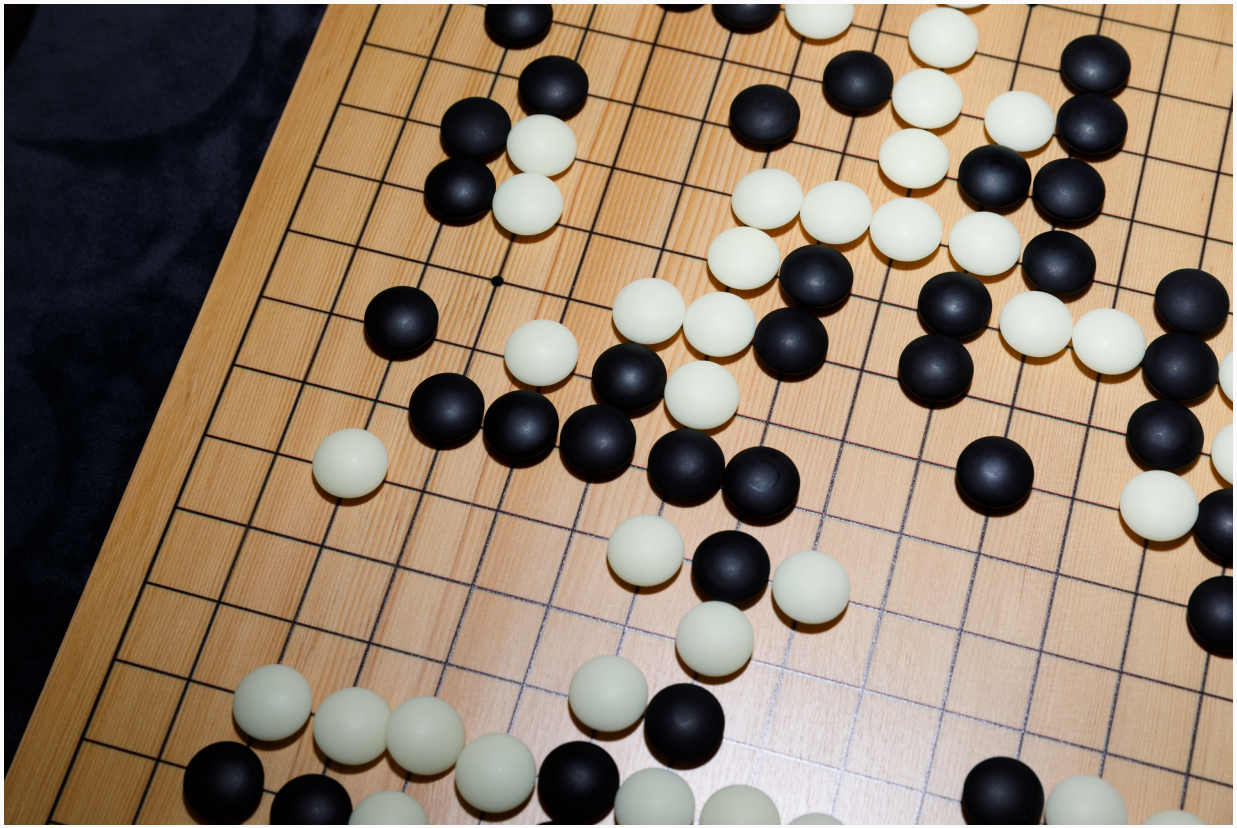
ACH2016 - Inteligência Artificial
Aula 11 - Busca Informada

Valdinei Freire da Silva

valdinei.freire@usp.br - Bloco A1 100-O

Russell e Norvig, Capítulo 3

AlphaGo



- Solução: busca aleatória heurística + aprendizado por reforço + redes neurais
- Documentário: AlphaGo

Resolução de Problemas

Ambientes: completamente observável, único agente, conhecido, determinístico, discreto, sequencial, estático.

Formulação de Problemas

- estado inicial: é o estado no tempo $t = 0$.
- ações possíveis: a função $\mathrm{ACTIONS}(s)$ retorna o conjunto de ações que podem ser executadas no estado s.
- modelo de transição: a função $\mathrm{RESULT}(s, a)$ retorna o estado resultante de aplicar a ação a no estado s.
- teste de meta: a função $\mathrm{GOAL}(s)$ determina se o estado s é um estado meta.
- custo de caminho: a função $\mathrm{PATHCOST}(h)$ retorna o custo associado a uma sequência $h = s_0, a_0, \ldots, s_{T-1}, a_{T-1}, s_T$.

