

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

INTRODUÇÃO E VISÃO GERAL

Carlos Pereira
ISEC 20-21

Índice

2

- Definições de Inteligência Artificial
- Teste de Turing
- Breve Perspectiva História da IA
- Aplicações

O que é a Inteligência Artificial?

3

- Objectivos Gerais da IA
 - ▣ Compreender as características do comportamento inteligente (humanos e máquinas)
 - ▣ Desenvolver máquinas que consigam executar tarefas complexas tão bem ou melhor que um humano.

O que é a Inteligência Artificial?

4

- ...
 - ▣ A Inteligência Artificial (IA) foca o **comportamento** inteligente em entidades artificiais, que envolve:
 - Percepção
 - Raciocínio
 - Adaptação
 - Comunicação
 - ▣ A IA é inter-disciplinar.
 - As suas raízes e posterior desenvolvimento baseiam-se em Filosofia, Matemática, Psicologia, Informática, Linguística

O que é a Inteligência Artificial?

5

□ Definições de IA

▢ “É a arte de criar máquinas que executem funções que necessitam de inteligência quando executadas por humanos.”

■ (R. Kurzweil)

▢ “É a compreensão dos mecanismos envolvidos no pensamento e comportamento inteligente e a sua transposição para sistemas artificiais.”

■ (Definição da AAAI)

O que é a Inteligência Artificial?

6

□ Definições de Inteligência

▢ “... global capacity of the individual to act purposefully, to think rationally, and to deal effectively with his environment.”

■ David Wechsler,

▢ “... a very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience.”

■ (Definição conjunta de investigadores da área, 1994)

O que é a Inteligência Artificial?

7

Quais as características que uma máquina deve exibir para que seja considerada inteligente?

O que é a Inteligência Artificial?

8

- Um sistema com comportamento realmente inteligente deve ter a capacidade para:
 - Planear as suas acções
 - Raciocinar
 - Agir de forma adequada em ambientes complexos
 - Agir de forma autónoma
 - Aprender
 - Comunicar com outras entidades
 - ...

O que é a Inteligência Artificial?

9

- Existem duas abordagens para se atingir esse comportamento “inteligente”?
 - ▣ IA forte
 - Criar máquinas realmente inteligentes, com uma mente equivalente à dos humanos.
 - Será possível?!
 - ▣ IA fraca
 - Criar máquinas que imitem o comportamento inteligente dos humanos

O que é a Inteligência Artificial?

10

- Inteligência Artificial Fraca
 - ▣ Desenvolvimento de máquinas que aparentam possuir um comportamento inteligente.
 - ▣ Duas possibilidades:
 - Sistemas que agem como um humano
 - Sistemas que agem de forma racional

Sistemas que agem como um humano

11

- Sistemas que agem como um humano:
 - Externamente, o comportamento da máquina deve ser semelhante ao de um ser humano
 - Como avaliar se uma máquina obedece a este princípio?
 - Teste de Turing (Alan Turing, *Computing Machinery and Intelligence*, 1950).

...Teste de Turing

12

- O objectivo é determinar se um programa de computador é ou não inteligente



- A partir de um teclado, um inquiridor humano conduz duas conversas com dois interlocutores:
 - :
 - Um interlocutor humano
 - O computador

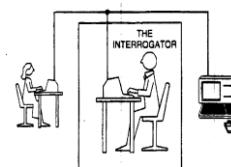


Figure 1.1 The Turing test.

- O programa é inteligente se o humano não conseguir descobrir se foi o humano ou o computador a responder às perguntas!

...Teste de Turing

13

- IA atual

- Siri
 - <https://www.apple.com/siri/>
- Alexa
 - <https://www.amazon.com/b?node=17934671011>
- Google assistant
 - <https://assistant.google.com/>
- Robotics
 - <https://www.bostondynamics.com/>



...Teste de Turing

14

- Para passar o teste de Turing, o sistema deve possuir:
 - Representação de conhecimento
 - Mecanismos de raciocínio
 - Capacidade de adaptação a novas circunstâncias
 - Capacidade de extrapolação
 - Processamento de linguagem natural (NLP)

...Teste de Turing

15

□ Teste de Turing ao contrário

- ▣ O computador solicita ao utilizador que realize um teste!
 - CAPTCHA (Completely Automated Public Turing Test to tell Computers and Humans Apart) – Teste de Turing público, completamente automatizado, para diferenciação entre computadores e humanos

- Exemplo:

Verificação de palavras: Escreva os caracteres que vê na imagem abaixo.



Não há distinção entre letras maiúsculas e minúsculas

Termos de utilização:

Verifique as informações da Conta Google que inseriu acima (pode alterar o que desejar) e leia os Termos de utilização abaixo.

Sistemas que agem como um humano

16

□ Humanoides

- Sophia

- <https://www.hansonrobotics.com/sophia/>

- “symbolic AI, neural networks, expert systems, machine perception, conversational natural language processing, adaptive motor control and cognitive architecture among others”



Sistemas que agem como um humano

17

□ Máquinas Sociais

■ Interacção com humanos de uma forma natural e expressiva.

■ Aprendizagem por imitação, observação

■ Exemplos

■ <https://robots.ieee.org/robots/aibo2018/>

■ <https://robots.ieee.org/robots/paro/>



Sistemas Racionais

18

□ Na IA fraca existe uma segunda alternativa à imitação do comportamento humano: Agir Racionalmente!

■ Na abordagem racional:

■ O sistema é inteligente se possuir um comportamento racional

■ Com base em **Conhecimento** e num conjunto de **Crenças**, a máquina actua de forma a atingir os seus objectivos

Sistemas Racionais

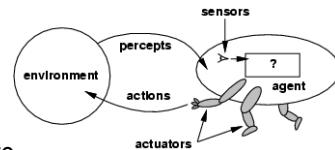
19

□ Agentes Inteligentes

- Entidades que agem forma racional.

■ Funcionamento

- Percepcionam o meio ambiente
- Agem de acordo com os seus **objectivos**
- Possuem **conhecimento** que os ajuda a escolher as ações mais apropriadas
- Nesta abordagem, a IA define-se como a disciplina que estuda e desenvolve agentes que se comportam de forma racional



Sistemas Racionais

20

□ ...

▫ Exemplos:

- Evolved Virtual Creatures (Karl Sims): desenvolvimento de criaturas virtuais que tenham a capacidade de se deslocar em ambientes físicos 3D
 - http://www.youtube.com/watch?v=JBgG_VSP7f8

■ Inteligência de Enxame

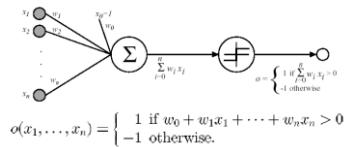
- “interactions between agents lead to the **emergence** of intelligent global behavior, unknown to the individual agents”
 - <http://www.swarmintelligence.org/>

Perspectiva Histórica

21

□ Principais Marcos do passado:

- McCulloch and Pitts (1943), Redes neurais
 - Propõem um modelo artificial de neurónios



■ Donald Hebb, 1949,

- Demonstra que uma rede neuronal pode **aprender** através de um algoritmo simples de alteração dos pesos das ligações entre neurónios,

Perspectiva Histórica

22

□ ...

- Claude Shannon, 1950 e Allan Turing, 1953:
Primeiros programas para jogar xadrez

- O HITECH prevê 10 milhões de jogadas antes de decidir um movimento. Foi o primeiro a vencer um mestre de xadrez.
- O Deep Thought 2 foi implementado pela IBM e Carnegie Mellon University (CMU).
- O Deep Blue (IBM) gera cerca de 100 a 200 biliões de jogadas por movimento. Em 1996 perdeu contra Kasparov. Em 1997 venceu-o.

Perspectiva Histórica

23

- ...
 - “Computing Machinery and Intelligence”, Alan Turing, 1950;
 - <http://www.abelard.org/turpap/turpap.php>
 - Minsky and Edmonds (1951), Computador com redes neuronais
 - Designação “Artificial Intelligence”, J. McCarthy, 1956

Perspectiva Histórica

24

- ...
 - Designação “Artificial Intelligence”,
 - Em 1956, John McCarty, Minsky, Shannon e outros reúnem-se numa workshop em Dartmouth
 - John McCarthy propõe o nome Artificial Intelligence para a nova área de investigação.



Claude Shannon



Marvin Minsky



John McCarty

Perspectiva Histórica

25

- ...

- Sistemas Generalistas:

- São considerados os primeiros Sistemas Inteligentes. A sua concepção é orientada para a resolução de um problema qualquer (General Problem Solving - GPS)

- Sistemas Baseados em Conhecimento (1970...)

- Baseados no conhecimento de um domínio de aplicação
 - O conhecimento pode ser extraído de peritos do domínio, registos informáticos, documentação...
 - Pode ser representado por regras (lógica, difusas, ...) casos (contextos, situações, ocorrências, ...) ou modelos (redes neurais, ...)

Desafios

26

- Tópicos

- Transparência
 - Privacidade
 - Ética
 - Algorithm bias

- <https://deepmind.com/>

Desafios

27

□ Google deepmind



■ <https://deepmind.com/research/case-studies/alphago-the-story-so-far>

■ [Https://www.youtube.com/watch?v=TnUYcTuZJpM](https://www.youtube.com/watch?v=TnUYcTuZJpM)

□ AlphaZero

■ Consiste numa versão genérica do “AlphaGo Zero” que permite jogar também xadrez e Shogi, além de Go.



INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

20-21

CAP. 2 AGENTES RACIONAIS

Carlos Pereira
ISEC

Índice

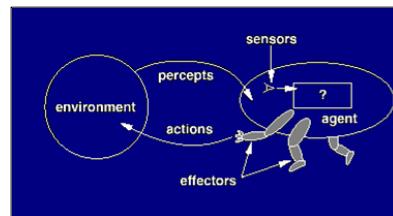
2

- Índice
 - Conceito de agente
 - Agente Racional
 - Estrutura Interna de um Agente
 - Tipos de Agentes
 - O Ambiente
 - Agentes Aprendizes

Conceito de Agente

3

- Um **Agente** é uma entidade que habita um denominado ambiente e é capaz de:
 - Percepcionar
 - Recebe informação do ambiente que a rodeia através de **sensores**
 - Agir
 - Actuar sobre o ambiente através de **actuadores**



Conceito de Agente

4

- ...
- Exemplos de Agentes
 - Um ser humano
 - Possui sensores (olhos, ouvidos, ...) e actuadores (braços, pernas, ...)
 - Um robot
 - Possui sensores (câmaras, sensores de infra-vermelhos, de pressão, ...) e actuadores (motores, braços mecânicos, ...)
 - Sistema de pesquisas autónomas na internet
 - Filtro de correio electrónico
 - Nas aplicações de software, a informação sobre o ambiente e acções, são representadas por informação (estruturas de dados) que o agente manipula.

Conceito de Agente

5

- ...
 - Jogos
 - Shopbots
 - comparar preços na internet
 - Assistentes Virtuais
 - Chatterbots, seres virtuais, ...
 - E muitos outros...
 - www.agentland.com
 - www.trsoccerbots.org
 - <http://ccl.northwestern.edu/netlogo>



Conceito de Agente

6

- Virtual Assistants
 - http://www.chatbots.org/virtual_assistant/
- Ask Anna from IKEA
 - "Created in 2003 by [Artificial Solutions](#), she remains one of the largest implementations of a Virtual Assistant worldwide. Anna resides in 20 countries, being able to communicate in 18 languages via all IKEA's country websites"
 - http://www.ikea.com/ms/en_GB/customer_service/contact_us/contact.html
 - <http://www.buscas.pt/index.html>

Conceito de Agente

7

□ ...

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

Agente Racional

8

□ Agente Racional

- Um Agente que escolhe a acção correcta, isto é, aquela acção que leva o agente a atingir o maior sucesso

□ É assim necessário avaliar o sucesso (Como e Quando?)

- Como? Não existe uma única função de avaliação.
 - Por exemplo, para um “agente de limpeza”, pode avaliar-se: A qualidade da limpeza, a electricidade consumida, o ruído gerado e/ou o tempo dispendido.

Agente Racional

9



- ...
- Quando?

- Poderemos considerar de melhor performance os agentes que agirem mais rapidamente.

- Exemplo: O Robot Aspirador

- Ambiente Área dividida em células
- Objectivo: aspirar todo o lixo minimizando o consumo de energia
- Percepções: Conteúdos da célula em que se encontra e célula em frente
- Acções: Mover-se para a frente, virar à esquerda, virar à direita, aspirar

Agente Racional

10



- ...
- O nível de racionalidade depende de quatro factores:

- **Conhecimento inicial** (à partida) do ambiente
- **Sequência de percepção** (tudo aquilo que já apercebeu)
- **Acções** que pode tomar
- Função usada para **avaliação** do sucesso



- O Agente Racional Ideal

- Para uma dada sequência de percepção, **toma a acção que espera vir a maximizar a função de medida do seu sucesso**, baseando-se na sequência de percepção e conhecimento inicial do ambiente

Agente Racional

11



■ Mapeamento Percepção-Ação

- O projecto de um Agente Racional Ideal consiste em mapear sequências de percepção em ações óptimas.

□ Agentes Autónomos

■ Se o comportamento do agente for determinado pela sua experiência, diz-se que o agente é autónomo.

- Um agente que baseie o seu comportamento no conhecimento prévio do ambiente (conhecimento embutido) pode ser considerado autónomo?

Agente Racional

12



■ Outras características de uma agente autónomo

- Deve conseguir **operar em ambientes diversos**, dado o tempo necessário para se adaptar.
- Um agente autónomo deve possuir capacidade de **aprendizagem**.
- Contudo, algum conhecimento embutido é ainda necessário, para que o agente não tenha de agir aleatoriamente no início (tal como acontece com os reflexos inatos no mundo animal...)

Estrutura Interna de um Agente

13

- Agente = Arquitectura + Programa
 - ▣ No projecto de agentes racionais, na IA pretendemos implementar funções ou programas que mapeiam percepções em acções
 - ▣ Os programas executam num contexto (hardware, plataforma de programação) designado por arquitectura
 - A arquitectura inclui as componentes físicas do agente: um computador e eventualmente câmaras, sensores diversos, filtros, dispositivos electro-mecânicos, etc.

Estrutura Interna de um Agente

14

- Programa de um agente racional: Tarefa principal da IA
 - ▣ O Agente como forma genérica de um Programa
 - ▣ O Agente necessita de registar as sequências de percepções, dado receber uma percepção de cada vez.
 - Recurso a estruturas de dados. Estas estruturas devem ser:
 - Actualizadas pelas novas percepções
 - Manipuladas pelo agente através dos seus procedimentos de tomada de decisão, com vista à escolha de uma acção

Estrutura Interna de um Agente

15

□ ...

■ Esqueleto de um programa

Modelo do mundo

```
Function Skeleton-Agent(percept) returns action
    static: memory /* A memória que o agente tem do ambiente */
    memory ← Update-Memory (memory, percept)
    action ← Choose-Best-Action (memory)
    memory ← Update-Memory (memory, percept)
    return action
```

Estrutura Interna de um Agente

16

□ ...

■ Exemplos para o robot aspirador

■ Exemplo A:

```
If Célula_Actual(Lixo) Then Aspirar
If Célula_Frente(Fronteira) Then
    Rodar_Esquerda
If Célula_Frente(Lixo) Then Avançar
If True Then Rodar_Direita
```



Estrutura Interna de um Agente

17

- ...
- Exemplo B

```
If Célula_Frente(Fronteira) Then Rodar_Direita  
If Célula_Frente(Vazio) Then Avançar  
If True Then Aspirar
```

- Que problemas identifica nos agentes descritos?
- Qual seria um bom agente?

