# IIA – 2015/2016 – RESUMOS TEÓRICA


## Miguel Costa 21220739


## Índice

# CAP. 2 - AGENTES

**Agente Inteligente** – Entidades que agem de forma racional.

**Agente Racional** – um agente que escolhe a acção correta. Toma decisões racionais. O objectivo é imitar o humano.

Podemos considerar quatro tipos de Agentes:

- Agentes Reactivos
- Agentes Reactivos com Estado Interno
- Agentes Guiados por Objectivos
- Agentes Baseados em Funções de Utilidade

## Agentes Reactivos

- **Respondem** a cada percepção **sempre da mesma forma**, tomando em linha de conta apenas a percepção mais recente.
- Simulam reflexos adquiridos ou inatos.

## Agentes Reactivos com Estado Interno (memória)

- Respondem mesma percepção de forma eventualmente diferente, **combinando a percepção mais recente co informação acerca do estado anterior do ambiente**.
- A sua actualização requer conhecimento sobre:
    - Como se modifica o mundo ao longo do tempo
    - Efeito que as acções têm no estado do mundo

**As percepçoes são combinadas com estado interno e conhecimento.**

## Agentes Guiados por Objectivos

- respondem a uma percepção de forma a atingirem um dado objectivo e combinam essa percepção com informação acerca do estado anterior do ambiente.

- Uma decisão deste tipo **considera o resultado futuro**:

    - O que acontece se virar à esquerda? E à direita? Isso será bom para o meu objectivo ?

**Se o objectivo for alterado**, o agente altera o seu comportamento.

Raciocínio para previsão de um estado futuro.

### Agentes Baseados em Funções de Utilidade

- Respondem a uma percepção de forma a atingirem um dado objectivo **maximizando o grau de sucesso** obtido na prossecução desse objectivo.

- A Função Utilidade:
    - Associa valores numéricos a estados, representando o grau de satisfação
    - Permite **optar pela melhor solução** de entre várias

Previsão de um Estado Futuro. Esse novo estado tornar-me-á feliz? **Quanto?**

## AMBIENTE

### Os agentes estão inseridos num ambiente:

A resolução do problema depende das características do ambiente.

Tal como os agentes, também os ambientes podem classificar-se em vários tipos:

- Acessível
- Determinístico
- Episódico
- Dinâmico
- Discreto

## Ambientes Acessíveis/Não Acessíveis

Se o conjunto de sensores do agente lhe der **acesso ao estado completo do ambiente.**

Caso contrário diz-se não acessível.

Exemplo:

- XADREZ – O agente consegue obter toda a informação relevante para a tomada de decisão – acessível.
- robô que seleciona peças – não acessível

## Ambientes Determinísticos/Estocásticos

O ambiente é determinístico se o seu próximo estado puder ser completamente determinado a partir do seu estado actual e da acção a executar. Ou seja, previsível. Caso contrário diz-se estocástico.

Exemplos:

- xadrez – determinístico
- robô que seleciona peças – não-determinístico

## Ambientes Episódicos/Não episódicos

Se a experiência do agente for dividida em episódios. É não-episódico no caso contrário.

- Cada episódio consiste numa percepção seguida de uma acção.
- O sucesso dessa acção depende apenas do episódio actual.
- Os ambientes episódicos tendem a gerar agentes mais simples, porque estes não precisam de “pensar no futuro”.

Exemplos:

- xadrez – não-episódico
- robô que seleciona peças – episódico

### Ambientes Dinâmicos/Estáticos

O ambiente diz-se **dinâmico** se puder mudar enquanto o agente se encontra a decidir. Caso contrário, diz-se **estático**.

Mais complexos, alteram com o tempo.

Se o ambiente não mudar com o tempo mas o desempenho do agente sim, o ambiente diz-se semi-dinâmico (xadrez com tempo controlado).

Exemplos:

- Xadrez sem relógio – estático
- Xadrez com relógio – semi-dinâmico
- robô que seleciona peças – dinâmico

### Ambientes Discretos/Contínuos

Diz-se discreto quando origina séries de percepções e acções perfeitamente distintas umas das outras. Caso contrário, diz-se contínuo.

Existe um número finito de estados ou de percepções/acções.