Busca por Solução

função Busca(problema) **retorna** uma solução, ou falha
    inicializar a fronteira usando o estado inicial do problema
    inicializar o conjunto explorado como vazio
    repita
        se fronteira vazia
            então retorna falha
        escolher um nó folha e remover da fronteira
        se o estado do nó for um estado objetivo
            então retorna solução correspondente
        adicionar o nó ao conjunto explorado
        expandir o nó escolhido, encontrando os próximos nós
        para cada próximo nó
            se não estiver na fronteira nem no conjunto explorado
                então adiciona à fronteira

Solução Ótima Parcial

Cada caminho h é avaliado pela função $\mathrm{PATHCOST}(h)$.

Considere um caminho parcial que passa pelo nó n na árvore de expansão, se existir, pode-se encontrar o caminho com menor custo que utiliza o nó n e chegue no estado meta.

Considere o valor de tal caminho como sendo $V^*(n)$, então o melhor nó a ser expandido é o nó com menor valor de $V^*(n)$.

Se a função $\mathrm{PATHCOST}(h)$ consiste em somatória de custos locais, então tem-se que:

$$V^*(n) = g(n) + h^*(n),$$

onde $g(n)$ é o custo acumulado até o nó n e $h^*(n)$ é o custo do caminho ótimo entre o estado representado pelo nó n e um estado meta.

Busca Informada

Busca informada ou heurística: estima qual o melhor nó da fronteira a ser expandido com base em funções heurísticas. Sabem se um estado não objetivo é "mais promissor".

Se tivermos uma estimativa $f(n)$ para $V^*(n)$, pode-se utilizar essa informação para construir novos algoritmos.

Uma heurística consiste em uma função heurística $h(n)$ que estima $h^*(n)$, e $h(n)$ estima o custo do caminho de menor custo do estado no nó n para um estado meta.

- Seja n e n' dois nós com o mesmo estado (quando permite-se inserir nós repetidos), então $h(n) = h(n')$.
- Seja n um nó com um estado meta, então $h(n) = 0$.

Busca Informada

Algoritmos:

- Busca Gulosa
- A*
- RBFS
- SMA*

Busca Gulosa (Greedy Best-first Search)

Expande o nó que está mais próximo da meta, segundo a heurística $h(n)$, isto é, considera uma fila de prioridade onde cada nó recebe $f(n) = h(n)$.

A cada passo tenta chegar o mais próximo possível da meta.