Tipos de Agentes

18

- Podemos considerar quatro tipos de Agentes
 - Agentes Reactivos
 - Agentes Reactivos com Estado Interno
 - Agentes Guiados por Objectivos
 - Agentes Baseados em Funções de Utilidade

Tipos de Agentes

19

□ Agentes Reactivos

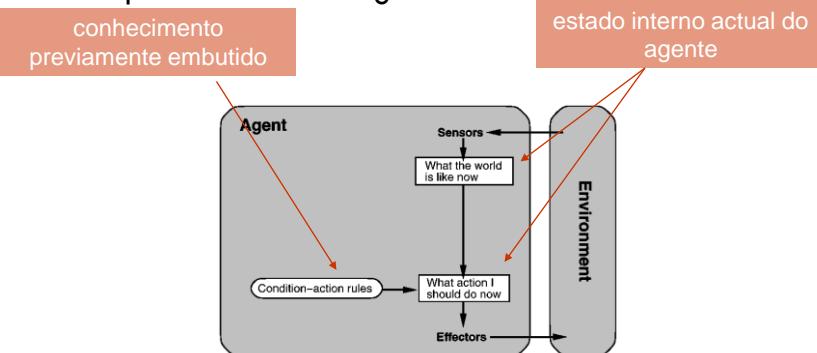
- Respondem a cada percepção **sempre da mesma forma**, tomando em linha de conta apenas a percepção mais recente.
- Funcionam como um simples reflexo, traduzível por uma regra do tipo “if...then...”
- Simulam reflexos adquiridos ou inatos.

Tipos de Agentes

20

□ ...

■ Arquitectura de um agente reactivo



Tipos de Agentes

21

- ...
- Possui um interpretador de regras
- Aplicabilidade reduzida

```
Function Simple-Reflex-Agent(percept) returns action
    static: rules /* Um conjunto de regras percepção-acção */
    state   ← Interpret-Input (percept)
    rule    ← Rule-Match (state, rules)
    action  ← Rule-Action (rule)
    return action
```

Interpret-Input cria (em *state*) uma representação abstracta da percepção actual

Rule-Match identifica a regra cujo antecedente é mais semelhante ao estado actual, *state*

Tipos de Agentes

22

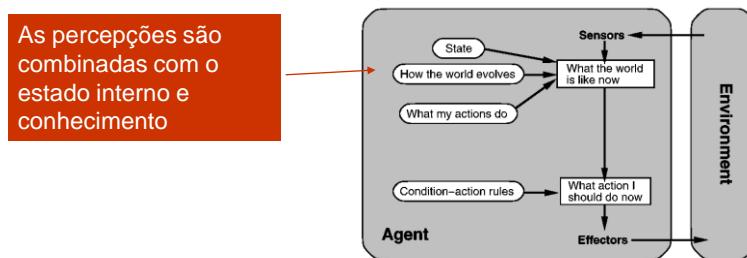
- Agentes Reactivos com Estado Interno (memória)
 - Respondem à mesma percepção de forma eventualmente diferente, **combinando a percepção mais recente com informação acerca do estado anterior do ambiente.**
 - A sua actualização requer conhecimento sobre:
 - Como se modifica o mundo ao longo do tempo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

Tipos de Agentes

23

□ ...

□ Arquitectura



Tipos de Agentes

24

□ ...

□ Programa

```
Function Reflex-Agent-with-Internal-State(percept) returns action
    static: state      /* Uma descrição do estado do ambiente */
              rules     /* Um conjunto de regras percepção-ação */
    state   ← Update-State (state, percept)
    rule    ← Rule-Match (state, rules)
    action  ← Rule-Action (rule)
    state   ← Update-State (state, action)
return action
```

Tipos de Agentes

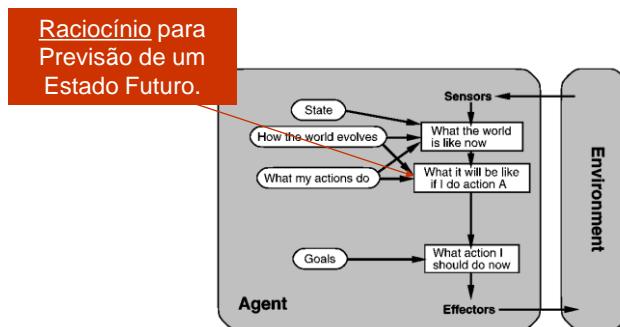
25

- Agentes Guiados por Objectivos
 - É também importante **considerar os objectivos** a atingir
 - Este agentes 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 (nos modelos anteriores, isto implicaria a escrita de novas regras if...then...).

Tipos de Agentes

26

- ...
- Arquitectura



Tipos de Agentes

27

- ...

função agente_guia do por_objectivos(percepção): acção

estado: modelo do estado actual do ambiente

descritor_de_acções: descreve o efeito das acções no estado do mundo

objectivo: estado que o agente deseja atingir

estado \leftarrow Actualiza_estado (estado, percepção)

acção \leftarrow Avaliador (estado, descritor_de_acções, objectivo)

estado \leftarrow Actualiza_estado (estado, acção)

fim_de_função

Tipos de Agentes

28

□ 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

- Pode ponderar factores contraditórios (distância/estado da estrada)

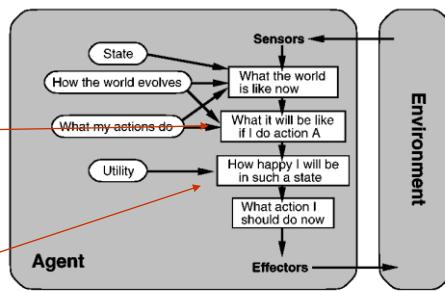
- Permite medir o grau de sucesso obtido quando um objectivo é atingido

Tipos de Agentes

29

□ ...

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



- Uma Função Utilidade mapeia um estado ou conjunto de estados num número real, que mede o grau de sucesso obtido pelo agente se optar por determinada acção.

O Ambiente

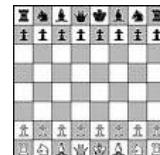
30

- 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

O Ambiente

31

- 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:
 - Jogo de Xadrez
 - O agente consegue obter toda a informação relevante para a tomada de decisão?
 - Jogo de Poker
 - Acessível ou não acessível?



O Ambiente

32

- 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. Caso contrário diz-se estocástico.
 - Exemplos
 - Jogo de xadrez. Determinístico ou estocástico?
 - Condutor autónomo de um veículo. Determinístico ou estocástico?



Google autonomous driving

O Ambiente

33

□ 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”.

▫ Exemplo

- Robot seleccionando peças. É relevante considerar o que aconteceu anteriormente para escolher a acção actual?



O Ambiente

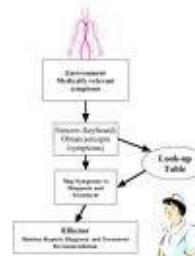
34

□ Ambientes Dinâmicos/Estáticos

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

- Sistema de diagnóstico médico.
Dinâmico ou estático?



O Ambiente

35

□ Ambientes Discretos/Contínuos

- Diz-se discreto quando origina séries de percepções e ações perfeitamente distintas umas das outras. Caso contrário, diz-se contínuo.
- Existe um número finito de estados ou de percepções/ações?
- Exemplos:
 - Jogo de Poker: Discreto
 - Condução de um veículo: Contínuo

O Ambiente

36

□ Exemplos de Ambientes

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

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

Agentes Aprendizes

37

- Os agentes racionais apresentados anteriormente mostram como escolher acções em função das circunstâncias do ambiente.
 - ▣ Mas quem implementa os programas?
 - ▣ Como se garante que os mapeamentos condição/acção são adequados?
 - ▣ **E se o ambiente mudar?**

Agentes Aprendizes

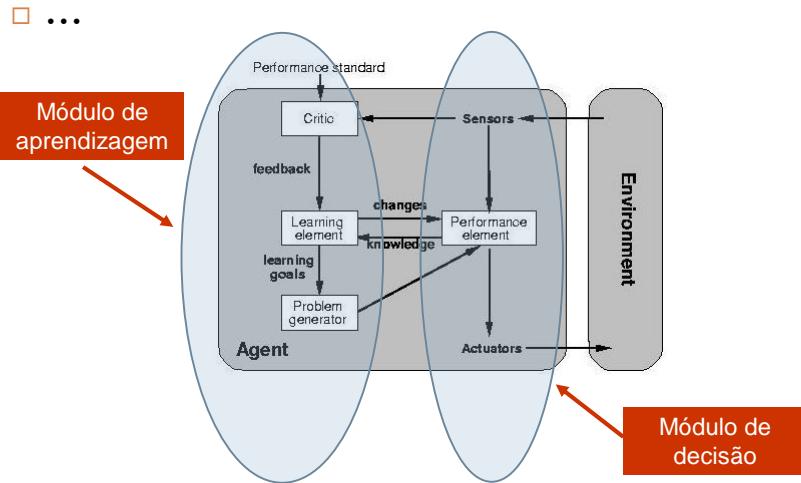
38

- ...
 - ▣ A solução consiste em construir sistemas que tenham capacidade de aprendizagem

“Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's?”

Agentes Aprendizes

39



INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

CAP. 3 AGENTES PARA RESOLUÇÃO DE PROBLEMAS

Carlos Pereira
ISEC

Índice

2

- Índice
 - Agentes de Resolução de Problemas
 - Formulação de Problemas

Agentes de Resolução de Problemas

3

- Agentes de Resolução de Problemas
 - São do tipo Guiado por Objectivos (goal-oriented).
 - Definido um “problema”, e no que consiste a sua “solução”, define-se um processo de pesquisa que procure essa solução
 - Problema
 - Diferença entre o “Estado Actual” e o “Objectivo” a atingir
 - Resolução
 - Especificação de uma sucessão de estados: (estado inicial,...,estado Final)

Agentes de Resolução de Problemas

4

- ...
 - Algoritmo Geral de Pesquisa – AGP (*General Problem Solving - GPS*)
 - Existem diversas variantes, mais ou menos complexas, que proporcionam melhores ou piores soluções.
 - Consideram-se três fases na construção destes agentes:
 - Formulação:
 - Formulação do objectivo (*goal*) e do problema
 - Pesquisa da solução
 - Execução

Agentes de Resolução de Problemas

5

□ ...

- Muitos dos problemas são difíceis de resolver, porque:
 - O número de soluções possíveis é extremamente elevado, impossibilitando uma análise exaustiva a todas elas;
 - Dificuldade de obtenção de soluções válidas devido a um elevado número de restrições associadas ao problema;
 - Em ambientes dinâmicos, o problema modifica-se ao longo do tempo;
 - ...

Agentes de Resolução de Problemas

6

■ Formulação do Objectivo

■ Objectivo

- é um estado ou um conjunto de estados do ambiente tais que, quando nesse(s) estado(s), o fim pretendido foi alcançado.
- Implica a necessidade de definição de “estado”
 - Cada acção do agente determina uma mudança de estado
 - Quando o objectivo for atingido, o conjunto de estados que as acções do agente determinou que fossem percorridos, constitui a solução do problema.

Agentes de Resolução de Problemas

7

□ ...



Agentes de Resolução de Problemas

9

■ Pesquisa da Solução

- No exemplo das cidades, quando a cidade actual coincidir com a cidade objectivo, foi encontrada uma solução.
- Existem várias soluções possíveis, porque vários caminhos conduzem ao mesmo objectivo.
- A Pesquisa corresponde à **procura de um caminho** que conduz à solução, ou à **procura do melhor caminho** de entre todos os possíveis.

Agentes de Resolução de Problemas

10

■ ...

- Um algoritmo de pesquisa:
- Tem como entrada a descrição de um problema: Estado Inicial; Estado Final; Acções Possíveis; Teste de Objectivo Atingido.
- Devolve uma sequência de acções que conduzem:
 - Do Estado Inicial ao Objectivo através de uma série de Estados Intermédios.
 - A Solução do Problema é esta sequência de acções.

■ Execução

- Conhecida uma solução, as acções podem ser executadas
 - Por exemplo, um humano iniciaria a marcha até ao objectivo

Formulação de Problemas

11

□ 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.).

Formulação de Problemas

12

□ ...

■ 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**.

Formulação de Problemas

13

- 1

■ 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

Exemplos de problemas

14

□ Problema do Caixeiro-Viajante (TSP)

- ❑ Visitar todas as localidades de um mapa, partindo de uma delas e retornando a ela.
 - Visitar cada localidade uma só vez.
 - Minimizar distância total
 - ❑ Conhecida a distância entre cada par de cidades, como deve ser planeado o itinerário?



Exemplos de problemas

15

- ...
- 1962

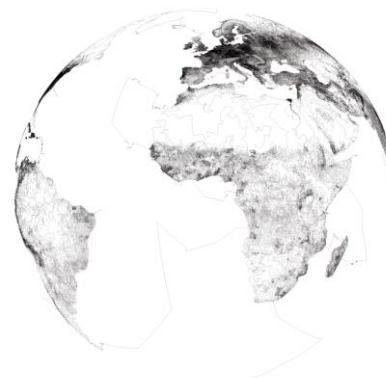


■ <http://www.tsp.gatech.edu/gallery/graphics/car54.html>

Exemplos de problemas

16

- ...
- 2006:
 - World TSP Contest
 - 1.904.711 localidades



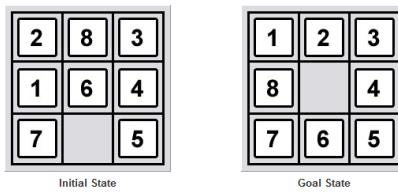
■ <http://www.tsp.gatech.edu/world/>

Exemplos de problemas

17

□ Puzzle-8

- Estados: A localização das 8 peças (A localização do quadrado vazio simplifica a implementação).
 - Operadores: O quadrado vazio move-se para a direita, esquerda, cima ou baixo (excepto nos limites)
 - Teste de Objectivo: As 8 peças ocupam as posições “Goal State”
 - Custo do Caminho: Soma de todos os movimentos.



Exemplos de problemas

18

□ Missionários e Canibais (Amarel, 1968)

□ Analisa a questão da formulação de um problema.

“3 missionários e 3 canibais encontram-se na margem de um rio e querem atravessar para a outra margem num barco que só leva 2 pessoas. Como conseguir esta travessia de modo a que nunca fiquem mais canibais que missionários, juntos, na mesma margem ?”



Exemplos de problemas

19

- ...

- Estados: Um terno (a,b,c) representando o número de missionários, canibais e barcos na margem origem do rio.
- Estado Inicial = $(3,3,1)$: 3 missionários, 3 canibais e 1 barco na margem origem.

- Operadores:

- Existem 5 possíveis operadores
 - Transportar 2 missionários, Transportar 2 canibais, Transportar 1 missionário, Transportar 1 canibal, Transportar 1 missionário e 1 canibal

Exemplos de problemas

20

- ...

- Teste de Objectivo:
 - Foi atingido o estado $(0,0,0)$?
- Custo do Caminho:
 - Número de travessias.
- Solução?

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

CAP. 4 PESQUISA NÃO INFORMADA

Carlos Pereira
ISEC

Índice

2

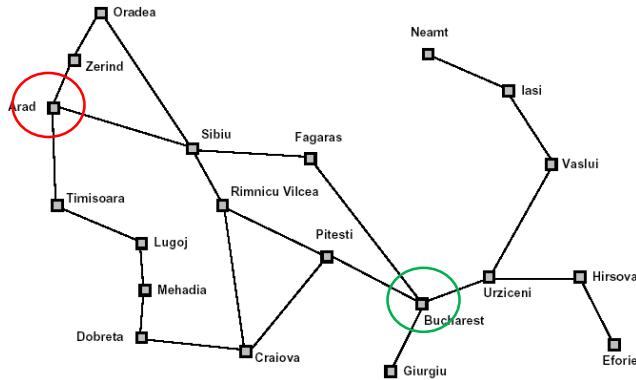
□ Índice

- Estratégias de Pesquisa
- Algoritmo Geral de Pesquisa
- Pesquisa em Largura
- Pesquisa em Profundidade
- Pesquisa Uniforme
- Pesquisa Profundidade Limitada
- IDS

Estratégia de Pesquisa

3

- Encontrar um caminho de Arad para Bucharest:



Estratégia de Pesquisa

4

- ...

