Métodos de Busca Heurística

Prof. Júlio Cesar Nievola

PPGIa - PUCPR

Implementação do método de busca do melhor primeiro

Function BUSCA_MELHOR_PRIMEIRO (*problema*, AVAL-FN) **returns** uma sequência solução

inputs: *problema*, um problema

Aval-Fn, uma função de avaliação

Ordenar-Fn uma função que ordena nós segundo Aval-Fn

return BUSCA_GERAL(*problema*, *Ordenar-Fn*)

Busca do melhor primeiro

- Utiliza uma medida estimada do custo da solução e tenta sua minimização
- A medida deve incorporar uma estimativa do custo do caminho de um estado ao estado objetivo mais próximo
- Há duas abordagens:
  - Expandir o nó mais próximo a um objetivo;
  - Expandir o nó no caminho da solução de menor custo

Busca Gulosa ("Greedy search")

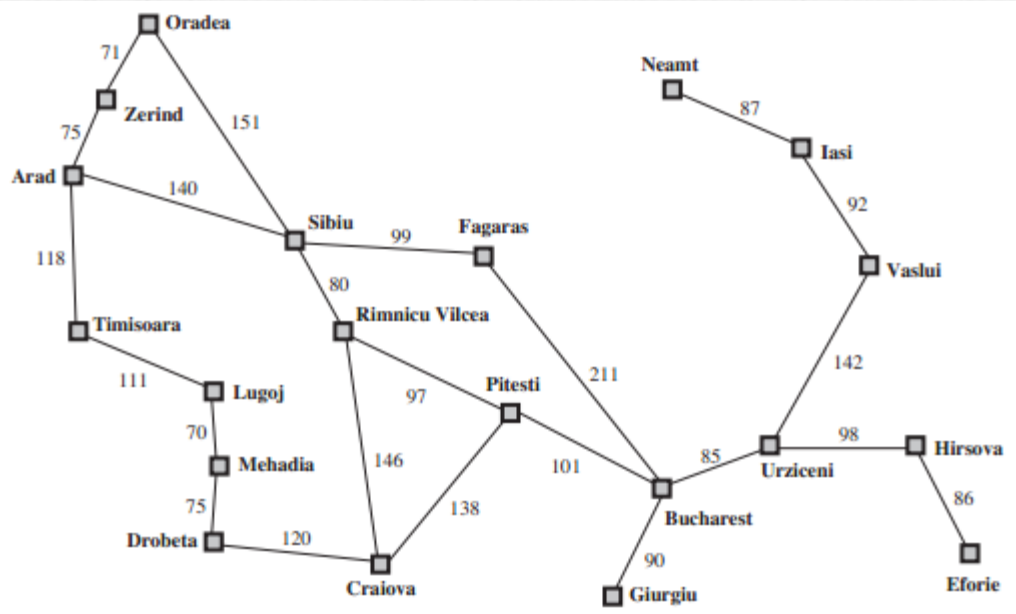
- A mais simples estratégia de busca do melhor primeiro consiste em minimizar o custo estimado para atingir o objetivo

$h(n) \equiv$ custo estimado do caminho mais "barato" do estado n até o estado objetivo

Function BUSCA_GULOSA(*problema*)
returns uma solução ou falha

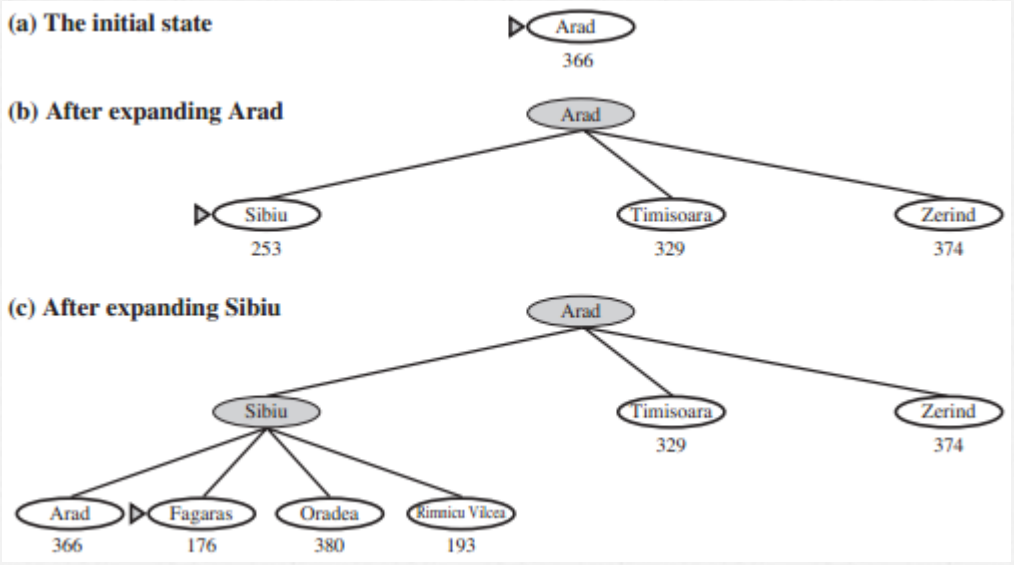
return BUSCA_MELHOR_PRIMEIRO(
problema, h)