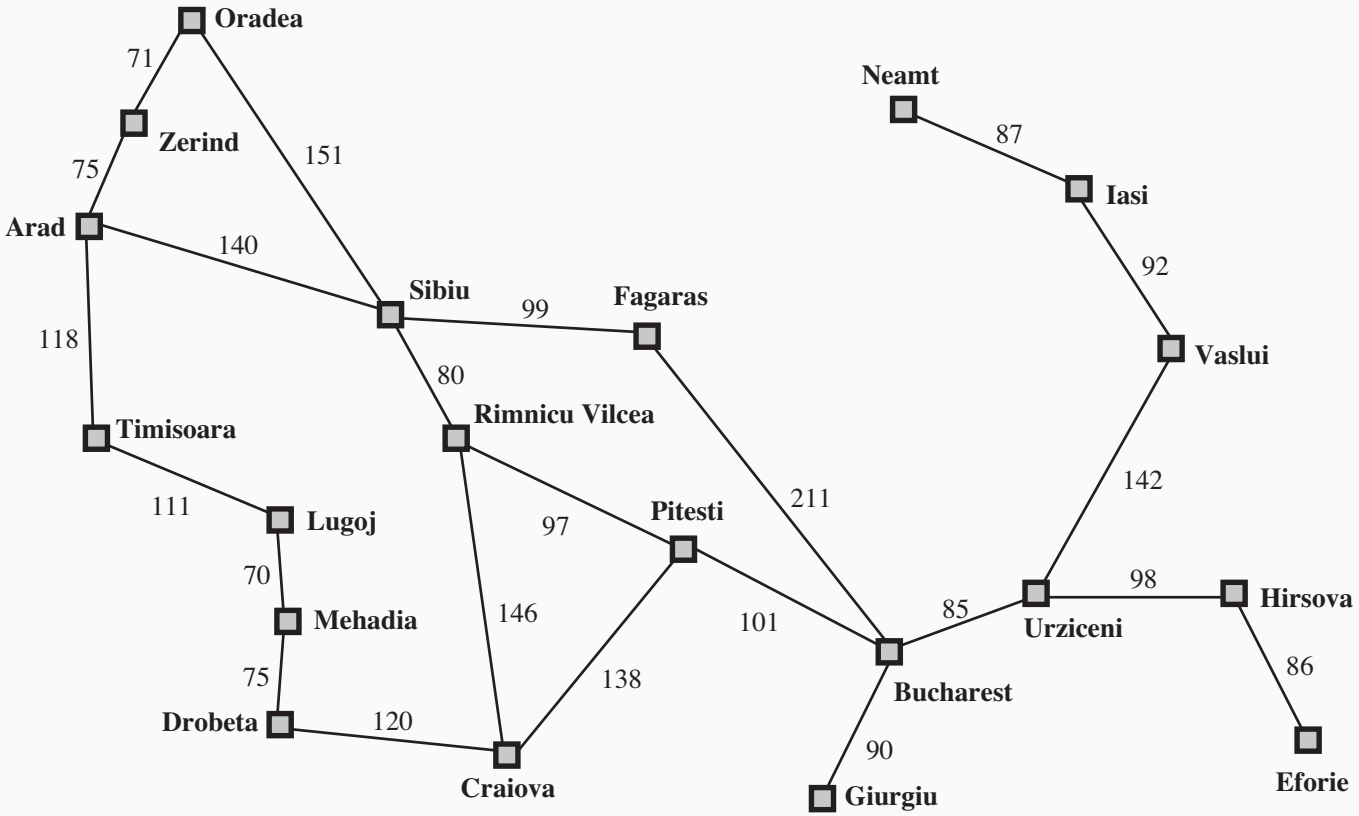
Completude: completo se os estados são finitos.

Otimalidade: garantida apenas se $h(n) = h^*(n)$.

Complexidade de tempo: no pior caso $O(b^m)$, mas, dependendo da qualidade da heurística pode ser bastante reduzida.

Complexidade de memória: no pior caso $O(b^m)$, mas, dependendo da qualidade da heurística pode ser bastante reduzida.

Exemplo: de Arad para Bucharest



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Busca A^* (A-estrela)

Considera uma fila de prioridade onde cada nó recebe $f(n) = g(n) + h(n)$.

- $g(n)$ é o custo do caminho na árvore do estado inicial até o estado do nó n.
- $h(n)$ estima o custo do caminho de menor custo do estado no nó n para um estado meta.
- $f(n)$ estima o custo do caminho de menor custo passando pelo nó n.

Busca A*

Condições para Otimalidade:

- **Heurística Admissível:** uma heurística admissível nunca superestima o custo para alcançar a meta, isto é, a heurística é otimista e $h(n) \leq h^*(n)$.
- **Heurística Consistente (monotonicidade):** uma heurística é consistente se, para todo nó n e todo sucessor n' de n, a estimativa $f(n)$ do custo do caminho passando por n é melhor que a estimativa $f(n')$ do custo do caminho passando por n', isto é, $h(n) \leq c(n, a, n') + h(n')$.
- Uma heurística consistente também é admissível.

Consistência é uma forma geral de desigualdade triangular, cada lado do triangulo não pode ser maior que a soma dos outros dois. O triângulo é formado por n, n' e o estado meta G_n mais próximo de n.

Busca A*

Quando se permite inserir nós repetidos, Busca A* é ótima se a heurística é admissível.

Quando não se permite inserir nós repetidos, Busca A* é ótima se a heurística é consistente.