1. Estado Actual = Arad
2. Se o estado actual não é Bucharest
3. Aplicar os operadores ao estado para gerar um novo conjunto de estados (expansão)
 - na primeira iteração, obtêm-se os sucessores de Arad= {Zerind, Sibiu, Timisoara}
4. Escolher um destes estados
5. Voltar ao ponto 2

Estratégia de Pesquisa

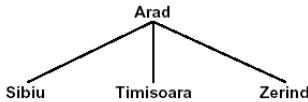
5

□ ...

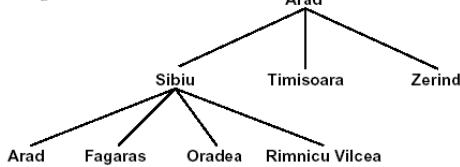
(a) The initial state

Arad

(b) After expanding Arad



(c) After expanding Sibiu



Estratégia de Pesquisa

6

□ ...

- ❑ Uma pesquisa constrói uma **Árvore de Pesquisa** cuja raiz é o Estado Inicial.
 - Em cada iteração as folhas são nós sem sucessores, porque não foram ainda expandidos (ou já o foram mas não possuem sucessores).
 - Os nós ainda não expandidos chamam-se Nós Fronteira.
- ❑ A ordem pela qual um conjunto de estados é expandido (ponto 4) define a **Estratégia da Pesquisa**.

Estratégia de Pesquisa

7

□ 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?

Estratégia de Pesquisa

8

□ ...

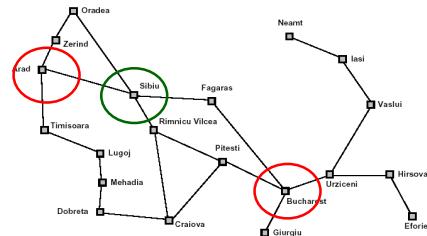
- Definem-se dois tipos de Pesquisa
 - Pesquisa Cega ou Não Informada
 - Procura uma solução sem recorrer a qualquer informação adicional que a guie.
 - Apenas pode comparar o estado actual com o objectivo
 - As variantes deste tipo de pesquisa variam de acordo com a ordem definida para expansão de estados
 - Pesquisa Heurística ou Informada
 - Procura uma solução recorrendo a informação adicional que permite escolher o nó a expandir primeiro

Estratégia de Pesquisa

9

□ ...

■ Exemplo de Pesquisa Informada:



- Como Bucharest fica a sudeste de Arad, e dos sucessores de Arad apenas Sibiu se encontra nessa direcção, Sibiu poderia ser seleccionada para expansão em primeiro lugar.

Algoritmo Geral de Pesquisa – AGP

10

□ Algoritmo

função pesquisa_geral (problema, estratégia):

devolve solução ou falhanço

Inicializa a raiz árvore de pesquisa com o estado inicial

Repete as seguintes acções

estado_atual ← Escolhe_estado (estratégia)

Se estado_atual = Ø Então

Devolve pesquisa falhou

Senão Se estado_atual = Objectivo Então

Devolve Solução

Senão

Expande estado_atual

Actualiza árvore de pesquisa

fim_Repete

fim_de_função

Algoritmo Geral de Pesquisa

11

□ Implementação...

- Uma árvore **Ar** de raiz = Estado Inicial regista os caminhos já gerados por aplicação dos operadores do problema aos nós que vão sendo expandidos;
- Uma lista **NósPorExpandir** contém os nós fronteira (da árvore) a cada momento. A ordem pela qual os sucessores de um nó são inseridos nesta lista determina qual a variante do AGP que se está a considerar.
- Uma lista **NósExpandidos** contém os nós que já foram expandidos. Esta lista evita que o mesmo nó seja expandido várias vezes, o que poderia originar loops infinitos (o mesmo nó pode ser atingido por vários caminhos)

Algoritmo Geral de Pesquisa

12

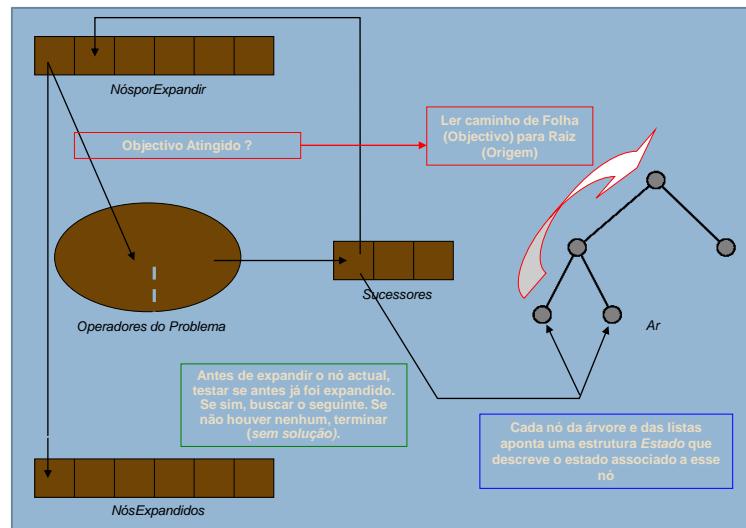
□ ...

- Cada nó a expandir é removido da 1^a posição de NósPorExpandir e inserido em NósExpandidos. São-lhe aplicados os operadores do problema e gerada uma lista de **Sucessores**. Estes Sucessores são colocados em NósPorExpandir e simultaneamente inseridos em Ar.
- Antes de expandir um nó, testa-se se esse nó é o objectivo. Se sim, termina.
 - A solução encontra-se em Ar e pode ser obtida atravessando-a desde esse nó até à raiz.

Algoritmo Geral de Pesquisa

13

□ ...



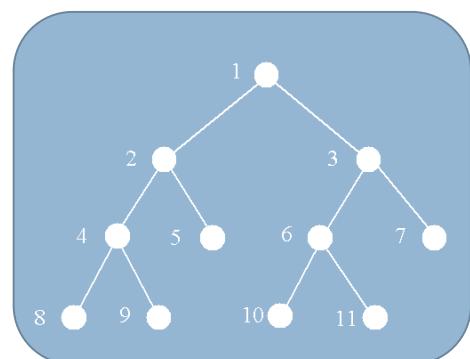
Pesquisa em Largura

14

□ Pesquisa em Largura (Breadth-First)

□ Estratégia

- A partir da raiz, a árvore é expandida por níveis
- Nós que se encontram a uma profundidade N são expandidos antes dos que se encontram a uma profundidade $N+1$



Pesquisa em Largura

15

□ ...

```

função pesquisa_em_largura(problema, insere_fila):
    devolve solução ou falhanço
    Fila_nos ← cria_fila.estado_inicial)
    Repete as seguintes acções
        Se fila_vazia(Fila_nos) Então
            Devolve pesquisa falhou
            estado_actual ← Retira_primeiro_fila(Fila_nos)
        Se estado_actual = Objectivo Então
            Devolve Solução
        Senão
            Insere_final(Fila_nos, expansão(estado_actual))
        fim_Repete
    fim_de_função

```

Pesquisa em Largura

16

□ ...

■ 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 uniformemente com a profundidade, as soluções com menos nós representam menor custo.

Pesquisa em Largura

17

□ ...

■ Desvantagens:

- O custo da pesquisa é muito elevado

- Complexidade temporal exponencial
- Complexidade espacial exponencial

■ Análise da Complexidade da Pesquisa em largura

- A Pesquisa em Largura tem complexidade temporal e espacial $O(b^d)$, com b=Factor de Ramificação e d=Número de Níveis da Árvore.

- Se considerarmos um factor de ramificação de 8, o número de nós expandidos é de $1+8+8^2+8^3+8^4+\dots+8^k$

Pesquisa em Largura

18

■ ...

- Para uma Pesquisa em Largura com branching factor (b) = 10 e Processamento executado a 1000 nós/segundo e ocupando 100 bytes/nó:

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

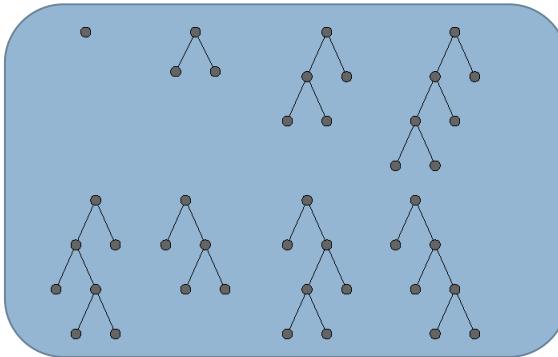
- Problemas de pesquisa cujos algoritmos têm complexidade exponencial, **apenas podem ser resolvidos para instâncias de pequena dimensão**

Pesquisa em Profundidade

19

□ Pesquisa em Profundidade

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



Pesquisa em Profundidade

20

□ 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 em Profundidade

21

□ ...

■ Vantagens da Pesquisa em Profundidade

Considere-se um Espaço de Estados com Factor de Ramificação “b” e profundidade máxima “d”:

- A Complexidade Temporal é $O(b^d)$ (como na Pesquisa em Largura, porque o número total de nós a gerar é o mesmo).
- A Complexidade Espacial é de apenas $O(b.d)$ porque não há necessidade de ter mais que $b.d$ nós em memória simultaneamente - necessita de pouca memória!

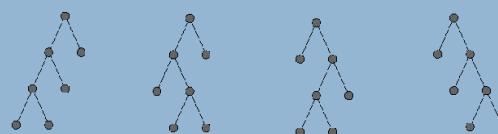
Pesquisa em Profundidade

22

□ ...

■ Considerando $b=2$, $d=3$:

■ No máximo há em memória $2^3=6$ nós + raiz



Pesquisa Uniforme

23

□ Variante da Pesquisa em Largura

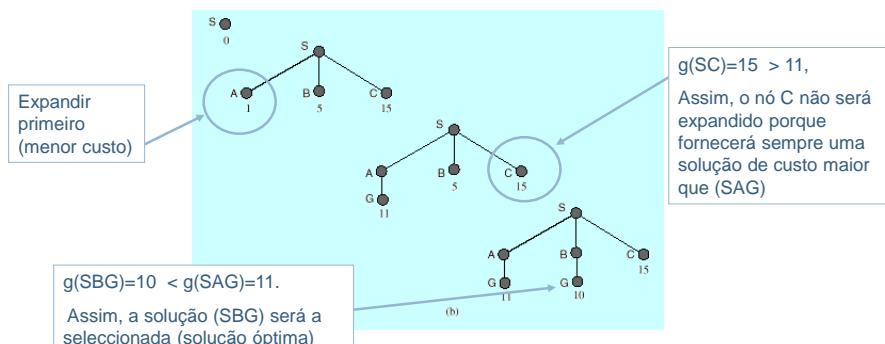
- Consiste em expandir primeiro os nós que têm um custo associado menor
 - Esta expansão pára quando for encontrada uma solução e o custo acumulado dos caminhos associados aos nós que falta expandir já for superior à solução encontrada.
- Garante a solução óptima, bastando apenas que o custo aumente 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.

Pesquisa Uniforme

24

□ ...

- Seja “g” o custo do caminho percorrido e “G” o nó objectivo



Pesquisa Uniforme

25

- ...

- Características

- Completa

- Óptima

- desde que o custo aumente com a profundidade: $g(\text{Sucessor}(n)) \geq g(n)$

- 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

26

- 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.

- Pode aplicar-se em espaços onde se sabe que a solução terá de existir dentro da profundidade "m"

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

Pesquisa Profundidade Limitada

27

- ...

- Características:

- Completa
 - Não Óptima
 - Complexidade Temporal $O(b^m)$
 - Complexidade Espacial $O(b \cdot m)$

IDS - Pesquisa por Aprofundamento Progressivo

28

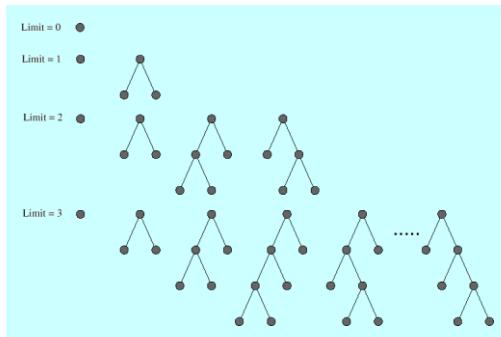
- A Pesquisa por Aprofundamento Progressivo (IDS - Iterative Deepening Search)

- 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 raiz)

IDS - Pesquisa por Aprofundamento Progressivo

29

□ ...



IDS - Pesquisa por Aprofundamento Progressivo

30

□ ...

■ Características

- Óptima, nas condições da Pesquisa em Largura (custo = função da profundidade)
- Completa, como a Pesquisa em Largura
- Complexidade Espacial $O(b \cdot m)$ (como a Pesquisa em Profundidade)
- Complexidade Temporal $O(b^m)$ (como a Pesquisa em Profundidade)

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

CAP. 5 PESQUISA INFORMADA (HEURÍSTICA)

Carlos Pereira
ISEC 19-20

Índice

2

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

Motivação

3

- 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

Métodos do tipo “Best-First”

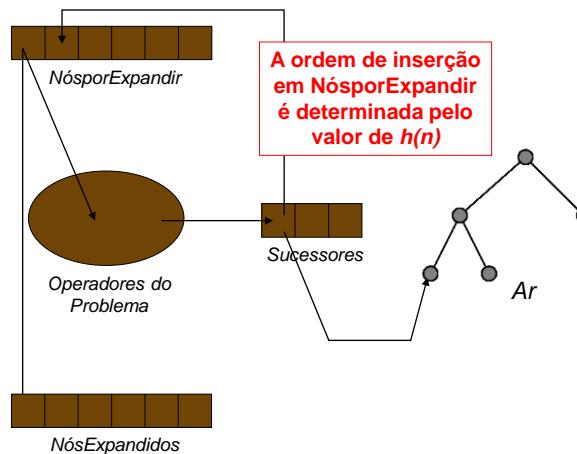
4

- Método “Best-First” – O melhor primeiro
 - ▣ Recorrem a conhecimento pericial para definir uma função de avaliação
 - Conhecimento específico acerca do domínio do problema
 - ▣ A Função de avaliação de um estado n - $h(n)$ retorna um valor indicativo da vantagem em expandir esse estado primeiro.
 - De acordo com a estrutura do AGP, cada nó sucessor é inserido ordenadamente na lista de nós a expandir, em função do valor de $h(n)$

Métodos do tipo “Best-First”

5

□ ...



Métodos do tipo “Best-First”

6

□ Pesquisa Sôfrega

- Expande-se em primeiro lugar o nó que parece estar mais perto do objectivo.
 - em muitos problemas pode obter-se uma estimativa do custo do caminho de um dado nó até ao objectivo
 - A estimativa é calculada por $h(n)$ - Função Heurística
 - Se $h(n)=0$, o nó n coincide com o nó objectivo
 - Se $h(n)>=0$, o nó objectivo pode ser atingido a partir do nó n , sendo o custo estimado em $h(n)$
 - Se $h(n)=\infty$, o objectivo não pode ser atingido a partir do nó n

...Pesquisa Sôfrega

7

□ ...

■ Exemplo

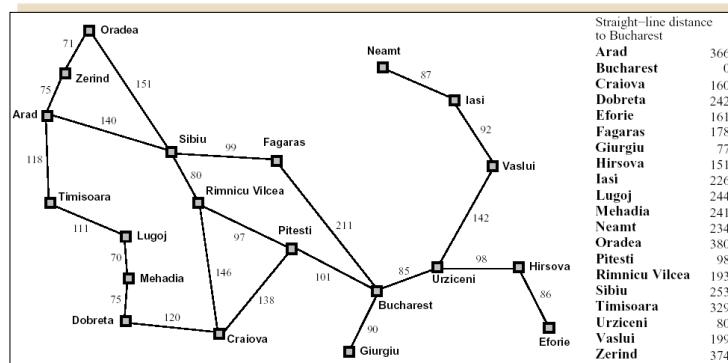
- Que heurística poderemos utilizar para o problema de determinar o melhor caminho entre duas cidades?
- poderá ser a Distância em Linha Recta, de cada cidade à cidade objectivo
- Neste caso, para calcular $h(n)$, basta ter as coordenadas (x,y) de cada cidade

...Pesquisa Sôfrega

8

□ ...