Mapa da Romênia

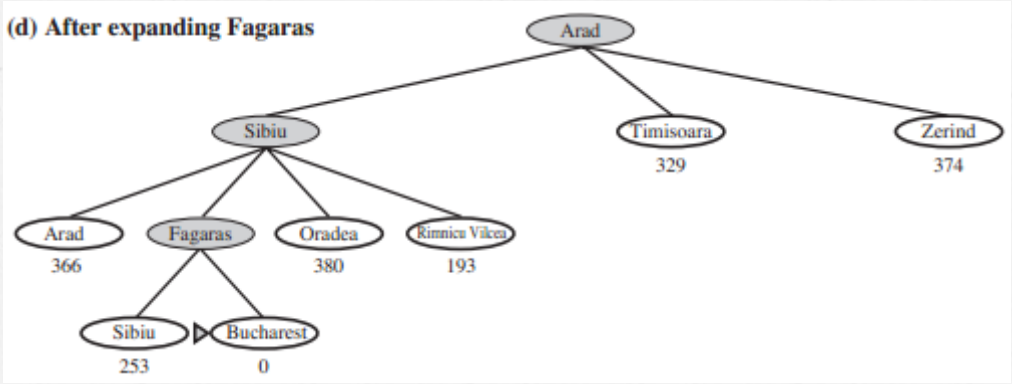


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Busca Gulosa para Bucareste - 1



Busca Gulosa para Bucareste - 2



Minimizando o custo total do caminho

- Busca Gulosa não é ótima nem completa
- Busca de custo uniforme (que usa *g(n)*) é ótima e completa mas é ineficiente
- Pode-se combinar as duas, usando-se *f(n)* como o custo estimado total da solução de menor custo passando por *n*:

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

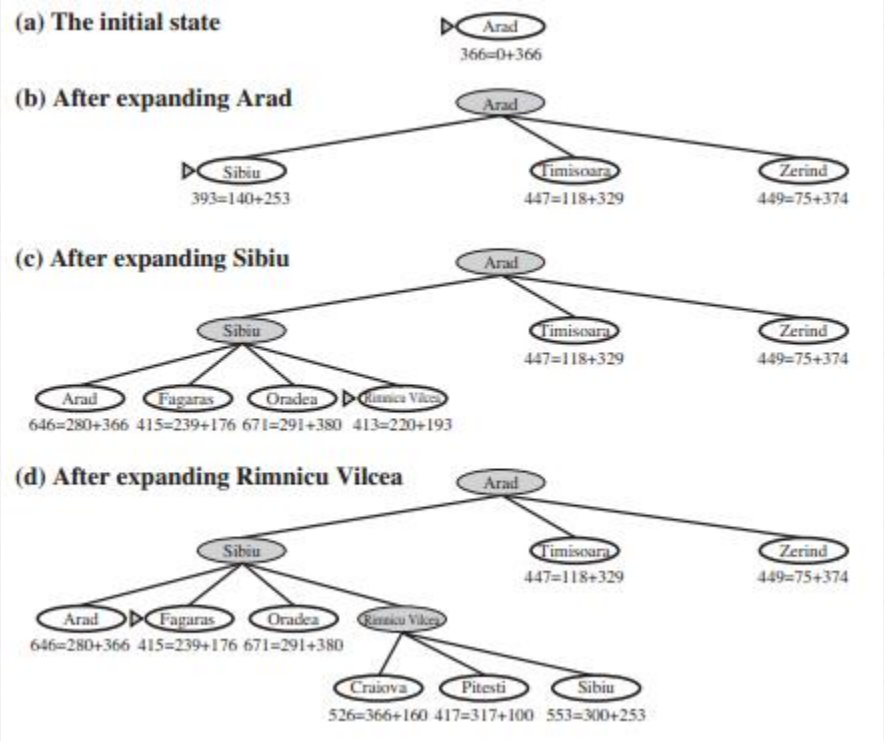
Estratégia A*

- Uma heurística é *admissível* se ela nunca superestima o custo de atingir o objetivo, ou seja, ela é otimista
- A estratégia A* faz uma busca do melhor primeiro, usando *f* como função de avaliação e uma função *h* admissível:

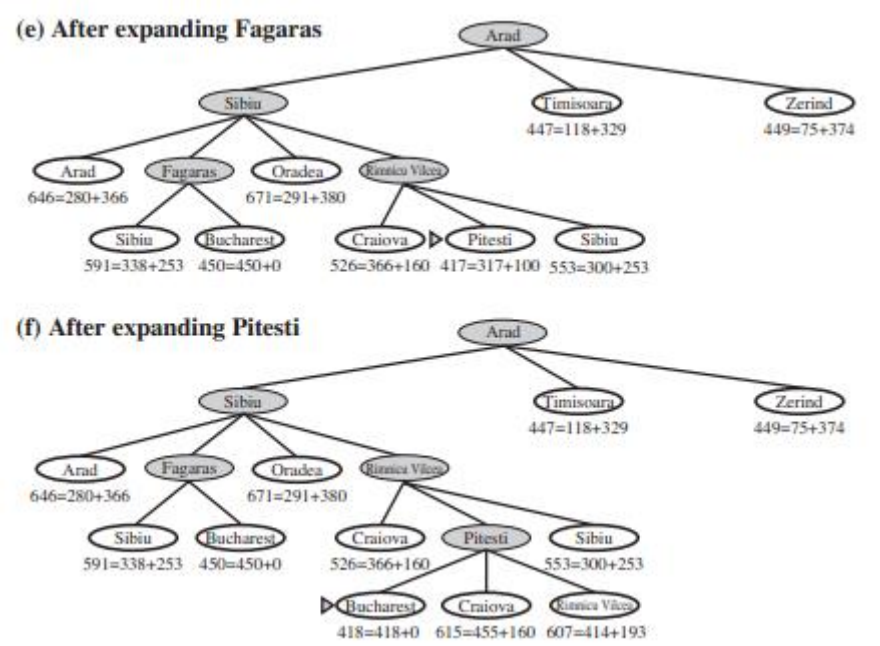
Function BUSCA- A*(*problema*) **returns** uma solução ou falha