Exemplos:

- Jogo de Poker: Discreto
- Condução de um veículo: Contínuo
- robô que seleciona peças – contínuo

**Conclusão:**

Os ambientes mais complexos são os ambientes inacessíveis, não-determinísticos, não-episódicos, dinâmicos e contínuos.

## CAP. 3 - RESOLUÇÃO DE PROBLEMAS

## Agentes de Resolução de Problemas

- São do tipo Guiado por Objectivos goal-oriented).

- Algoritmo Geral de Pesquisa – *General Problem Solving - GPS*)

### *1.* Formulação do Objectivo

**Exemplo:**

Pretendemos viajar de Lisboa para o Porto.

- Estados - são as diversas cidades que constam num mapa conhecido do agente;
- Objectivo - Porto;
- Estado inicial - Lisboa;
- Solução - Qualquer sequência de cidades a percorrer partindo de Lisboa até chegarmos ao Porto, é uma solução do problema (podem existir diversas soluções!).

### 2. Formulação do Problema

### 3. Pesquisa da Solução

- A Pesquisa corresponde **à procura de um caminho** que conduz à solução, **ou à procura do melhor caminho** de entre todos os possíveis.

- **Componentes de um Problema:**

Um Problema é composto por:

· **Estado Inicial**: descreve o estado de que o agente parte;
· **Estado Final**: descreve o objectivo;
· **Operadores**: determinam consequências das possíveis acções, informando o estado atingido após cada acção

· Função Sucessores: devolve o conjunto de estados S(x) que podem ser alcançados a partir do estado x.

· **Teste de Satisfação do Objectivo**: testa se o estado actual coincide com o Estado Final.
· **Custo** do Caminho (opcional): calcula o custo de uma dada solução (em horas, em Km, em nº de movimentos, etc.).

### Espaço de Estados

Conjunto de todos os estados que podem ser atingidos a partir do Estado Inicial por aplicação dos operadores (i.e., através de qualquer caminho possível).

A resolução de um problema pode ser vista como uma **Pesquisa num Espaço de Estados**.

### Avaliação da Pesquisa

Uma pesquisa pode ser avaliada segundo 3 critérios:

- Alguma **solução foi encontrada?**
- O **Custo do Caminho/Solução** é baixo?
- Qual o **Custo da Pesquisa** em termos de tempo e memória necessários?

- **Custo Total = Custo do Caminho + Custo da Pesquisa**
- Normalmente, um custo do caminho mais baixo implica custos de pesquisa mais altos

# CAP. 4 - PESQUISA NÃO INFORMADA

## Estratégia de Pesquisa

Uma pesquisa constrói uma **Árvore de Pesquisa** cuja raiz é o Estado Inicial.

### Estratégias:

#### *Critérios de Classificação:*

- **Completude:** A estratégia garante que se encontra uma solução quando existe de facto uma?
- **Optimização:** Encontra a melhor solução quando há várias possíveis?
- **Complexidade Temporal:** Quanto tempo leva a encontrar uma solução?
- **Complexidade Espacial:** Quais os requerimentos de memória?

### Dois tipos de Pesquisa

#### *Pesquisa Cega ou Não Informada*

- Procura uma solução sem recorrer a qualquer informação adicional que a guie.

- Apenas pode comparar o estado actual com o objectivo
- As variantes deste tipo de pesquisa variam de acordo com a ordem definida para expansão de estados

*Pesquisa Heurística ou Informada*

- Procura uma solução recorrendo a informação adicional que permite escolher o nó a expandir primeiro

# Algoritmo Geral de Pesquisa –AGP

# CAP. 4 - Pesquisa não informada

## Pesquisa em Largura (Breadth-First)

Primeiro expande-se o nó N e só depois o N+1

A partir da raiz, a árvore é expandida por níveis

**Vantagens:**

- Completa

Porque procura todas as soluções possíveis e portanto encontrará uma caso exista

- Óptima

Desde que o Custo do Caminho seja uma Função Não-Decrescente da profundidade dos nós.

  - A Pesquisa em Largura propõe sempre como solução a que tiver menor número de nós. Portanto, se o custo aumentar com a profundidade, as soluções com menos nós representam menor custo.