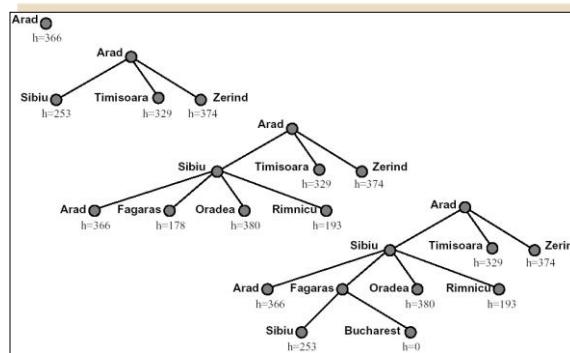
■ ?



...Pesquisa Sôfrega

9

□ ...



- Solução: Arad → Sibiu → Fagaras → Bucharest

...Pesquisa Sôfrega

10

□ ...

□ Características

- Complexidade temporal exponencial - $O(b^d)$
 - com b =Factor de Ramificação e d =Número de Níveis da Árvore (máxima profundidade do espaço)
- Complexidade espacial exponencial - $O(b^d)$
 - Complexidades temporal e espacial podem ser substancialmente reduzida se $h(n)$ for adequada
- Não óptima
 - não garante que se encontre o caminho de menor custo
- Incompleta
 - pode seguir caminhos infinitos

Pesquisa A*

11

- Pesquisa A*

- Combina Pesquisa Uniforme com Pesquisa Sôfrega

- Pesquisa Uniforme

- Guiada pelo custo mínimo da origem a cada nó n, expande primeiro o nó efectivamente mais perto da origem
- Óptima se o custo aumentar com a profundidade
- Complexidade $O(b^d)$

Pesquisa A*

12

- ...

- Pesquisa Sôfrega:

- Guiada pelo custo mínimo aparente de cada nó n até ao objectivo, expande primeiro o nó que parece mais perto do objectivo
- Não óptima
- De complexidade $O(b^d)$ mas muito reduzida para boas funções $h(n)$

Pesquisa A*

13

□ ...

- Estes dois tipos de pesquisa são complementares:
 - A Uniforme “mede” a parte inicial do percurso - $g(n)$
 - A Sôfrega “mede” a aparente parte restante - $h(n)$
 - Os custos do caminho provenientes de ambas podem combinar-se numa simples soma: $f(n)=g(n)+h(n)$
 - “mede” o custo estimado da solução que passa pelo nó n
 - No AGP, a inserção na lista de Nós a Expandir é feita por ordem crescente de $f(n)$

Pesquisa A*

14

□ ...

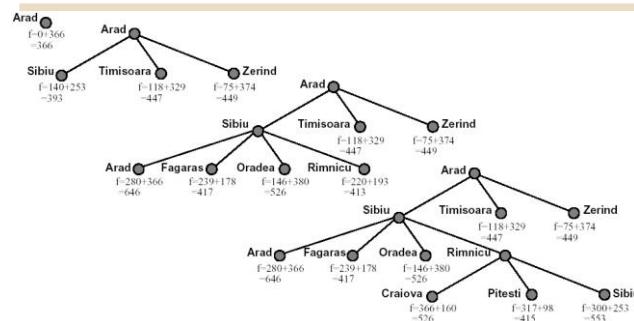
- A pesquisa A* é óptima e completa desde que:
 - A heurística utilizada nunca sobreestime o custo do caminho do nó n até ao objectivo (isto é, nunca possa assumir um valor superior ao do custo real).
 - Se assim for, constitui uma **Heurística Admissível**.
 - Será a distância em linha recta entre cidades uma heurística admissível?

Pesquisa A*

15

□ ...

■ Exemplo:



$$\blacksquare \quad f(n) = g(Arad, n) + h_{SLD}(n, Bucharest)$$

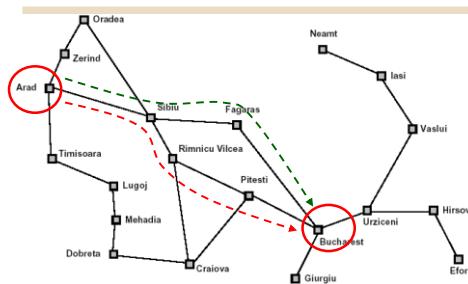
Pesquisa A*

16

□ ...

■ Solução:

- Arad → Sibiu → Rimnicu → Pitesti → Bucharest



Pesquisa A*

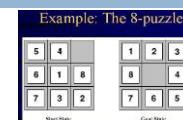
17

- ...
- Outros exemplos:
 - ▣ <http://aispace.org/search/>
 - ▣ <http://aigamedev.com/open/interviews/mario-ai/>

Funções Heurísticas

18

- Considere-se o Puzzle de 8 peças:
 - Factor de ramificação médio: $b=3$ (quadrado vazio no meio=4, no canto=2, junto a um lado=3)
 - Tipicamente, uma solução ocorre em 20 passos, Portanto, uma pesquisa exaustiva procuraria, em média, num espaço de 3^{20} estados
 - Evitando a passagem por estados repetidos, este número pode reduzir-se a 362.880
 - O recurso a uma heurística efectua uma redução significativa do número de estados!



Funções Heurísticas

19

□ ...

■ Qual a melhor heurística?

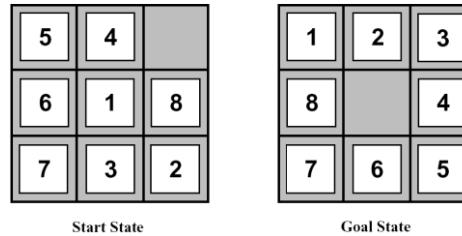
- A heurística deve escolher nunca pode sobre-estimar o custo do caminho até à solução.
- Uma possibilidade:
 - h_1 = Número de peças que, em cada estado, estão numa posição não coincidente com a posição que devem ocupar na solução
 - nunca sobre-estima o custo do caminho, porque cada peça fora de ordem terá de ser movida pelo menos uma vez:
Assim, 7 é o custo mínimo do caminho até à solução

Funções Heurísticas

20

□ ...

■ Exemplo:



■ Estado inicial: $h=7$

■ Estado Final: $h=0$

Funções Heurísticas

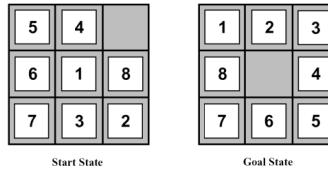
21

□ ...

■ Outra Possibilidade

- $h_2 = \text{Soma das distâncias das peças até à sua posição final.}$

- Soma das distâncias horizontais com as verticais (Manhattan Distance)



- No exemplo, para o Estado Inicial, $h=2+3+3+2+4+2+0+2=18$
- h nunca sobre-estima o custo do caminho, porque cada peça fora de ordem terá de ser movida pelo menos x vezes até atingir a sua posição final (não podem mover-se na diagonal)

Funções Heurísticas

22

□ ...

■ Uma heurística é caracterizada pelo Factor de Ramificação Efectivo, b^* :

- Seja N o número total de nós expandidos pela pesquisa A*,
Então: $N=1+b^*+(b^*)^2+\dots+(b^*)^d$

- Uma boa heurística apresenta um b^* próximo de 1

- $b^*=1$ significa que a pesquisa escolheria sempre o nó correcto a expandir, progredindo sempre, sem erros, em direcção à solução!

Funções Heurísticas

23

□ ...

- 100 puzzles gerados aleatoriamente
- Soluções a profundidades que variam entre 2 e 24
 - Nós expandidos e factor de ramificação efectivo (b^*) para as heurísticas h_1 e h_2 do puzzle 8:

d	Search Cost			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

Funções Heurísticas

24

□ ...

- Uma função heurística que calcule o custo exacto da solução para um problema com menos restrições, embora derivado do original (relaxed problem) constitui geralmente uma boa solução
- A função heurística cujo valor se aproxima mais do custo da solução real, é habitualmente a melhor

Funções Heurísticas

25

- ...

- **Puzzle-8 com menos restrições:**

- **Exemplo 1**

- Um puzzle em que as peças se podem mover logo para a posição final (em vez de uma só posição de cada vez e apenas para a posição livre)

- **Exemplo 2**

- Um puzzle em que as peças se possam mover uma só posição de cada vez, mas para qualquer posição, e não apenas para a livre

- Qual a melhor heurística para cada um dos exemplos? h1 ou h2?

Funções Heurísticas

26

- ...

- **No exemplo 1**

- h1 daria o custo exacto da solução!

- **No exemplo 2,**

- h2 daria o custo exacto da solução!

Variantes A*

27

□ 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*

28

□ IDA*

■ Está para 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, $f(n)=g(n)+h(n)$

IDA*

29

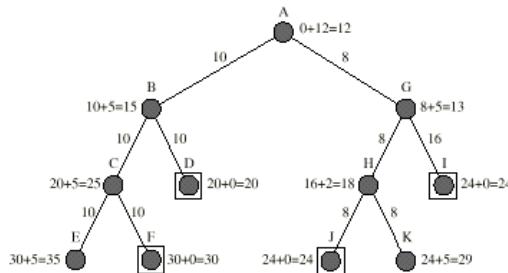
□ ...

- Para cada “limite de custo estimado”, f_i , “exclui” os nós cujo valor f é superior
- Pára quando atingir um nó objectivo cujo $f \leq$ que o limite actual
- Enquanto não encontrar um objectivo nestas condições, progride para o limite seguinte, f_i+1 . Este limite, f_i+1 :
 - Pode provir de outro nó situado à mesma profundidade do que proporcionou o limite anterior, f_i : O IDA* é controlado pelo valor de f e não pela profundidade do nó, d
 - É determinado na iteração i , escolhendo o menor custo estimado de entre todos os custos estimados associados aos nós por expandir

IDA*

30

□ Exemplo



- D, I, F e J representam todos o objectivo a atingir, assim, os ramos que a eles conduzem são **vários caminhos possíveis** até esse objectivo

IDA*

31

□ ...

■ Estado Inicial

■ Nós por Expandir: A

■ Limite Iteração Seguinte =

= Custo Estimado (A, objectivo) =

= f_1 =

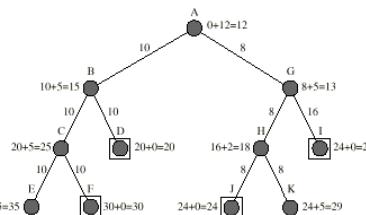
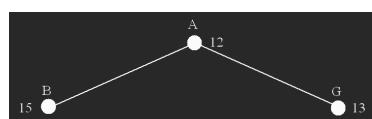
= $\min(f_{\text{NósPorExpandir}}) = g(A) + h(A) = 0 + 12 = 12$

IDA*

32

□ ...

■ Iteração 1



■ A foi expandido

■ Limite Actual= $f_1=12$

■ Como $f(B)=15$ e $f(G)=13$ são >12 , B e G são excluídos

■ Limite Iteração Seguinte = $f_2 = \min(f_{\text{NósNãoExpandidos}}) =$

= $\min(f(B), f(G)) = 13$

■ o nó G será o próximo a expandir (por ser o 1º da lista G,B)

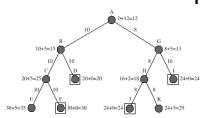
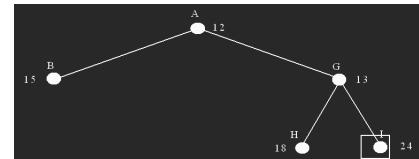
IDA*

33

... □

Iteração 2

- G foi expandido
- Limite Actual = $f_2 = 13$
- Como $f(H) = 18 > 13$, H é excluído
- Como $f(I) = 24 > 13$, I é excluído, apesar de ser Nό Objectivo
- Como o limite 13 não conduziu a nada, tomar o limite seguinte:
Limite Iteração Seguinte = $f_3 = \min(f_{\text{NósNãoExpandidos}}) = f(B) = 15$
- Como em termos de f ao nó G se segue B (e G já foi expandido), então AB é o caminho escolhido pelo A* em seguida



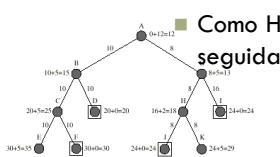
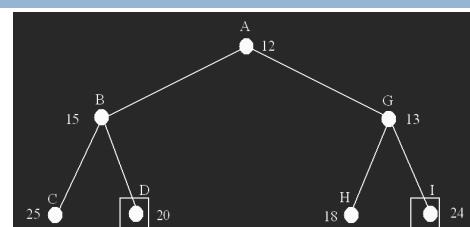
IDA*

34

... □

Iteração 3

- B foi expandido
- Limite Actual = $f_3 = 15$
- Como $f(C) = 25 > 15$, C é excluído
- Como $f(D) = 20 > 15$, D é excluído, apesar de ser Nό Objectivo
- Como o limite 15 não conduziu a nada, tomar o limite seguinte:
Limite Iteração Seguinte = $f_4 = \min(f_{\text{NósNãoExpandidos}}) = f(H) = 18$
- Como H é o nó de menor f ainda por expandir, AGH é usado em seguida



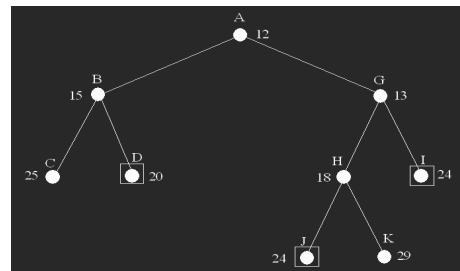
IDA*

35

□ ...

■ Iteração 4

- H foi expandido
- Limite Actual = $f_4 = 18$



- Como $f(J)=24>18$, J é excluído, apesar de ser Nό Objectivo
- Como $f(K)=29>18$, K é excluído
- Como o limite 18 não conduziu a nada, tomar o limite seguinte:
- Limite Iteração Seguinte = $f_5 = \min(f_{\text{NósNãoExpandidos}}) = f(D) = 20$
- Como a ordem de expansão imposta pelo A* era (H,D,C,I) e o nó H já foi expandido, em seguida usa-se o caminho ABD

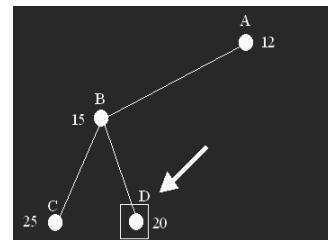
IDA*

36

□ ...

■ Iteração 5

- Limite Actual = $f_5 = 20$



- Mas, antes de se expandir D, verifica-se que ele é nό objectivo ...
- ... e que se encontra dentro do limite actual, porque $f(D)=20$ e $f_5=20$

... Assim, D é a solução (óptima) procurada

IDA*

37

- ...

- 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*

38

- 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
- ...

SMA*

39

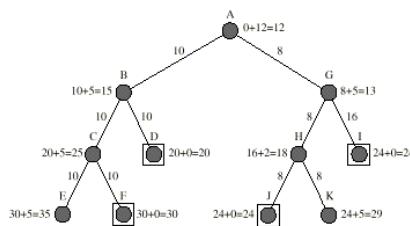
□ ...

- 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.

SMA*

40

□ Exemplo



- D, F e J são nós que contêm o estado objectivo
- Suponha-se que a memória só tem capacidade para 3 nós.