return BUSCA_MELHOR_PRIMEIRO(*problema, g + h)*

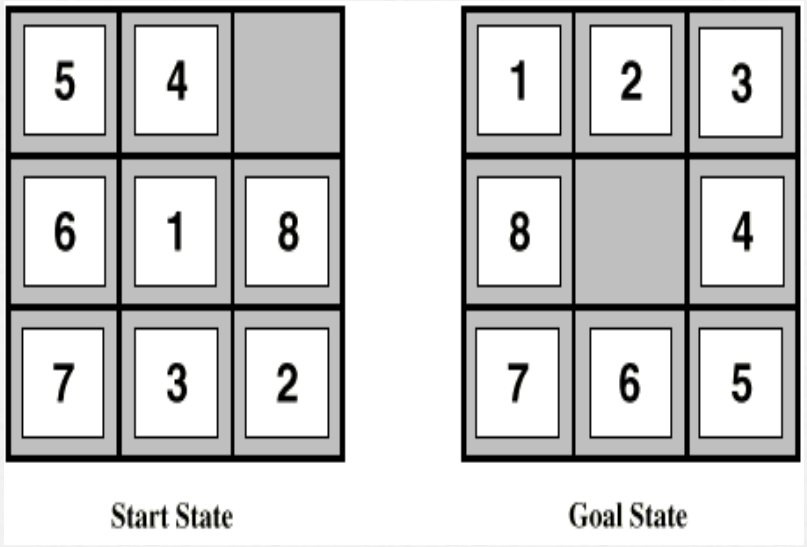
A* para a busca de Bucareste - 1



A* para a busca de Bucareste - 2



Funções Heurísticas

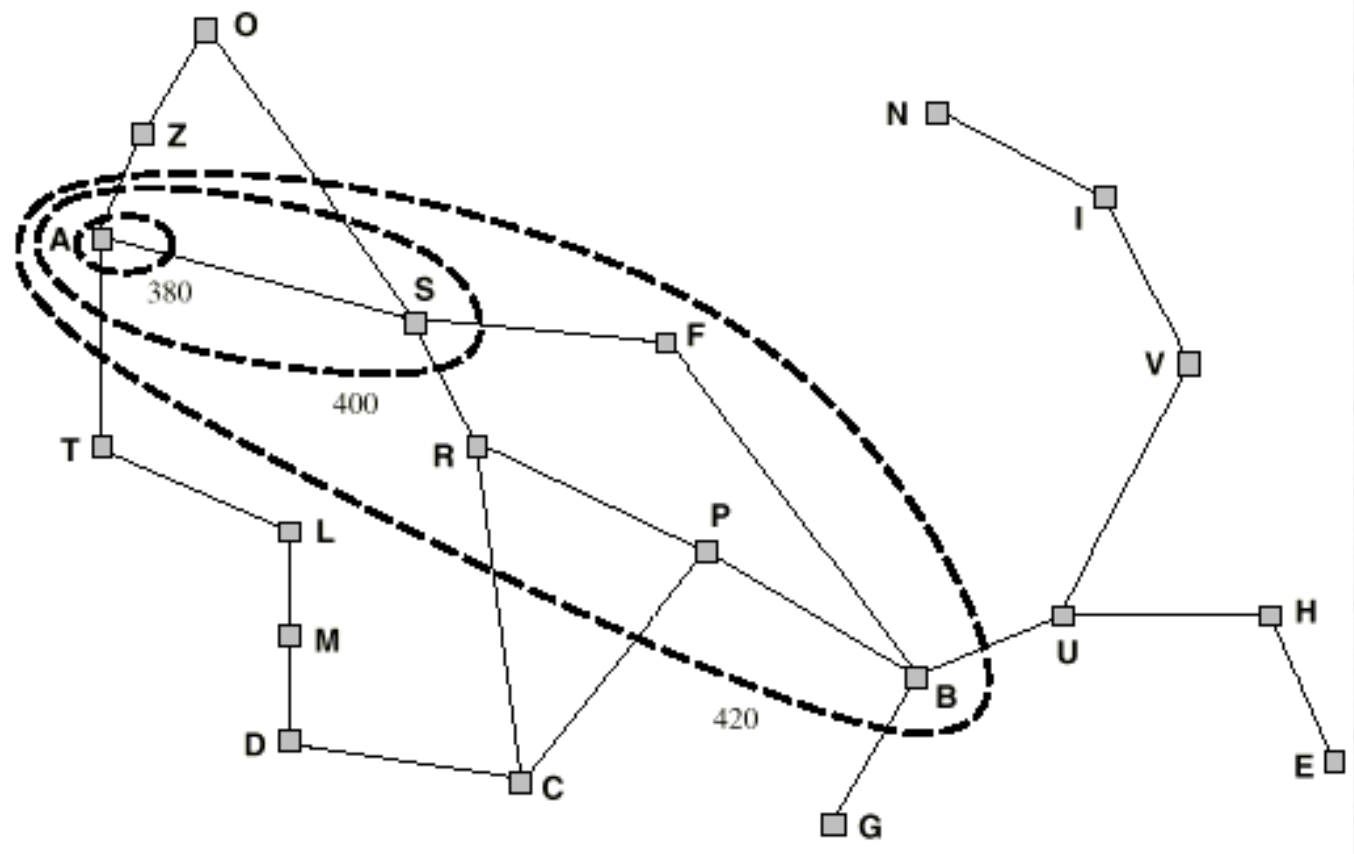


- h_1 quantidade de blocos na posição incorreta
- h_2 soma das distâncias de cada bloco à sua posição objetivo

Comparação entre A* com h_1 e h_2 com aprofundamento iterativo

	Search Cost			Effective Branching Factor		
d	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

Mapa da Romenia com contornos com $f = 380$, $f = 400$



Aprofundamento iterativo A*

- Com a finalidade de conservar memória pode-se usar a idéia do algoritmo de aprofundamento iterativo
- Transforma-se a estratégia de busca A* em aprofundamento iterativo usando-se a noção de contorno de mesmo custo
- Boa estratégia em domínios pequenos mas possui dificuldades em domínios grandes

IDA* - Algoritmo principal

Function IDA*(*problema*) **returns** uma sequência solução
  inputs: *problema*, um problema
  static: *f-limite*, o limite atual *f* –CUSTO, *raiz*, um nó
    raiz ← CRIAR_NÓ(ESTADO_INICIAL[*problema*])
    f-limite ← *f* -CUSTO(*raiz*)
    loop do
      solução, f-limite ← DFS-CONTORNO(*raiz, f-limite*)
      if *solução* é não-nula **then return** *solução*
      if *f-limite* = ∞ **then return** falha;
    end

IDA* - Busca no contorno

Function DFS-CONTORNO(*nó, f-limite*) **returns** uma sequência solução e um novo *f* -CUSTO limite

inputs: *nó*, um nó *f-limite*, o limite atual *f* -CUSTO

static: *próximo-f*, o *f* -COST limite para o próximo contorno, inicialmente ∞

if *f* -CUSTO[*nó*] > *f-limite* **then return** nulo, *f* -CUSTO[*nó*]

if TESTE_OBJETIVO[*problema*](ESTADO[nó]) **then return** *nó*, *f-limite*

for each nó *s* **in** SUCESSORES(*nó*) **do**

solução, *novo-f* ← DFS-CONTORNO(*s, f-limite*)

if *solução é* não-nulo **then return** *solução, f-limite*

próximo-f ← MÍN(*próximo-f, novo-f*)

end

return nulo, *próximo-f*