I)
Maria	{Faria, Garcia} (II)
Bebé	Julio (III)

[FORMALIZAÇÃO PHF] $\rightarrow \text{filho(F, P)} \leftarrow \text{mão filho(F, P)} \wedge \text{não exceção(F, P)}$

[VALORES NULOS TIPO (I)] $\text{filho(belém, alguém).}$
INCRITO
[VALORES NULOS (II)] $\text{exceção(F, P)} \leftarrow \text{filho(F, alguém)}$

[VALORES NULOS (III)] $\text{exceção(maria, faria).}$
TIPO IMPRECISO

$\text{exceção(maria, gracial).}$
 $\text{filho(bebe, julio).}$
 $\text{exceção(F, P)} \leftarrow \text{filho(bebe, P).}$
 mão(bebel).
 $\leftarrow \text{filho(f, julio)} \wedge \text{não mão(F)}$

Representação do Conhecimento

Estática do Conhecimento

Representação Dinâmica do Conhecimento

SISTEMA DE INFÉRÉNCIA

X Sistema de inferência (ínter - predicado):

s!: Questão x Resposta

- Verdadeiro:

$\exists x : q(x)$

- Falso:

$\exists x : \neg q(x)$

- Desconhecido:

$\neg \exists x : q(x) \vee \neg q(x)$

• s!: (Questão, verdadeiro) :- Questão.

• s!: (Questão, falso) :- - Questão.

• s!: (Questão, desconhecido) :-

não (Questão),

não (- Questão).