**Desvantagens:**

- O custo da pesquisa é muito elevado
  - Complexidade temporal exponencial
  - Complexidade espacial exponencial

- Análise da Complexidade da Pesquisa em largura
    - Exponencial

Problemas de pesquisa cujos algoritmos têm complexidade exponencial, **apenas podem ser resolvidos para instâncias de pequena dimensão**.

## Pesquisa em Profundidade

Cada nó é expandido até ser atingido o último nível da árvore, a menos que uma solução seja encontrada entretanto:

**Características:**

- Incompleta
    - No caso de a profundidade da árvore ser infinita (neste caso tentará atingir o último nível - que nunca alcança - e não retorna nenhuma solução.

- Não Óptima
    - Retorna uma solução qualquer e nenhuma condição pode garantir que seja a melhor.

## Pesquisa Uniforme (f = g)

*"Olha para o passado".*

g = custo acumulado desde a origem até ao estado actual (custo do caminho já percorrido).

**Variante da Pesquisa em Largura**

- Consiste em expandir primeiro os nós que têm um custo associado menor, que estão mais perto da origem.

    - Esta expansão pára quando for encontrada uma solução e o custo acumulado dos caminhos associados aos nós que falta expandir já for superior à solução encontrada.

- Garante a solução óptima, bastando apenas que o custo aumente com a profundidade

    - Comparativamente com a pesquisa em largura, resolve a limitação da solução óptima só ser garantida para custos que aumentam uniformemente com a profundidade de todos os caminhos.

**Características:**

- Completa
- Óptima

    - Desde que o custo aumente com a profundidade: g(Sucessor(n)) >= g(n)
    - Se admitirmos que o custo possa diminuir com a profundidade, então seria preciso explorar a árvore toda para determinar qual o caminho óptimo!

### Pesquisa Profundidade Limitada

Resolve a limitação da Pesquisa em Profundidade de não retornar resultados em espaços de profundidade muito grande, impondo um limite, m, à profundidade máxima a atingir.

Por exemplo, se um mapa contem 20 cidades, o caminho entre quaisquer duas tem de ser composto, no máximo, por 19. Logo, m=19.

### Pesquisa por Aprofundamento Progressivo (IDS -Iterative Deepening Search)

Combina as Pesquisas em Largura e em Profundidade

Evita a necessidade de se definir "m"antecipadamente:

- Em vez de se estabelecer um só limite geral, começa por se estabelecer um limite inicial de profundidade = 0

- Este limite vai-se alargando (1,2, 3,...m) para as iterações seguintes (i.e. faz-se uma pesquisa em profundidade de nível 1, depois 2, depois 3... mas para cada pesquisa reinicia-se o algoritmo da pesquisa em profundidade, desde a raíz).

## CAP. 5 - Pesquisa informada

- Pesquisa Sôfrega
- Pesquisa A*
- Variantes do A*

Os métodos de **Pesquisa Não Informada** são muito ineficientes:

- Exigem Grandes requisitos de tempo
- Exigem Grandes requisitos de memória
- As Soluções encontradas nem sempre são óptimas

## Pesquisa Sôfrega (f = h)

Expande-se em primeiro lugar o nó que parece estar mais perto do objectivo, tendo em conta o custo.

**"Olhar para o futuro usando heurística".**

**Características:**

- Não óptima
  - não garante que se encontre o caminho de menor custo
- Incompleta
  - pode seguir caminhos infinitos

## Pesquisa A* (f = g + h)

Combina Pesquisa Uniforme com Pesquisa Sôfrega

Estes dois tipos de pesquisa são complementares:

- A Uniforme "mede" a parte inicial do percurso - g(n)
- A Sôfrega "mede" a **aparente** parte restante - h(n)

No AGP, a inserção na lista de Nós a Expandir é feita por ordem crescente de f(n).

- $f(n)=g(n)+h(n)$

A pesquisa A* é **óptima e completa** desde que:

- A heurística utilizada nunca sobrestime o custo do caminho do nó n até ao objectivo (isto é, **nunca possa assumir um valor superior ao do custo real**).
    - Se assim for, constitui uma **Heurística Admissível**

### Heurística Admissível

Se for inferior ou igual ao custo real.