1. Se $h(n)$ é consistente, então o valor $f(n)$ em qualquer caminho é não-decrescente.
2. Quando um nó n é selecionado para ser expandido, o caminho ótimo até aquele nó foi encontrado.
3. Então, segue que a sequência de nós expandidos está em ordem não-decrescente de $f(n)$ e portanto o primeiro nó com estado meta expandido deve ser uma solução ótima, já que $h(n) = 0$ quando n representa o estado meta.

Busca A*

Se C^* é o custo do caminho ótimo, então busca A* expande todos os nós n tal que $f(n) < C^*$.

Completude: requer que exista uma quantidade finita de nós com custo menor ou igual à C^*, que pode ser atendida se o fator de ramificação b é finito e se o custo local é maior que algum $\epsilon > 0$.

Complexidade: difícil calcular, mas a complexidade de memória aparece como uma questão prática antes da complexidade de tempo.

Busca A* com Limite de Memória

Iterative-deepening A* (IDA*): busca em profundidade limitada pelo custo, na qual em cada iteração o valor de limite é o menor valor de nó que não foi expandido na iteração anterior.

Recursive Best-Fisrt Search

Simplified Memory-Bounded A*

Recursive Best-Fisrt Search (RBFS)

Busca em profundidade recursiva, mas mantém uma variável f_limit que aponta para a melhor alternativa entre seus antecessores.

Se o nó corrente ultrapassa esse limite, ele retorna para o caminho alternativo.

Quando o algoritmo retorna, ele atualiza a função $f(n)$ de cada nó com o melhor valor de seus filhos.

Algoritmo RBFS

função RBFS(problema) **retorna** uma solução, ou falha

    retorna RBFS(problema, NODE(problema.estadoInicial),∞)

função RBFS(problema, nó, *f_limit*) **retorna** uma solução, ou falha e um novo custo limite *f*

    se problema.GOAL(nó) **então retorna** solução
    sucessores$\leftarrow$ { }
    para cada próximo_nó
        adiciona próximo_nó em sucessores
    se sucessores está vazio **então retorna** falha,∞
    repita
        best $\leftarrow$ o nó com menor valor de *f* em sucessores
        se *best*.*f* $>$ *f_limit* **então retorna** falha, *best*
        alternative $\leftarrow$ o segundo menor *f* em sucessores
        result, *best*.*f* $\leftarrow$ RBFS(problema, *best*, min(*f_limit*, *alternative*))
        se *result* $\neq$falha **então retorna** *result*

Simplified Memory-Bounded A* (SMA*)

Se comporta como A* até acabar a memória:

- Apaga o nó mais custoso em termos de $f(n)$
- Faz o back up do valor esquecido para o nó antecessor

Implementação:

- Se todo os nós possuem o mesmo valor, deleta o mais antigo
- Se o caminho ótimo possui mais nó que a memória, retorna uma solução sub-ótima.

Qualidade da Heurística

Dominância: se $h_2(n) \geq h_1(n)$ para todo n, então h_2 domina h_1.

Se ambas heurísticas são admissíveis, melhor escolher a heurística que domina a outra.

Problema Relaxado: um problema com menor restrições de ações que no problema original.

Resolve-se o problema relaxado e utiliza-se a solução obtida como heurística.

A solução obtida é consistente.

Qualidade da Heurística

7	2	4
5		6
8	3	1

estado inicial

	1	2
3	4	5
6	7	8

estado meta

Original: Uma peça pode se mover do quadrado A para o quadrado B se: A é adjacente a B **AND** B está vazio.

Opções de Relaxamento:

1. Uma peça pode se mover do quadrado A para o quadrado B se A é adjacente a B.
2. Uma peça pode se mover do quadrado A para o quadrado B se B está vazio.
3. Uma peça pode se mover do quadrado A para o quadrado B.

Qualidade da Heurística

h_1 = soma da distância Manhattan de cada peça entre a posição atual e o estado meta.

h_2 = quantidade de peças fora do lugar.

h_3 = quantidade de peças fora da linha + quantidade de peças fora da coluna

h_1 domina h_2 e h_3
h_3 domina h_2

Se nenhuma heurística domina a outra pode-se construir uma nova:

$$h(n) = \max h_1(n), \ldots, h_m(n)$$

Toy Problems - 8-puzzle

Custo da Busca

d	IDS	A*(h_1)	A*(h_2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113
22	–	18094	1219
24	–	39135	1641