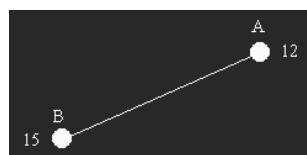
SMA*

41

 ... Estado Inicial

SMA*

42

 ... Iteração 1

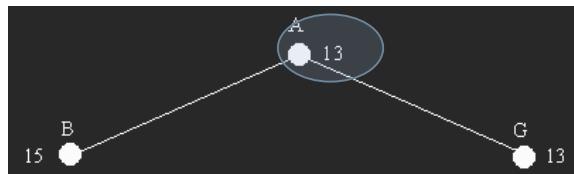
- Iniciar a expansão do nó A adicionando-lhe o primeiro dos seus filhos, B

SMA*

43

□ ...

■ Iteração 2



Como ainda só há 2 nós em memória ($< \text{limite} = 3$), continuar a expansão do nó A, adicionando-lhe o segundo dos seus filhos, G

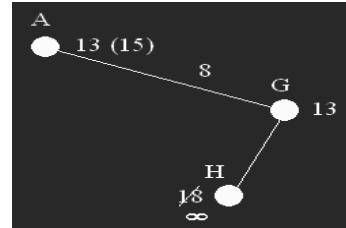
- O nó A encontra-se agora completamente expandido: actualiza-se o valor f do nó pai com o mínimo dos valores f dos seus filhos, $f(A)=f(G)=13$

SMA*

44

□ ...

■ Iteração 3



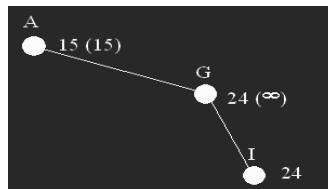
- Elimina-se o nó de maior profundidade e com maior valor de f (nó B) e o seu valor, $f(B)=15$, é guardado em A (entre parêntesis)
- H não é objectivo!
 - Por outro lado, para expandir H seria preciso mais memória, uma vez que em H já há 3 nós ocupados
 - Portanto, o caminho nunca poderá ser construído através de H: Marca-se H com $f(H)=\infty$

SMA*

45

□ ...

■ Iteração 4



SMA*

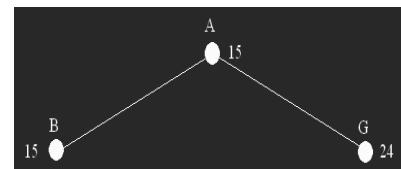
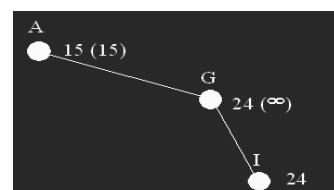
46

□ ...

■ Iteração 5

I é um estado objectivo, mas como $f(I)=24 > f(A)=15$, é possível que haja uma solução melhor, e portanto a solução AGI não é considerada! Portanto, há que expandir A “para o outro lado”.

O nó B pode ser gerado (pela 2ª vez) e adicionado a A

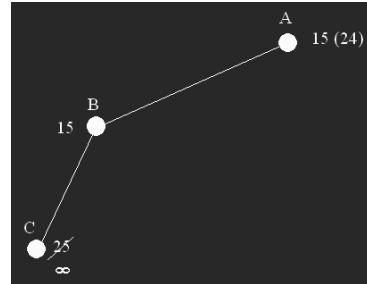


SMA*

47

□ ...

■ Iteração 6



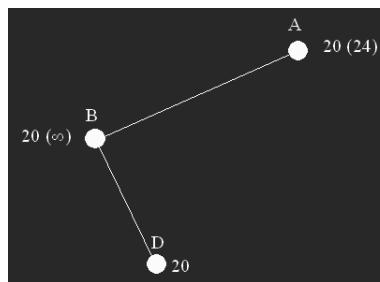
- C (tal como H no 3º passo) não é objectivo e origina a presença de 3 nós em memória, portanto, o caminho nunca poderá ser construído através de C:
- Marca-se C com $f(C)=\infty$

SMA*

48

□ ...

■ Iteração 7



o nó B foi completamente expandido. Quando isto acontece, actualiza-se o valor f do nó pai com o mínimo dos valores f dos seus filhos , $f(B)=\min(\infty, 20)=20$

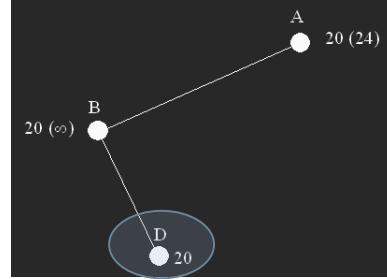
Como $f(B)$ muda, $f(A)$ também tem de ser actualizado. Como $f(G)=24$ ficou guardado em A entre parêntesis, o mínimo f entre os filhos de A é $\min(20, 24)=20$

SMA*

49

□ ...

■ Iteração 8



- D é agora o nó seleccionado para expansão, porém, como é nó objectivo e $f(D)$ não é maior que nenhum dos valores de f noutras nós da árvore, o caminho ABD é a solução

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

20/21

06. PESQUISA LOCAL – MELHORAMENTO ITERATIVO

Carlos Pereira
ISEC

Índice

2

- Índice
 - Problemas com restrições
 - Técnicas de Melhoramento Iterativo
 - Trepa-Colinas
 - Recristalização Simulada
 - Pesquisa Tabu

Problemas com Restrições

3

- 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

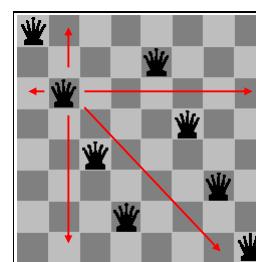
Problemas com Restrições

4

- ...

- Exemplo

- Problema das 8 Rainhas



- Variáveis: Localizações das 8 rainhas no tabuleiro
- Domínio das Variáveis: As coordenadas de cada um dos 64 quadrados do tabuleiro
- Restrições: Quaisquer 2 rainhas (combinações da 8 duas a duas) não podem ficar na mesma linha, coluna ou diagonal

Problemas com Restrições

5

- ...

- Num CSP interessa determinar apenas um “estado” final válido e não um caminho que leve a esse estado
 - O “**Estado Final**” é **desconhecido** e constitui a solução do problema:
- No caso do exemplo das 8 rainhas
 - Dispor 8 rainhas de modo a que nenhuma seja atacada: Interessa apenas conhecer uma disposição das rainhas, um estado final, e não a ordem pela qual se devem colocar no tabuleiro (é indiferente!):
 - O estado final, ou solução, é composto por 8 rainhas dispostas de modo a não haver ataques

Problemas com Restrições

6

- ...

- 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
 - Simulated-Annealing
 - Pesquisa Tabu
 - ...

Problemas com Restrições

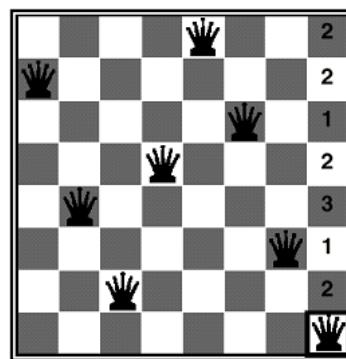
7

- Heurística dos “Conflitos Mínimos”
 - As rainhas são inicialmente dispostas 1 por coluna, e ao acaso, numa linha qualquer.
 - Os operadores de modificação movem uma rainha de cada vez.
 - O movimento faz-se colocando essa rainha na posição em que sofre menos ataques, isto é, produz menos conflitos.

Problemas com Restrições

8

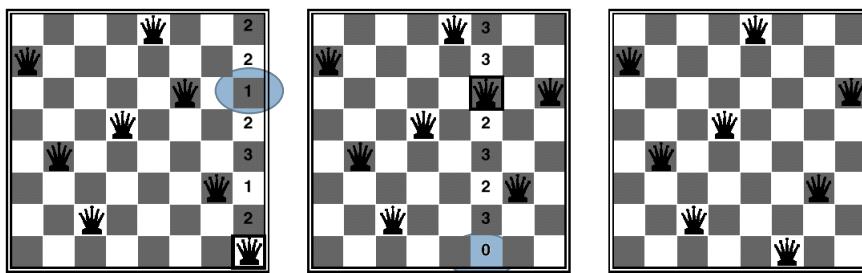
- ...



Problemas com Restrições

9

- ...



Melhoramento Iterativo

10

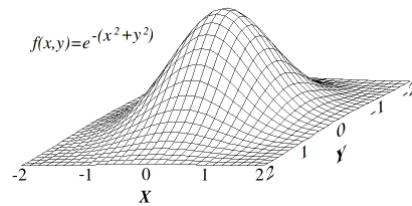
- Os algoritmos de melhoramento iterativo caracterizam-se por:
 - ▣ Não anotam os estados intermédios que conduzem a uma solução, apresentando apenas a “configuração” válida que a compõe.
 - ▣ Partem de uma configuração inicial completa (que viola as restrições), eventualmente gerada aleatoriamente, e melhoram-na sucessivamente até alcançarem uma solução.

Melhoramento Iterativo

11

□ Trepa-Colinas

- ❑ Também conhecido por “Hill-Climbing” ou “Gradient-Descent”
- ❑ O nome e ideia base provem de uma analogia com a decisão tomada por um agente que, perdido numa encosta, pretende atingir o topo:
 - Deslocar-se-á na direção “em que o caminho sobe”.



...Trepa-Colinas

12

□ ...

```

function HILL-CLIMBING(problem) returns a solution state
  inputs: problem, a problem
  static: current, a node
           next, a node
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  loop do
    next  $\leftarrow$  a highest-valued successor of current
    if VALUE[next] < VALUE[current] then return current
    current  $\leftarrow$  next
  end

```

Trepa-Colinas

13

□ ...

■ Implementação

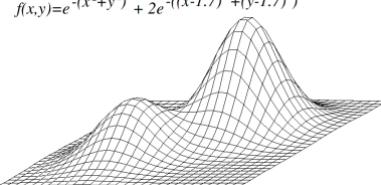
- Parte de um estado inicial dado ou gerado aleatoriamente
 - todas as variáveis com valores atribuídos
- Gera os estados sucessores do estado actual
- Através de uma Função de Avaliação, avalia cada estado assim gerado e escolhe o de maior valor
- Pára quando o estado seleccionado tiver um valor inferior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

Trepa-Colinas

14

□ 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



Trepa-Colinas

15

□ ...

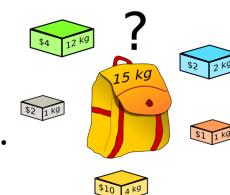
- Tentativa de resolução dos problema relativo ao atingir um ponto de não progresso:
 - Reiniciar a pesquisa partindo de um estado inicial diferente, gerado aleatoriamente (Random-Restart-Hill-Climbing)
 - Guarda o melhor resultado obtido nas pesquisas anteriores
 - até ao ponto de não-progresso
 - Pára quando atingir o número de reinícios máximo ou quando o melhor resultado guardado não for ultrapassado durante “n” iterações (valor “n” é pré-fixado)

Trepa-Colinas

16

□ Problema da Mochila

- Um conjunto de N objectos,
 - caracterizados por um peso e um lucro.
- Uma mochila com capacidade limitada.



□ Pretende-se:

- Encontrar o conjunto ideal de objectos para colocar na mochila;
- A capacidade não pode ser excedida e o **lucro deve ser máximo**

Trepa-Colinas

17

□ ...

■ Representação de uma solução

- Os objectos que estão na mochila, sequência binária com N posições (1 significa que o objecto está na mochila)

- Exemplo, para N=8:

■ **1 0 1 0 0 0 0 0**

- Esta representação admite soluções inválidas!

Trepa-Colinas

18

□ ...

■ Como definir a vizinhança de uma solução?

- Adicionar ou remover um objecto

■ Exemplo:

■ Solução actual : 1 0 1 0 0 0 0 0

Adicionar objecto 8

■ Solução vizinha: 1 0 1 0 0 0 0 1

Trepa-Colinas

19

□ ...

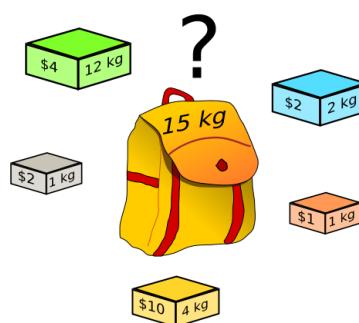
- ❑ Como decidir se uma nova solução é relevante para a pesquisa?
 - Através da Função de avaliação
 - Associa a cada solução um valor numérico (a sua qualidade)
 - Permite comparar soluções alternativas
 - Neste caso, devemos considerar o lucro e garantir que a capacidade não é ultrapassada:

$$\text{fitness}(x) = \begin{cases} \sum_{i=1}^N x[i] \cdot L[i] & \text{se } x \text{ for uma solução legal} \\ 0 & \text{se } x \text{ for uma solução ilegal} \end{cases}$$

Trepa-Colinas

20

□ ...



- Qual a solução ótima?

Trepa-Colinas

21

□ ...

■ Considere-se outro problema:

- 8 objectos
- Capacidade da mochila: 35

■ Propriedades dos objectos

Objectos	1	2	3	4	5	6	7	8
Peso	11	18	12	14	13	11	10	16
Lucro	5	8	7	6	9	6	5	3

■ Solução inicial: 1 0 1 0 0 0 0 0

■ $Q=12$

Trepa-Colinas

22

□ ...

■ Iteração 1

■ Vizinhos (considere-se a distância de hamming de 1):

Vizinhos	Qualidade
00100000	7
11100000	0
10000000	5
10110000	0
10101000	0
10100100	18
10100010	17
10100001	0

■ Nova solução: 1 0 1 0 0 1 0 0; $Q=18$

Trepa-Colinas

23

□ ...

■ Iteração 2

Vizinhos	Qualidade
001000100	13
101000100	0
101000100	11
101100100	0
101011000	0
101000000	12
101000110	0
101000101	0

- Fim da Optimização!
- Solução: 1 0 1 0 0 1 0 0 **Q=18**

Será a solução óptima?

Variantes do Trepa-Colinas

24

□ Variantes

- Permitir o deslocamento ao longo de um planalto
 - Deverá ser sempre autorizado este tipo de deslocamento?
- Trepa Colinas “First-Choice”
 - Visita os vizinhos de forma aleatória
 - Aceita um vizinho de melhor qualidade e termina iteração
 - Útil quando a vizinhança é grande
 - Algoritmo não determinista
- Random Restart
 - Aplicar o algoritmo diversas vezes com diferentes pontos de partida

Recristalização Simulada

25

- Método de Recristalização simulada (*simulated annealing*)
 - ▣ 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.

Recristalização Simulada

26

- ...
 - ▣ 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.

Recristalização Simulada

27

- ...

- 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

Recristalização Simulada

28

- ...

- 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

Recristalização Simulada

29

□ ...

```

function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
           schedule, a mapping from time to “temperature”
  static: current, a node
           next, a node
           T, a “temperature” controlling the probability of downward steps

  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  for t  $\leftarrow$  1 to  $\infty$  do
    T  $\leftarrow$  schedule[t]
    if T=0 then return current
    next  $\leftarrow$  a randomly selected successor of current
     $\Delta E \leftarrow$  VALUE[next] – VALUE[current]
    if  $\Delta E > 0$  then current  $\leftarrow$  next
    else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 

```

- $e^{\Delta E/T} = 1 / e^{-\Delta E/T}$ porque só ocorre com $\Delta E < 0$:
- A probabilidade de um “movimento para um estado pior” decresce com o tempo

Recristalização Simulada

30

□ ...

- **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!**

Pesquisa Tabu

31

- 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)

Pesquisa Tabu

32

- ...


```

função Tabu (problema, vizinhança, Memória): solução
  It ← 0
  Solução ← Gera_solução_aleatoriamente(Espaço de procura)
  Repete enquanto (it < Max_it)
    Lista ← Obtém_vizinhos (Solução, vizinhança)
    Lista ← Retira_Tabus (Lista, Memória)
    Solução ← Escolhe_melhor(Lista)
    Actualiza_memória
    it ← it + 1
  fim_Repete
  Devolve Solução
fim_de_função
```

Pesquisa Tabu

33

□ ...