A heurística a escolher nunca pode sobrestimar o custo do caminho até à solução.

A função heurística cujo valor se aproxima mais do custo da solução real, é habitualmente a melhor

## Variantes A*

### Variantes do A* com limitação de memória

#### *IDA**

- A* com aprofundamento progressivo
- "IDS para A*"

#### *SMA**

- "Simplified Memory Bounded A*"
- Desenhado para não ultrapassar o limite de memória disponível para resolver um problema.

### IDA*

Está para a a pesquisa A*, assim como o IDS está para a pesquisa em profundidade:

- No IDS cada iteração é limitada por um nível de profundidade crescente
- No IDA* cada iteração é limitada por um valor crescente da função de custo, f(n)=g(n)+h(n)
- Para cada "limite de custo estimado", fi ,"exclui" os nós cujo valor f é superior
- Pára quando atingir um nó objectivo cujo f é <= que o limite actual
- Enquanto não encontrar um objectivo nestas condições, progride para o limite seguinte, fi+1

### Características da Pesquisa IDA*:

- É completa
- É óptima
- Por ser baseada na pesquisa em profundidade, o requerimento de memória é baixo e pode ser **aproximado por b.d**(b=branching factor, d=profundidade da solução)

### SMA*

Tenta utilizar apenas a memória disponível para resolver um problema.

- É completo e óptimo desde que a memória possibilite a sua execução completa
- Se a memória estiver toda utilizada devido às expansões efectuadas, "esquece" os nós menos promissores (os de valor de f mais elevado), usando o espaço assim libertado para o resultado de outras expansões
- O nó a expandir é o de menor valor de f
- Porém, **quando se expande esse nó, adiciona-se-lhe apenas um sucessor em cada iteração**
- Quando um nó se encontrar completamente expandido, o seu custo estimado, f, é actualizado com o mínimo dos valores de f dos seus nós filhos da iteração

## CAP. 6 - PESQUISA LOCAL – MELHORAMENTO ITERATIVO

## Problema com Restrições (CSP-Constraint Satisfaction Problem)

Trata-se de um problema cuja solução só é válida se satisfazer certas condições.

Um CSP é caracterizado por:

- **Variáveis**: Os seus valores finais representam a solução
- **Domínio**: Conjunto de valores que as variáveis podem assumir
- **Restrições**: atuam sobre as variáveis
- **Problema**: Assinalar valores às variáveis sem violar as restrições

**Exemplo**: problema das 8 rainhas.

Num CSP interessa determinar apenas um estado final válido e não um caminho que leve a esse estado (não interessa o caminho, apenas o estado final).

- O **Estado Final é desconhecido** e constitui a solução do problema.

Um CSP pode ser resolvido por técnicas de pesquisa, contudo são geralmente ineficientes neste contexto, dado gerarem muitos estados desnecessariamente.

Existem algoritmos especialmente adaptados à resolução de CSPs:

- Hill-Climbing (**mais usado**)
- Simulated-Annealing
- Pesquisa Tabu

### *Heurística dos "Conflitos Mínimos"*

Tentar garantir o mínimo conflito com as restrições.

### Os algoritmos de melhoramento iterativo caracterizam-se por:

- Não anotam os estados intermédios que conduzem a uma solução, apresentando apenas a "configuração" válida que a compõe

- Partem de uma configuração inicial completa (que viola as restrições), eventualmente gerada aleatoriamente, e melhoram-na sucessivamente até alcançarem uma solução

## Trepa-Colinas

Também conhecido por "Hill-Climbing" ou "Gradient-Descent"

O nome e ideia base provêm de uma analogia com a decisão tomada por um agente que, perdido numa encosta, pretende atingir o topo:

- Deslocar-se-á na direcção "em que o caminho sobe".

**Espaço de procura** é um espaço de soluções, ao contrário dos outros algoritmos que vimos antes (A*, etc…), em que a pesquisa era feita num espaço de estados.