■ Aplicação ao Problema da mochila

- Solução inicial: 10100000
- Vizinhança: soluções a uma distância de Hamming = 1
- Memória:
 - Guarda os movimentos recentes (Tabu)
 - Janela temporal escolhida neste caso: **2** movimentos
- Solução inicial: 1 0 1 0 0 0 0 0
 - **Q=12**

Pesquisa Tabu

34

□ ...

■ Iteração 1

■ Vizinhos

Vizinhos	Qualidade
0 0 1 0 0 0 0 0	7
1 1 1 0 0 0 0 0	0
1 0 0 0 0 0 0 0	13
1 0 1 1 0 0 0 0	0
1 0 1 0 1 0 0 0	0
1 0 1 0 0 1 0 0	18
1 0 1 0 0 0 1 0	17
1 0 1 0 0 0 0 1	0

- Nova solução: 1 0 1 0 0 1 0 0; **Q=18**

■ Memória

- Posição 6 – n. de iterações Tabu=2

Pesquisa Tabu

35

□ ...

■ Iteração 2

■ Vizinhos

Vizinhos	Qualidade
0 0 1 0 0 1 0 0	13
1 1 1 0 0 1 0 0	0
1 0 0 0 0 1 0 0	11
1 0 1 1 0 1 0 0	0
1 0 1 0 1 1 0 0	0
X 1 0 1 0 0 0 0 0	12
1 0 1 0 0 1 1 0	0
1 0 1 0 0 1 0 1	0

■ Nova solução: 0 0 1 0 0 1 0 0; Q=13

■ Memória

■ Posição 1 – n. de iterações Tabu=2

■ Posição 6 – n. de iterações Tabu=1

Pesquisa Tabu

36

□ ...

■ Iteração 3

■ Vizinhos

Vizinhos	Qualidade
X 1 0 1 0 0 1 0 0	18
0 1 1 0 0 1 0 0	0
0 0 0 0 0 1 0 0	6
0 0 1 1 0 1 0 0	0
0 0 1 0 1 1 0 0	0
X 0 0 1 0 0 0 0 0	7
0 0 1 0 0 1 1 0	18
0 0 1 0 0 1 0 1	0

■ Nova solução: 0 0 1 0 0 1 1 0; Q=18

■ Memória

■ Posição 1 – n. de iterações Tabu=1

■ Posição 7 – n. de iterações Tabu=2

Pesquisa Tabu

37

□ ...

■ Iteração 4

■ Vizinhos

Vizinhos	Qualidade
X 1 0 1 0 0 1 1 0	0
0 1 1 0 0 1 1 0	0
0 0 0 0 0 1 1 0	11
0 0 1 1 0 1 1 0	0
0 0 1 0 1 1 1 0	0
0 0 1 0 0 0 1 0	12
X 0 0 1 0 0 1 0 0	13
0 0 1 0 0 1 1 1	0

■ Nova solução: 0 0 1 0 0 0 1 0; $Q=12$

■ Memória

■ Posição 6 – n. de iterações Tabu=2

■ Posição 7 – n. de iterações Tabu=1

Pesquisa Tabu

38

□ ...

■ Iteração 5

■ Vizinhos

Vizinhos	Qualidade
1 0 1 0 0 0 1 0	17
0 1 1 0 0 0 1 0	0
0 0 0 0 0 0 1 0	5
0 0 1 1 0 0 1 0	0
0 0 1 0 1 0 1 0	21
X 0 0 1 0 0 1 1 0	18
X 0 0 1 0 0 0 0 0	7
0 0 1 0 0 0 1 1	0

■ Nova solução: 0 0 1 0 1 0 1 0; $Q=21$

■ Memória

■ Posição 5 – n. de iterações Tabu=2

■ Posição 6 – n. de iterações Tabu=1

Pesquisa Tabu

39

□ Características

■ Vantagens

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

■ Desvantagens

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

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

20-21

07. ALGORITMOS GENÉTICOS

Carlos Pereira
ISEC

Índice

2

□ Índice

- Introdução
- Funcionamento
- Selecção
- Recombinação
- Mutação

Introdução

3

□ Computação Evolucionária (CA)

- ❑ A área de investigação designada por “Computação evolucionária (EC)” envolve:
 - Algoritmos Genéticos (GA)
 - Estratégias Evolucionárias (ES)
 - Programação Evolucionária (EP)

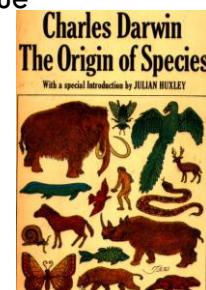


Introdução

4

□ Algoritmos Genéticos

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

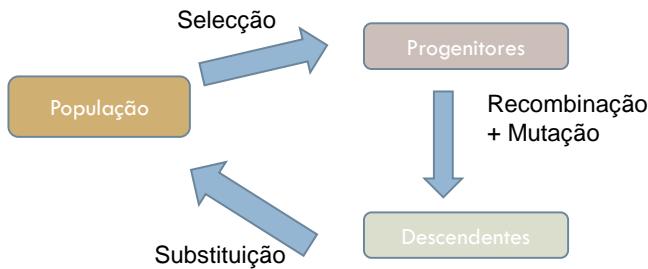


- Sub-classe da computação evolucionária.
 - A Computação evolucionária desenvolveu-se nos anos 60
 - Os GAs foram criados a meio da década de 70 por John Holland

Introdução

5

□ Princípio Básico de Funcionamento



6

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 **probabilisticamente** as melhores hipóteses entre si, e “mutando” algumas outras.

Fucionamento

7

□ Protótipo

- **Fitness:** Função de avaliação
- **Fitness_threshold:** Critério de Fim de Ciclo
- **p** = Número de hipóteses da população
- **r** = Fracção da população a ser recombinada
- **m** = Mutation Rate

$GA(Fitness, Fitness_threshold, p, r, m)$

- **Initialize:** $P \leftarrow p$ random hypotheses
- **Evaluate:** for each h in P , compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness_threshold$
 1. **Select:** Probabilistically select $(1 - r)p$ members of P to add to P_s .

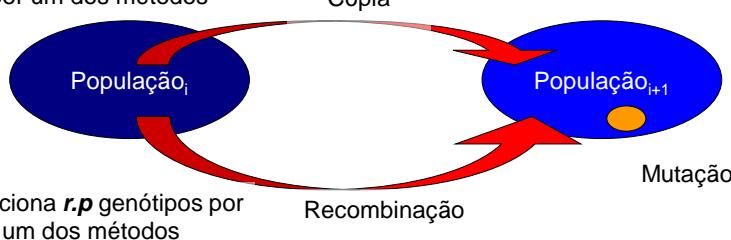
$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^p Fitness(h_j)}$$
 2. **Crossover:** Probabilistically select $\frac{r}{2}p$ pairs of hypotheses from P . For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s .
 3. **Mutate:** Invert a randomly selected bit in $m \cdot p$ random members of P_s
 4. **Update:** $P \leftarrow P_s$
 5. **Evaluate:** for each h in P , compute $Fitness(h)$
- Return the hypothesis from P that has the highest fitness.

Fucionamento

8

□ ...

Seleciona $(1-r).p$ genótipos por um dos métodos



Fucionamento

9

□ ...

■ 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.

Fucionamento

10

□ ...

■ Selecção

- Com base na função fitness, há vários métodos de selecção das hipóteses a incluir na próxima geração:
 - $(1-r).p$ hipóteses são copiadas
 - $r.p$ hipóteses são sujeitas a recombinação produzindo outros tantos descendentes

Selecção

11

□ Selecção Proporcional

- A probabilidade de selecção de uma hipótese é proporcional ao quociente q entre a sua aptidão e a soma das aptidões das restantes
- As hipóteses com maior valor de q são seleccionadas mais vezes)

Selecção

12

□ ...

- Exemplo: a solução de maior fitness foi seleccionada duas vezes!

Working Sheet of a GA

The problem is to optimize $f(x) = x$

String Number	Before Crossover (generation 0)				After Fitness Prop. Selection		After Crossover (generation 1)		
	i	String X_i	Fitness $f(X_i)$	$\frac{f(X_i)}{\sum f(X_j)}$	String X_i	Fitness $f(X_i)$	Cross-Point	X_i	$f(X_i)$
1	011	3	0.25	0.25	011	3	2	111	7
2	001	1	0.08	0.08	110	6	2	010	2
3	110	6	0.50	0.50	110	6	-	110	6
4	010	2	0.17	0.17	010	2	-	010	2
Total		12				17			17
Worst		1				2			2
Average		3.00				4.25			4.25
Best		6				6			7

Creating generation 1 from generation 0 by application of selection and crossover

Selecção

13

□ Implementação da Selecção Proporcional

■ Método da Roleta

- Cada hipótese de uma dada população possui uma fitness f_i
 - $f_1=1/6, f_2=1/3, f_3=1, f_4=1/2$
- Calculam-se os valores acumulados:
 - $A=f_1=1/6; B=f_1+f_2=1/2; C=f_1+f_2+f_3=3/2;$
 $D=f_1+f_2+f_3+f_4=2$
- Normalizam-se este valores
 - $A=0.0833, B=0.25, C=0.75, D=1$

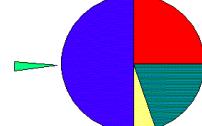
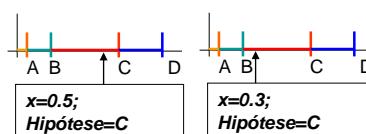
14

Selecção

□ ...

■ Gera-se um número aleatório x entre 0 e 1 e verifica-se sobre qual das hipóteses ele “cai”

- Como qualquer x é igualmente provável, ele cairá mais vezes sobre a zona correspondente à hipótese que ocupa maior espaço na recta
- Exemplos



Selecção

15

□ ...

- Seleção por Torneio:
 - Selecionar k hipóteses (Tamanho do Torneio) de entre a população.
 - De entre elas, selecionar a de maior fitness.
 - Duas hipóteses são selecionadas aleatoriamente de entre a população.
 - Com uma probabilidade pré-definida p , a de maior aptidão é selecionada (a outra é selecionada com probabilidade $(1-p)$).

Selecção

16

□ ...

- Seleção por Posicionamento (Ranking Selection)
 - As hipóteses são ordenadas de acordo com a sua aptidão, da melhor para a pior.
 - O **valor do ranking** (posição depois da ordenação) é usado (em vez da aptidão) por uma função que determina a probabilidade de seleção da hipótese (o espaço que ocupará na roleta)

Seleção

17



■ Exemplo:

- Escolher um número k entre 0 e 1: Seja k=0.6
- O indivíduo de maior aptidão, ID1, ocupará 60% da área da roleta
- D2 ocupará 60% da área restante: $(1-0.6)*0.6=24\%$
- ID3 ocupará 60% da área restante: $(1-0.6-0.24)*0.6=0.16*0.6=9.6\%$
- ...
- IDn ocupará a área restante

Recombinação

18

□ Operadores de recombinação

- As hipóteses são, muitas vezes, representadas por strings, o que permite uma implementação simples das operações de recombinação e mutação

■ Exemplos

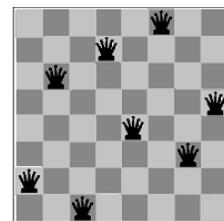
- Problema da mochila
 - “1” = objecto na mochila, “0” = objecto fora da mochila)
- 8 Rainhas
 - Cada hipótese é um “estado do tabuleiro”, representado por uma string do tipo “q1q2q3q4q5q6q7q8”
 - a hipótese “62714053” é uma solução “Rainha 1 = C1,L6... Rainha 8 = C8, L3”

Recombinação

19

... □

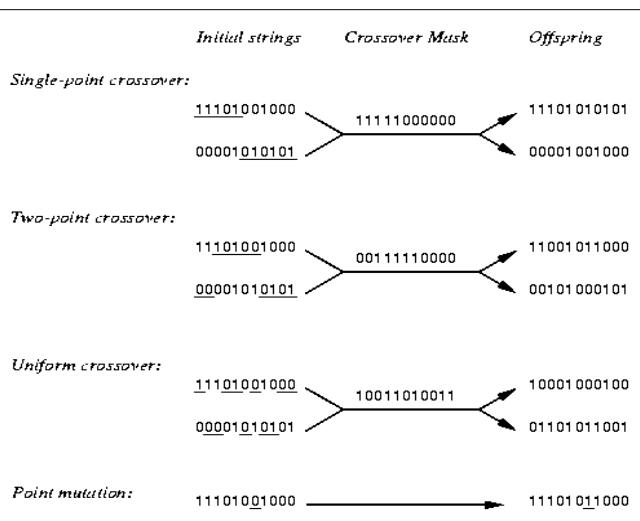
- A string “62714053” representa o genótipo (alusão ao material genético que caracteriza cada indivíduo)
- A configuração do tabuleiro representa o fenótipo (a tradução no mundo real daquilo que o genótipo determina)



Recombinação e Mutação

20

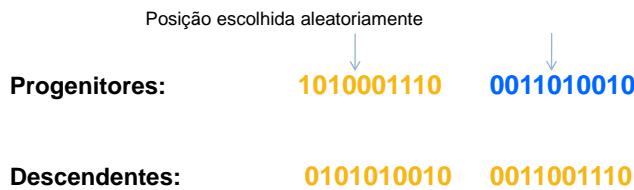
□ ...



Recombinação

21

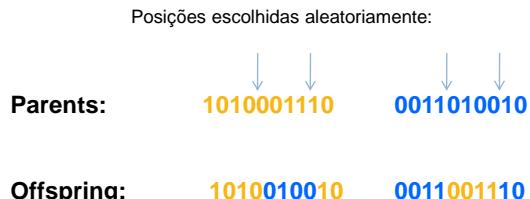
- ...
- Single-Point Crossover



22

Recombinação

- ...
- Double-Point Crossover



Recombinação

23

- ...
- Uniform Crossover

Máscara: 0110011000 (Gerada aleatoriamente)

Progenitores: 1010001110 0011010010

Descendentes: 0011001010 1010010110

Mutação

24

- Operadores de Mutação

Gera posição aleatória



Progenitor: 1010001110

Descendente: 1010101110

Problema do Caixeiro Viajante

25

□ TSP

- Neste caso, são necessários novos operadores.
- Para a recombição é importante não perder informação, tais como:
 - Ordem pela qual as cidades são visitadas
 - Adjacência entre cidades
 - Posição absoluta das cidades na sequência

Problema do Caixeiro Viajante

26

□ ...

- Recombinação por ordem
 - Considerem-se os progenitores P1 e P2
 - O descendente D1 é criado da seguinte forma:
 - Selecionar dois pontos de corte C1 e C2 ($C2 > C1$)
 - Copiar secção entre C1 e C2 de P1 para D1
 - Com início em C2, copiar as cidades de P2 para D1, omitindo as que já se encontram na sequência
 - O Descendente D2 é criado de forma análoga

Problema do Caixeiro Viajante

27

□ ...

P1: **123456789**
 P2: **248139576**

■ D1 **4567**
 ... **456728**
 ... **139 456728**

Problema do Caixeiro Viajante

28

□ ...

- Operadores de Mutação
 - Inserção 
 - Troca 
 - Inversão 

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

20-21

CAP. 8 ALGORITMOS PARA JOGOS

Carlos Pereira
ISEC

Índice

2

- Índice
 - Introdução
 - MiniMax
 - Alpha-Beta Pruning
 - Outras abordagens
 - Funções de Avaliação
 - Jogos com Elemento Sorte

Introdução

3

□ 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 funcional! Pois é necessário contar com os movimentos de B.

Introdução

□ ...