### Implementação

- Parte de um estado inicial dado ou gerado aleatoriamente
    - todas as variáveis com valores atribuídos
- Gera os estados sucessores do estado actual
- Através de uma Função de Avaliação, avalia cada estado assim gerado e escolhe o de maior valor
- Pára quando o estado seleccionado tiver um valor inferior ao escolhido na iteração anterior
    - isto significa que a solução "piorou" e que se "está a descer a colina, em vez de a subir"

### Problemas

- Um **máximo local** pode ser atingido sem que corresponda ao máximo absoluto (melhor solução)
- Nos "planaltos" é necessário **escolher uma direcção** aleatoriamente
- Um cume pode ter lados tão inclinados que o passo seguinte conduz ao "outro lado do cume" e não ao seu topo. Neste caso a **solução poderá "oscilar"** nunca atingido o máximo pretendido

### Tentativa de resolução dos problemas relativo ao atingir um ponto de não progresso:

- Reiniciar a pesquisa partindo de um estado inicial diferente, gerado aleatoriamente (Random-Restart-Hill-Climbing)

- Guarda o melhor resultado obtido nas pesquisas anteriores
  - até ao ponto de não-progresso

- Pára quando atingir o número de reinícios máximo ou quando o melhor resultado guardado não for ultrapassado durante "n" iterações (valor "n" é pré-fixado)

## Recristalização Simulada (*simulated annealing*)

**Temperar materiais** – aquecer os metais para depois os arrefecer bruscamente e assim ficarem mais resistente.

**"Andar para trás para depois avançar para uma solução melhor (Têmpera)"**

Usa a seguinte estratégia para ultrapassar máximos locais:

- Quando encontra um máximo (pode ser apenas um máximo local!), o algoritmo prossegue "durante algum tempo" a pesquisa no sentido descendente.

- Em vez de se escolher sempre o estado seguinte de maior valor, escolhe-se um, aleatoriamente

- Se a sua avaliação:
  - For superior à do estado anterior, é sempre escolhido

  - For inferior, é escolhido mas apenas com uma certa probabilidade (<1) que baixa à medida que um parâmetro 'T' tende para zero ao longo das sucessivas iterações.

- Quando T for muito pequeno, a escolha de estados de pior avaliação quase nunca ocorre, e o "Simulated Annealing" comporta-se (quase) como o "Hill-Climbing".

  - Com a continuação, as descidas vão sendo cada vez menos permitidas, de modo que no final apenas a subida para um máximo (que parte já de um valor elevado) é permitida: Com maior probabilidade, este poderá ser o máximo absoluto

- O nome do algoritmo (Têmpera Simulada) provém de realizar uma analogia com o processo de têmpera de certos metais ou arrefecimento de um líquido até que congele.

  - O parâmetro T simula a temperatura, que baixa com o tempo

- Algoritmo **probabilístico**
  - O resultado é não determinista.
  - Deve-se executar o algoritmo mais do que uma vez.

- Se o arrefecimento for "suficientemente" lento é sempre atingido o **ótimo global!**

**ΔE = Valor (estado seguinte) - Valor (estado actual) -> MAXIMIZAÇÃO**

**ΔE = Valor (estado actual) - Valor (estado seguinte) -> MINIMIZAÇÃO**

## Pesquisa Tabu

**Princípio de funcionamento:**

- Durante a pesquisa, **forçar a exploração de novas zonas do espaço de procura.**
  - Pode assim evitar-se entrar em ciclos

**Implementação:**

- Recurso a uma memória de curta-duração
  - Indica quais os movimentos **proibidos** (movimentos TABU)

**Características:**

- **Vantagens**
  - Escolhe sempre o melhor vizinho, desde que seja válido, exibindo assim um comportamento determinista
  - Ao aceitar soluções de pior qualidade, pode evitar óptimos locais

- **Desvantagens**
  - Nem sempre é fácil ajustar o limite de memória e número máximo de iterações

# CAP.7 - ALGORITMOS GENÉTICOS (PESQUISA GLOBAL)

## Introdução

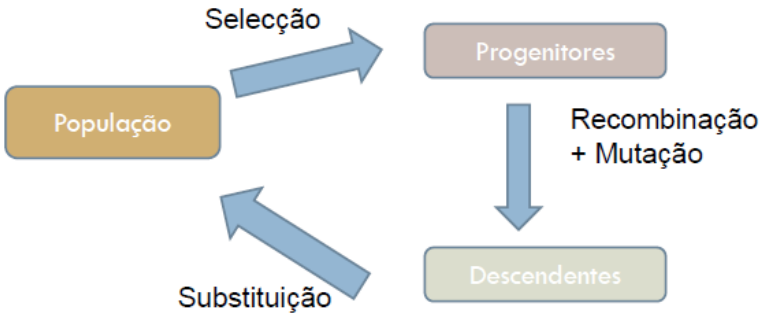
- **Computação Evolucionária (CA)**
  - A área de investigação designada por "Computação evolucionária (EC) " envolve:
    - Algoritmos Genéticos (GA)
    - Estratégias Evolucionárias (ES)
    - Programação Evolucionária (EP)

## Algoritmos Genéticos

- Técnica para resolução de problemas que necessitem de optimização
- Baseados na Teoria de Evolução de Darwin

**Sub-classe da computação evolucionária.**

## Princípio Básico de Funcionamento




**Selecção**

As "melhores hipóteses" são as de maior "aptidão". Esta aptidão é avaliada por uma função (*fitness function*).

**Recombinação (crossover) e Mutação:**

Em vez de procurarem sistematicamente uma solução (hipótese h), os AGs geram hipóteses sucessoras das actuais (descendência/offspring) recombinando **propabilisticamente**as melhores hipóteses entre si, e "mutando" algumas outras.

**Função de Fitness (aptidão)**

Define o critério que avalia cada hipótese de acordo com o objectivo a atingir.

- É a base de qualquer critério de selecção na geração seguinte.

**Exemplos:**

- Mochila: Lucro da solução
- N-Rainhas: A função de fitness calcula a soma dos ataques que as rainhas produzem, em cada configuração.

## PESQUISA LOCAL (a cada iteracção temos apenas 1 solução)

Trepa-Colinas

Tabu

Recristalização Simulada

## PESQUISA GLOBAL (a cada iteracção temos muitas soluções – População)

Algoritmo Genético

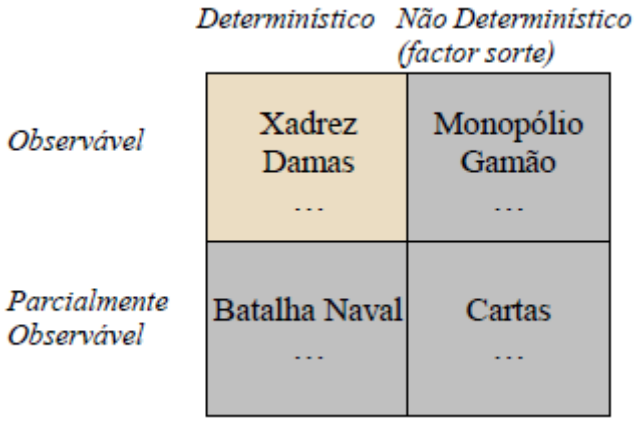
# CAP. 8 - ALGORITMOS PARA JOGOS

Os jogos diferenciam-se pela inclusão de um factor de incerteza devido à presença de um adversário

**Incerteza do tipo não probabilística:**

- O adversário (B) tentará a melhor jogada para ele, o que implica a pior jogada para o oponente (A).
- A aplicação de algoritmos de pesquisa para encontrar a melhor solução para A não funciona! Pois é necessário contar com os movimentos de B.

Tipos de jogos




O Minimax aplica-se a jogos **determinísticos** e **observáveis**

## Algoritmo Minimax

Jogo com dois indivíduos, designados por MAX e MIN

- Jogam alternadamente - MAX joga primeiro
- No final do jogo pode acontecer:
    - MAX ganha (MIN perde)
    - MAX perde
    - Empatam
- Baseia-se no princípio de "selecção da melhor jogada por parte de cada jogador"
- **A árvore é toda construída inicialmente**.
    - Toda a árvore é percorrida
    - Travessia em profundidade

- **Algoritmo recursivo**
    - Atribuição de valores é feita dos nós terminais para a raiz

- **Impraticável para jogos complexos**
    - Memória

## Alpha-Beta Pruning

- Recursos: Tempo e Memória
    - O algoritmo Minimax necessita de memória e tempo consideráveis, mesmo para jogos relativamente simples!