□ Tipos de jogos

	Determinístico <i>(factor sorte)</i>	Não Determinístico <i>(factor sorte)</i>
<i>Observável</i>	Xadrez Damas ...	Monopólio Gamão ...
<i>Parcialmente Observável</i>	Batalha Naval ...	Cartas ...

- O Minimax aplica-se a jogos determinísticos e observáveis

Introdução

5

□ ...

- Os jogos constituem problemas complexos e de difícil resolução.
- O exemplo mais conhecido é o **jogo de xadrez**:
 - Factor de ramificação $b \approx 35$
 - Em média desenrola-se ao longo de 50 lances por jogador.
 - A árvore de pesquisa (completa) teria **35^{100} nós!**
- Mais exemplos...
 - <http://aigamedev.com/>



Introdução

6

□ ...

- O HITECH foi o primeiro sistema a vencer um mestre de xadrez.
 - gera cerca de 10 milhões de estados antes de decidir um movimento.
- O Deep Thought 2 foi implementado pela IBM em parceria com Carnegie Mellon University (CMU).
 - Situava-se entre os 100 melhores jogadores humanos.
- Deep Blue (IBM) gera cerca de 100 a 200 biliões de posições por movimento.
 - <http://www.research.ibm.com/deepblue/>
 - Em 1997, venceu o Kasparov.



Introdução

□ ...

- X3D Deep Fritz
 - Em 2007 venceu o campeão do mundo Vladimir Kramik
 - 2 milhões de movimentos por segundo!
 - Ambiente Multi-processador

■ <http://www.chessbase.com/>



Introdução

□ Jogo de Damas

- 1950, Strachey, M.A., National Research Development Corporation,
- 1956 Arthur Samuel, IBM
- 1990, Chinook
 - Derrotou o campeão mundial
 - <http://webdocs.cs.ualberta.ca/~chinook/>
 - “Checkers is solved”
 - que jogo sem erros conduz a empate!
 - <http://www.sciencemag.org/content/317/5844/1518.abstract?keytype=ref&siteid=sci&ijkey=VmVcXy2%2FNTnY>



Introdução

□ Go

- Características do jogo
 - Inventado na China, 2000 AC
 - Tabuleiro 19×19 , alternadamente, os jogadores vão colocando peças nas intersecções
 - Objectivo:
 - Cercar as peças do oponente



□ Ainda mais difícil que Xadrez!

- Avaliação de posições é difícil e número de jogadas possivel é muito elevado

□ MoGo

- 25 nós, 4 cpus por nó (800 cores) a 4.7 GHz – total de 15 Teraflops
 - <http://www.iri.fr/~teytaud/mogo.html>
- Ganhou um jogo a Catalin Taranu (nº 5 do ranking)



MinMax

10

□ 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

■ Por exemplo no xadrez a avaliação pode ser +1 (MAX ganha), -1 (MAX perde) ou 0 (empatam). Noutros jogos a avaliação pode ser traduzida por pontos (cartas, etc..)

- Baseia-se no princípio de “selecção da melhor jogada por parte de cada jogador”

Minimax

11

□ ...

■ Definição formal do jogo

- Estado Inicial: Posição inicial, valor das “peças” e indicação de quem inicia o jogo.
- Operadores: Funções que definem as jogadas permitidas
- Teste de Final: Função que determina se o jogo acabou.
 - Os estados atingidos no final do jogo designam-se por Estados Terminais
- Função de Utilidade: mede o “proveito” que o estado terminal alcançado representa para cada um dos jogadores.

Minimax

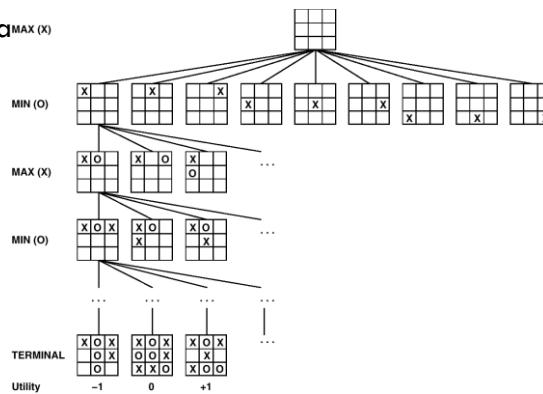
12

□ Como resolver o problema?

■ <https://scratch.mit.edu/projects/133865314/>

■ Construção da Árvore do jogo.

- Exemplo para $\text{MAX}(X)$



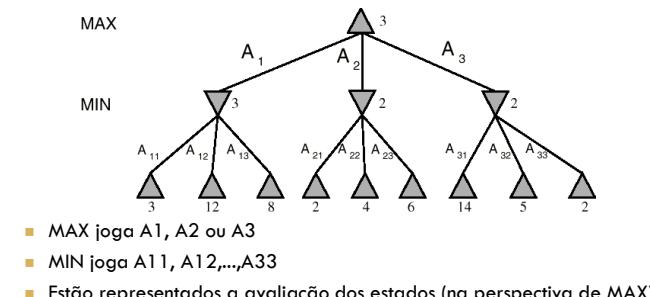
Minimax

13



- Considere-se um jogo que termina ao fim de duas meias-acções - Uma para MAX e outra para MIN.

- A árvore correspondente poderá ser:



- MAX joga A_1 , A_2 ou A_3
- MIN joga A_{11} , A_{12} , ..., A_{33}
- Estão representados a avaliação dos estados (na perspectiva de MAX)

- Qual a melhor estratégia?

Minimax

14



- MAX, para decidir se joga A_1 , A_2 ou A_3 , deve conhecer previamente os respectivos valores, o que implica:
 - Determinar os valores de todos os estados terminais
 - Partir do princípio de que MIN jogará de forma a prejudicar MAX
- Neste caso, qual a melhor jogada?

Minimax

15



1. Gerar a árvore do jogo
2. Determinar a Utilidade de cada estado terminal (valor para MAX)
3. Progredir para o nível anterior (neste nível é MIN que joga)

A cada nó assinalar o valor mínimo dos nós seus filhos (isto traduz que MAX espera que MIN jogue de modo a minimizar a pontuação de MAX)

4. Progredir para o nível anterior (neste nível é MAX que joga):
A cada nó assinalar o valor máximo dos nós seus filhos (isto traduz que MAX jogará da melhor forma)
5. Prosseguir assim até ser atingida a raiz da árvore.

Minimax

16



■ Algoritmo (de Decisão) Minimax:

```

function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← -∞
  for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
  return v

```

Minimax

17

□ ...

- 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
 - Porquê?

Alpha-Beta Pruning

18

□ Recursos: Tempo e Memória

- O algoritmo Minimax necessita de memória e tempo consideráveis, mesmo para jogos relativamente simples!
- Será viável aplicar o Minimax ao jogo de xadrez?
 - Num torneio de xadrez, cada jogada demora, em média, 150 segundos.
 - Com recursos razoáveis, admitamos que será possível pesquisar cerca de 1000 posições por segundo, o que implica 150.000 posições por jogada.
 - Assim, o programa só teria tempo para avaliar 4 meias-jogadas, o que corresponde a um nível de principiante! (com um factor de ramificação de 35, $35^4=1.500.625$)

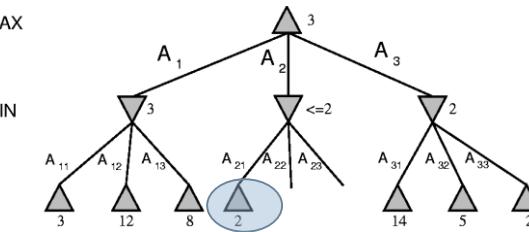
Alpha-Beta Pruning

19



- Será possível obter a decisão correcta sem ter de gerar toda a árvore?

MAX



- Se jogar A1 obtém 3. Se jogar A2 obtém 2 ou inferior
 - A valorização de A22 e A23 deixou de interessar!!
 - Considerando que o algoritmo é recursivo, a avaliação de todos os possíveis descendentes de A22 e A23 deixa também de ser necessária.

20

Alpha-Beta Pruning



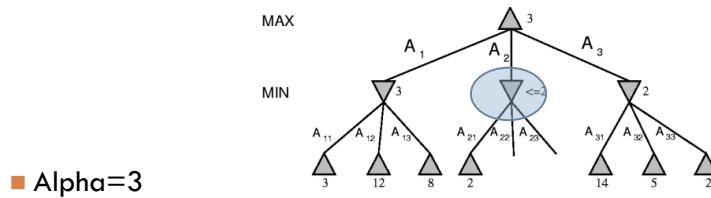
- ...
- O algoritmo baseia-se na utilização de dois parâmetros, “Alpha” e “Beta”:

- Alpha: representa o valor mínimo garantido que MAX poderá obter.
 - Como representa um limite inferior é inicializado a $-\infty$ 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 $+\infty$ e posteriormente vai decrescendo (actualizado num nó MIN)

Alpha-Beta Pruning

21

□ ...



■ Alpha=3

- provém da visita já realizada ao ramo esquerdo da árvore, que provou que MAX, optando por A1, pode obter o valor 3 (mesmo que MIN jogue o melhor possível)

■ Beta=2

- provém da análise dos filhos do nó atingível por A2, que prova que MIN pode impor um limite de 2 (ou eventualmente inferior, dependendo de A22 e A23)

Alpha-Beta Pruning

22

□ ...

```

function ALPHA-BETA-DECISION(state) returns an action
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    same as MAX-VALUE but with roles of α, β reversed

```

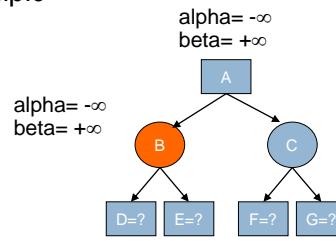
Alpha-Beta Pruning

23

□ ...

- Ao atingir-se um nó em que $\alpha \geq \beta$, pode cortar-se o ramo

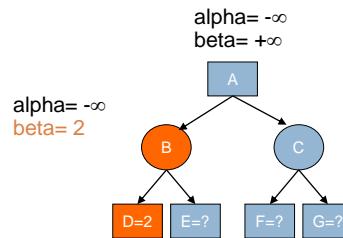
- Exemplo



Alpha-Beta Pruning

24

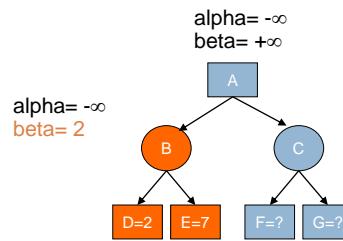
□ ...



Alpha-Beta Pruning

25

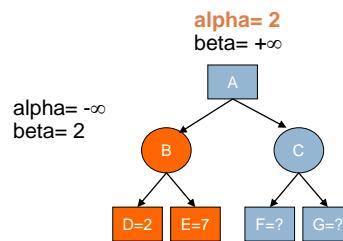
□ ...



Alpha-Beta Pruning

26

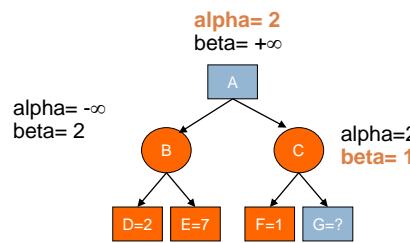
□ ...



Alpha-Beta Pruning

27

□ ...



- **O ramo G pode ser excluído – processo de pruning**

- porque qualquer que seja o seu valor, MAX nunca optará pelo ramo G!

Alpha-Beta Pruning

□ ...

- **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

Funções de Avaliação

□ Corte e Funções de Avaliação

- Outra forma de evitar a expansão completa da árvore de jogo.

Consiste no seguinte processo:

- Expande a árvore até um limite pré-determinado
- Avalia cada uma das folhas (usando uma heurística – função de avaliação)
- Proceder como no MiniMax
 - os valores dados pelas funções de avaliação são retornados como se as folhas correspondessem a estados terminais.

- Retorna uma estimativa, de natureza heurística, da função utilidade

■ Problema: Qual a heurística mais adequada?

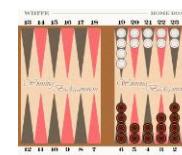
Jogos com Elemento Sorte

□ Elemento Sorte

- Muitos jogos contêm o elemento sorte.

■ Exemplo:

- Gamão: O jogador lança dois dados e a jogada é efetuada em função do resultado do lançamento.

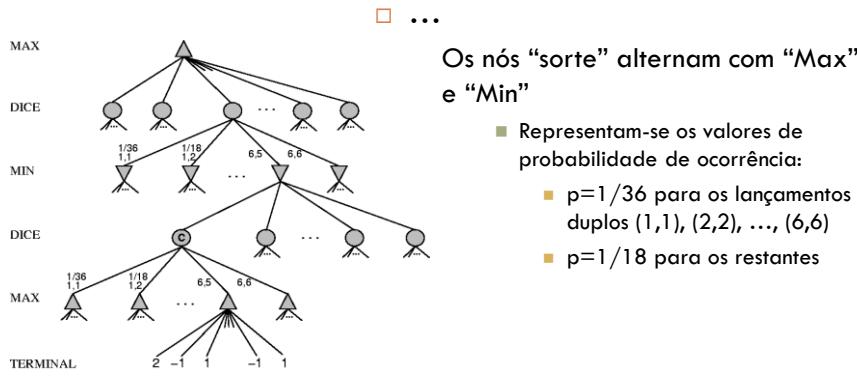


- Será possível aplicar o algoritmo Minimax também nestes jogos?

- Sim, contudo a árvore deverá incluir também nós que traduzam o fator sorte.



Jogos com Elemento Sorte



Jogos com Elemento Sorte

□ ...

■ Algoritmo:

- Calcula-se a utilidade nos estados terminais
- Nos nós superiores Max obtém-se o maior valor (como no Mimax)
- Nos nós superiores Min obtém-se o menor valor (como no Mimax)
- Nos nós sorte, calcular o valor esperado:

$$\text{para Max : } E = \sum_i P(d_i) \max(\text{utilidad}(s))$$

$$\text{para Min : } E = \sum_i P(d_i) \min(\text{utilidad}(s))$$

■ Exemplo - Nô C:

$$\begin{aligned} E &= 1/36 * x(1,1) + 1/18 * x(1,1) + \dots + 1/18 * \max(2, -1, \dots, 1) + \dots + \\ &\quad 1/18 * x(6,6) \end{aligned}$$

Jogos com Elemento Sorte

33

□ expectminimax

```

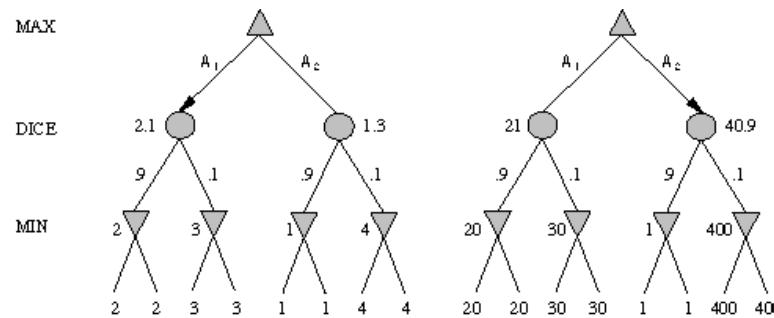
function expectiminimax(node, depth)
    if node is a terminal node or depth = 0
        return the heuristic value of node
    if the adversary is to play at node
        // Return value of minimum-valued child node
        let a := +∞
        foreach child of node
            a := min(a, expectiminimax(child, depth-1))
    else if we are to play at node
        // Return value of maximum-valued child node
        let a := -∞
        foreach child of node
            a := max(a, expectiminimax(child, depth-1))
    else if random event at node
        // Return weighted average of all child nodes' values
        let a := 0
        foreach child of node
            a := a + (Probability[child] * expectiminimax(child, depth-1))
    return a

```

Jogos com Elemento Sorte

34

□ ...



INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

20-21

CAP. 9 Aprendizagem com Redes
Neuronais

Carlos Pereira
ISEC

Índice

2

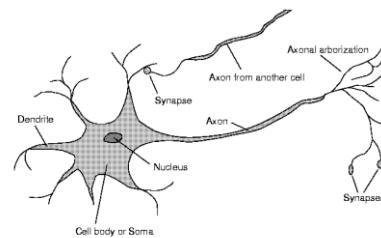
- Índice
 - Introdução
 - Aprendizagem
 - Perceptrão
 - Redes Lineares
 - Redes Multi-Camada
 - Aplicações

Introdução

3

□ Cérebro Humano

- 100 biliões de neurónios!
- Cada neurónio liga-se a outros 100.000;
- Tempo de resposta: 0.001 s
- Reconhecimento de padrões: 0.1s



Axónios

Linhos de Transmissão

Dendrites

Receptores

Synapses

Mediadores das interacções entre neurónios

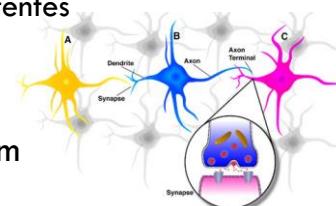
Introdução

4

□ Aprendizagem

□ Aprendizagem Automática

- Modificação das sinapses existentes
- Criação de novas ligações



□ Mecanismos de Aprendizagem

- Supervisionada
- Por Reforço
- Não Supervisionada

Introdução

5

□ 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.

Introdução

6

□ ...

- 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.

Aprendizagem

7



Sequência de eventos:

- ① A rede neuronal é “estimulada pelo meio”.
- ② A rede neuronal modifica-se, em resposta aos estímulos externos.
- ③ A rede neuronal vai produzir uma resposta diferente aos estímulos externos.

What we have to learn to do we learn by doing.

- Aristotle, Ethics

História

8

□ Os neurónios de McCulloch e Pits (1943)

□ McCulloch e Pits propuseram o que hoje em dia é considerado a primeira rede neuronal.

- Usaram o princípio de que a combinação de elementos computacionais simples era uma fonte de poder computacional acrescido.
- Cada neurónio implementa uma função lógica simples. Os pesos são calculados previamente. Não existia, nesta situação, um algoritmo de aprendizagem.
- Os neurónios eram organizados em rede, podendo produzir qualquer saída dada pela combinação de funções lógicas.
- Assume-se um intervalo temporal no fluxo de um sinal de um neurónio para outro. Permite assim a modelização de fenómenos fisiológicos como a percepção de calor ou frio.

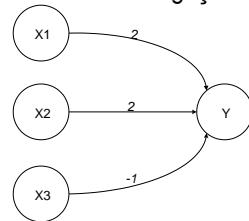
História

9

□ ...

■ Características

- A função de activação é binária. Os neurónios estão conectados entre si, e a cada ligação está associado um peso.
- Uma conexão pode induzir ou inibir uma resposta.
- A cada neurónio está associado um valor limite. Se a entrada ultrapassar esse limite o neurónio dispara.
- A cada ligação está associado um intervalo temporal



$$f(y_in) = \begin{cases} 1, & y_in \geq \theta \\ 0, & y_in < \theta \end{cases}$$

História

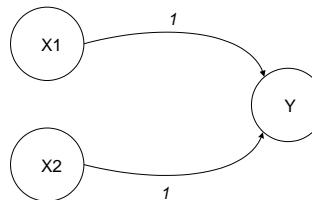
10

□ ...

■ Implementação de funções lógicas

■ AND

- (*Threshold=2*)

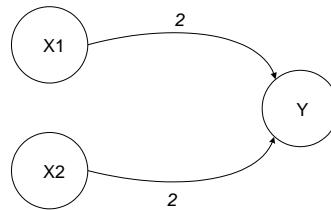


História

11

□ ...

■ OR

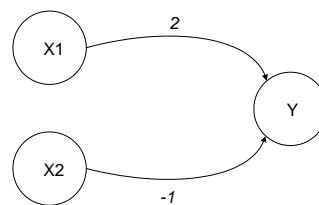


História

12

□ ...

■ AND NOT

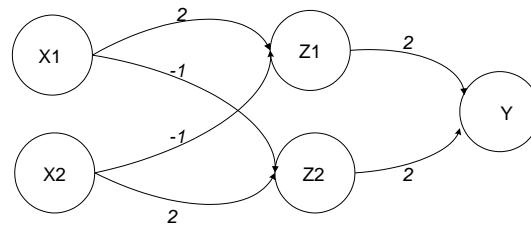


História

13

□ ...

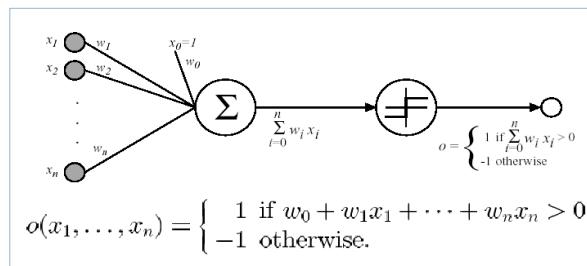
■ XOR



Perceptrão

14

□ Arquitetura

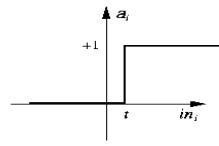


Perceptrão

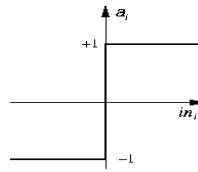
15

□ ...

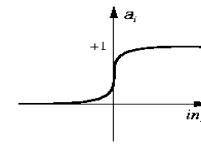
■ Funções de activação mais comuns:



(a) Step function



(b) Sign function

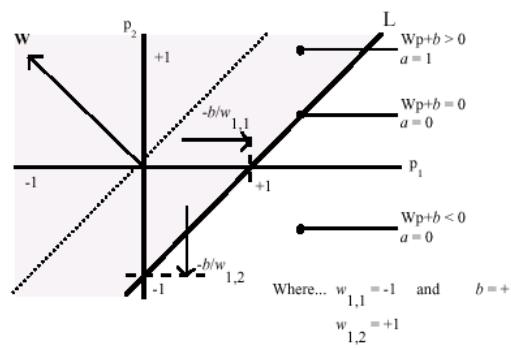


(c) Sigmoid function

Perceptrão

16

□ Superfície de Decisão

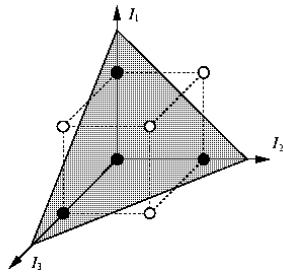


Perceptrão

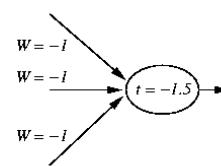
17

□ ...

■ O que podem representar?



(a) Separating plane



(b) Weights and threshold

Perceptrão

18

□ Algoritmo de treino

- Os coeficientes w_i são inicializados com valores aleatórios
- Por cada exemplo aplicado, os valores dos coeficientes são actualizados de acordo com:

$$\begin{aligned} w_i &\leftarrow w_i + \Delta w_i \\ \Delta w_i &= \eta(t - o)x_i \end{aligned}$$

- t = (target value) saída desejada
- o = (output) saída apresentada pelo perceptrão
- x_i = input i (entrada)
- η = Learning Rate: Constante a definir

Perceptrão

19

□ ...

■ Convergência do algoritmo

- Demonstra-se que existe convergência para os valores de coeficientes que classificam correctamente todos os exemplos, desde que:
 - Os exemplos de treino sejam Linearmente Separáveis
 - O coeficiente de aprendizagem seja suficientemente pequeno

Perceptrão

20

□ Exemplos

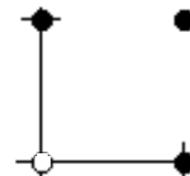
Função lógica “OR”

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, t_1 = 0 \right\}$$

$$\left\{ \mathbf{p}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, t_2 = 1 \right\}$$

$$\left\{ \mathbf{p}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, t_3 = 1 \right\}$$

$$\left\{ \mathbf{p}_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_4 = 1 \right\}$$

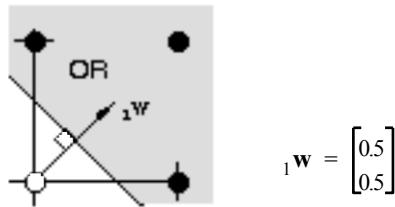


Perceptrão

21

□ ...

Uma solução:



$$_1\mathbf{w}^T \mathbf{p} + b = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} + b = 0.25 + b = 0 \quad \Rightarrow \quad b = -0.25$$

Perceptrão

22

□ ...

Conjunto de treino

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \end{bmatrix} \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = \begin{bmatrix} 0 \end{bmatrix} \right\}$$

Pesos Iniciais

$$\mathbf{W} = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} \quad b = 0.5$$

Perceptrão

23

□ ...

Iteração 1

$$a = \text{hardlim}(\mathbf{W}\mathbf{p}_1 + b) = \text{hardlim} \left(\begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0.5 \right)$$

$$a = \text{hardlim}(-0.5) = 0 \quad e = t_1 - a = 1 - 0 = 1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} + (1) \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 0.5 + (1) = 1.5$$

Perceptrão

24

□ ...

Iteração 2

$$a = \text{hardlim}(\mathbf{W}\mathbf{p}_2 + b) = \text{hardlim} \left(\begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + (1.5) \right)$$

$$a = \text{hardlim}(2.5) = 1$$

$$e = t_2 - a = 0 - 1 = -1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix} + (-1) \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -1.5 & -1 & -0.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 1.5 + (-1) = 0.5$$

Perceptrão

25

- ...

Teste:

$$a = \text{hardlim}(\mathbf{W}\mathbf{p}_1 + b) = \text{hardlim}([\begin{matrix} -1.5 & -1 & -0.5 \end{matrix}] \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0.5)$$

$$a = \text{hardlim}(1.5) = 1 = t_1$$

$$a = \text{hardlim}(\mathbf{W}\mathbf{p}_2 + b) = \text{hardlim}([\begin{matrix} -1.5 & -1 & -0.5 \end{matrix}] \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + 0.5)$$

$$a = \text{hardlim}(-1.5) = 0 = t_2$$

Redes Lineares

26

- Perceptrão com Unidades Lineares
 - uma Unidade Linear consiste num “perceptrão sem função Sinal”, cuja saída é dada por:

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

- Algoritmo de Treino
 - Gradiente Descente

Redes Lineares

27

- ...
- GRADIENT-DESCENT(*training_examples*, η)
Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).
- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - * Input the instance \vec{x} to the unit and compute the output o
 - * For each linear unit weight w_i , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$
 - For each linear unit weight w_i , Do

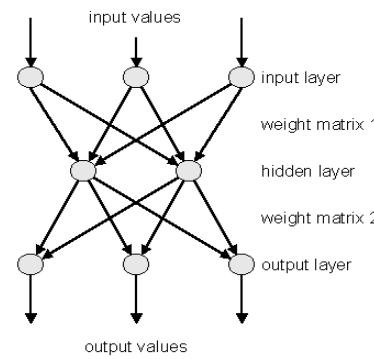
$$w_i \leftarrow w_i + \Delta w_i$$

Redes Multi-Camada

28

□ Arquitectura

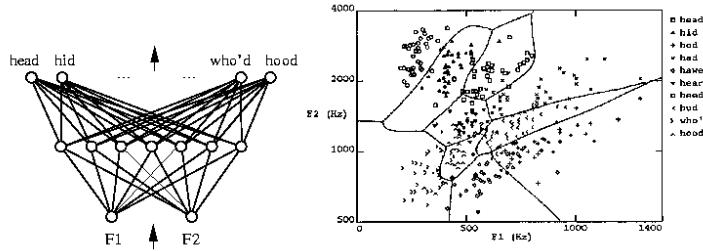
- As Redes Multi-Camada usam unidades não lineares, normalmente de função sigmoide, distribuídas por várias camadas interligadas entre si.
- podem representar superfícies de decisão de forma muito variada
- São treinadas pelo algoritmo “BackPropagation”



Redes Multi-Camada

29

□ ...



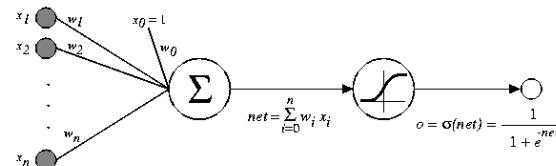
- Rede Neuronal para Distinção entre 10 vogais num contexto “h...d”
- Duas entradas para as frequências f1 e f2
- Uma camada interna

Redes Multi-Camada

30

□ ...

- Usam funções de activação não lineares
- Habitualmente a função sigmoidal



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Redes Multi-Camada

31

□ ...

■ Algoritmo

Backpropagation Algorithm

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
 1. Input the training example to the network
and compute the network outputs
 2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k} \delta_k$$

4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

Aplicações

32

□ As primeiras aplicações...

■ Reconhecimento de caracteres, Lecun, 1989

■ Reconhece códigos postais em envelopes escritos à mão.

Usa uma matriz de 16x16 de pixels, três camadas escondidas e 10 unidades de saída (0..9)

■ <http://yann.lecun.com/exdb/mnist/>

Aplicações

33

□ ...

- Como Reconhecer?

0 0 0 1 1 1 1 1 1 2
 2 2 2 2 2 2 3 3 3
 3 4 4 4 4 4 5 5 5
 6 6 7 7 7 7 8 8 8
 8 8 8 8 9 9 9 9

Aplicações

34

□ ...

- ALVINN (Autonomous Land Vehicle In a Neural Network) (1993).
- Aprende a conduzir um veículo em diferentes condições, a partir de um exemplo de condução humana.

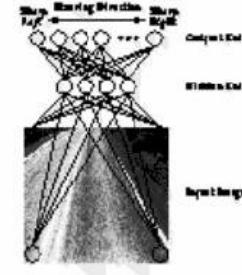


Aplicações

35



- ...
- Aprende após 3 minutos de observação de um vídeo.
- A performance é suficiente para não sair da estrada. Não faz ultrapassagens!
- Usa duas câmaras.



Aplicações

36

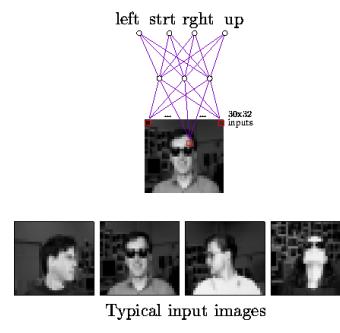
□ Reconhecimento de padrões

- Rede de $960 \times 3 \times 4$ treinada com 260 imagens

- Objectivo: Classificar a posição da face: Olhar à esquerda, direita, cima baixo

- Resultado: 90% de acertos num conjunto de imagens distinto dos exemplos de treino

Neural Nets for Face Recognition



Aplicações

37

□ Jogos

▫ Blondie24

- Jogo de damas
- Baseia-se no algoritmo MiniMax e usa uma **rede neuronal** para implementar a função de avaliação.
- Rede neuronal MLP
 - Recebe: estado do jogo (posições das peças no tabuleiro)
 - Devolve: o valor de utilidade do estado
 - Treino: Os pesos da rede determinados por um algoritmo genético:
 - Um indivíduo corresponde a um programa Blondie24; Jogando entre eles, os melhores são selecionados.

Aplicações

38

□ ...

▫ AlphaGo

- Primeiro programa a derrotar um jogador profissional de Go – humano.
 - Primeiro a derrotar o campeão do Mundo de Go.
 - <https://deepmind.com/research/case-studies/alphago-the-story-so-far>
 - Combina métodos de pesquisa – “Monte Carlo tree search”, com aprendizagem automática – “redes neurais”
 - As redes neurais são treinadas para identificar os melhores movimentos – percentagem de vitórias.

▫ AlphaZero

- Aprende sozinho, sem conhecimento pericial.
 - Aprendizagem com reforço
 - Também aplicado a outros jogos (damas, shogi)