**O algoritmo baseia-se na utilização de dois parâmetros, "Alpha" e "Beta":**

- **Alpha**: representa o valor mínimo garantido que MAX poderá obter.
    - Como representa um limite inferior é inicializado a -∞e vai crescendo, sendo actualizado num nó MAX.

- **Beta**: representa o valor máximo que MIN consegue impor a MAX
    - MAX nunca conseguirá jogar para obter um valor superior a beta
    - Sendo um limite superior, é inicializado a +∞e posteriormente vai decrescendo (actualizado num nó MIN)

- Ao atingir-se um nó em que alfa >= beta, pode cortar-se o ramo

**Características:**

- **Algoritmo óptimo**
    - A estratégia sugerida é igual à que teria sugerida pelo MiniMax (sem pruning)
- A Eficácia do algoritmos depende da ordem pela qual os sucessores são avaliados

**PERGUNTA DE EXAME:**

**P:** Aplicando o Alpha-Beta Pruning, o MAX sai beneficiado ou prejudicado, relativamente ao MINIMAX?

**R:** Nem uma coisa nem outra, porque o Alpha-Beta Prunning não constrói a árvore toda. Corta só onde sabe que pode cortar.

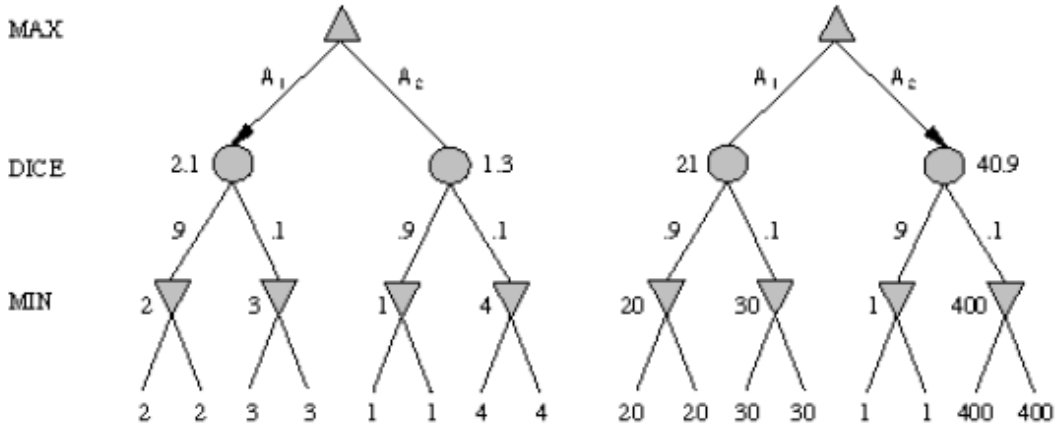
A decisão entre um e o outro TEM de ser exactamente a mesma.

## Jogos com Elemento Sorte

- Será possível aplicar o algoritmo Minimax também nestes jogos?
    - Sim, contudo a **árvore deverá incluir também nós que traduzam o fator sorte.**

Os nós "sorte" alternam com "Max" e "Min":

- **MAX**
- **SORTE (PROBABILIDADE)**
    - Representam-se os valores de probabilidade de ocorrência
- **MIN**




**Cálculo dos Nós Sorte:**

- 2.1 = .9 x 2 + .1 x 3
- 1.3 = .9 x 1 + .1 x 4

## CAP. 9 - Aprendizagem com Redes Neuronais

Imitam o cérebro humano (neurónios).

### Rede Neuronal Artificial

- Consiste num elevado número de interconexões entre unidades de processamento elementares (neurónios, com uma funcionalidade análoga no neurónios biológicos).
- O **conhecimento** é armazenado através dos valores dos pesos, obtidos através de um processo de adaptação ou aprendizagem a partir de um conjunto de dados de treino
- O ajusto dos pesos -**Aprendizagem**é realizada de forma automática.
- Caracteriza-se por um processamento distribuído**.**

## Aprendizagem:

- Processo pelo qual os parâmetros de uma rede neuronal são adaptados através de um processo de treino baseado em dados experimentais

- O tipo de aprendizagem determina a forma de adaptação dos parâmetros.