Busca com Informação

- Métodos de busca sem informação podem ser (são) bem ineficientes
- Se houver algum conhecimento maior sobre o problema que possa ser utilizado, a busca pode ser muito mais eficiente
- Uso de uma função de avaliação
  - o nodo a ser expandido é escolhido de acordo com uma função de avaliação *f(n)*
  - Expande-se o nodo com menor *f(n)*
  - Em geral, *f(n)* usa alguma heurística

Função Heurística

- De forma geral, a função heurística fornece uma indicação do melhor caminho para a solução
- *h(n)* é o custo **estimado** do melhor caminho de um nodo *n* até o gol
  - *h(gol)* = 0
- Exemplos
  - Romênia: distância direta entre cidades
  - 8-Puzzle: número de quadrados fora de posição

Busca Gulosa (Greedy)

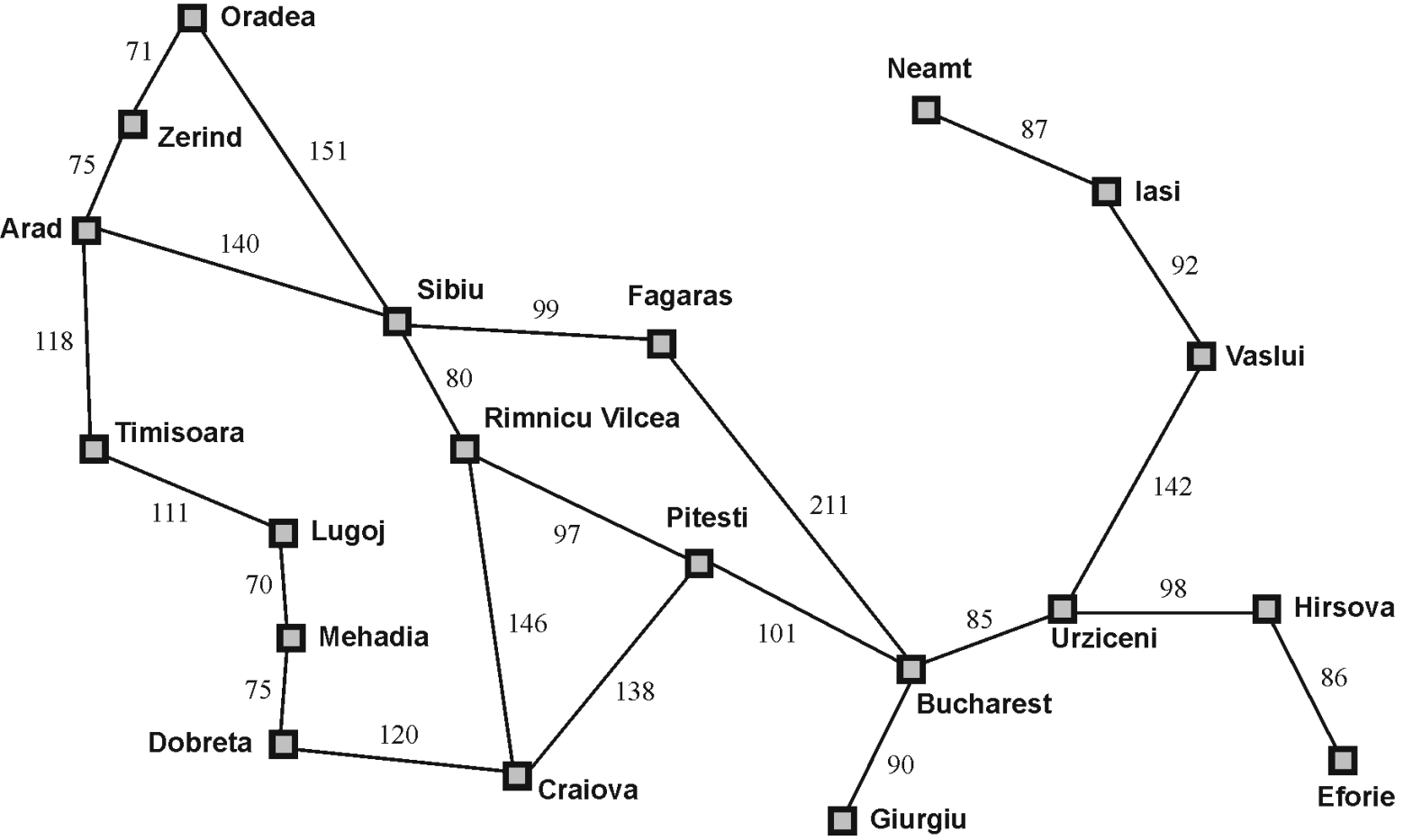
- Expande o nodo que está mais próximo do gol de acordo com a função heurística
  - $f(n) = h(n)$

- O termo "gulosa" significa que o método procura reduzir o custo imediato para alcançar o objetivo na expansão de cada nó, porém sem se preocupar com o custo total do caminho
  - Também chamada de *Best First Search*

Busca Gulosa – Exemplo

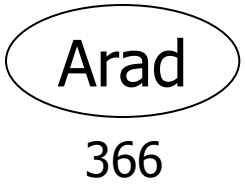
- Viagem na Romênia
  - Queremos achar um caminho (o melhor caminho) entre Arad e Bucarest
  - A heurística usada é a distância em linha reta entre as cidades

Busca Gulosa – Exemplo



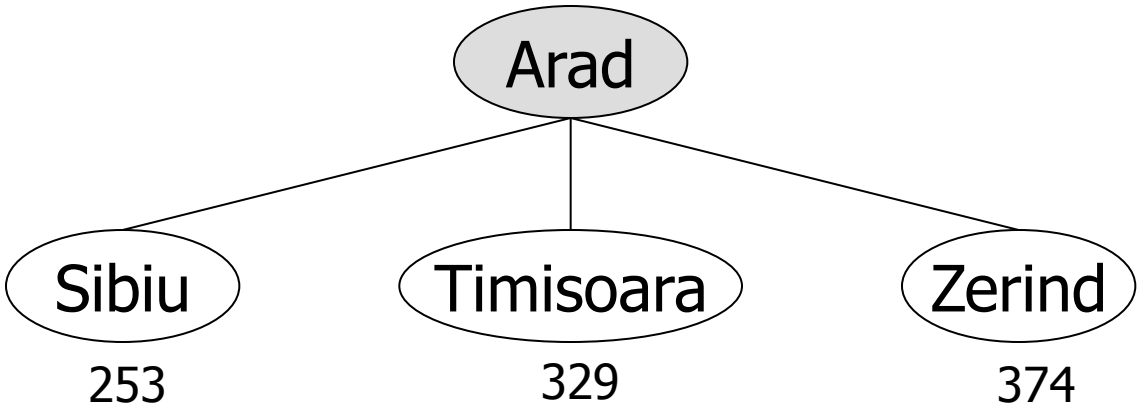
Arad	366
Bucharest	**0**
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu V.	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca Gulosa – Exemplo



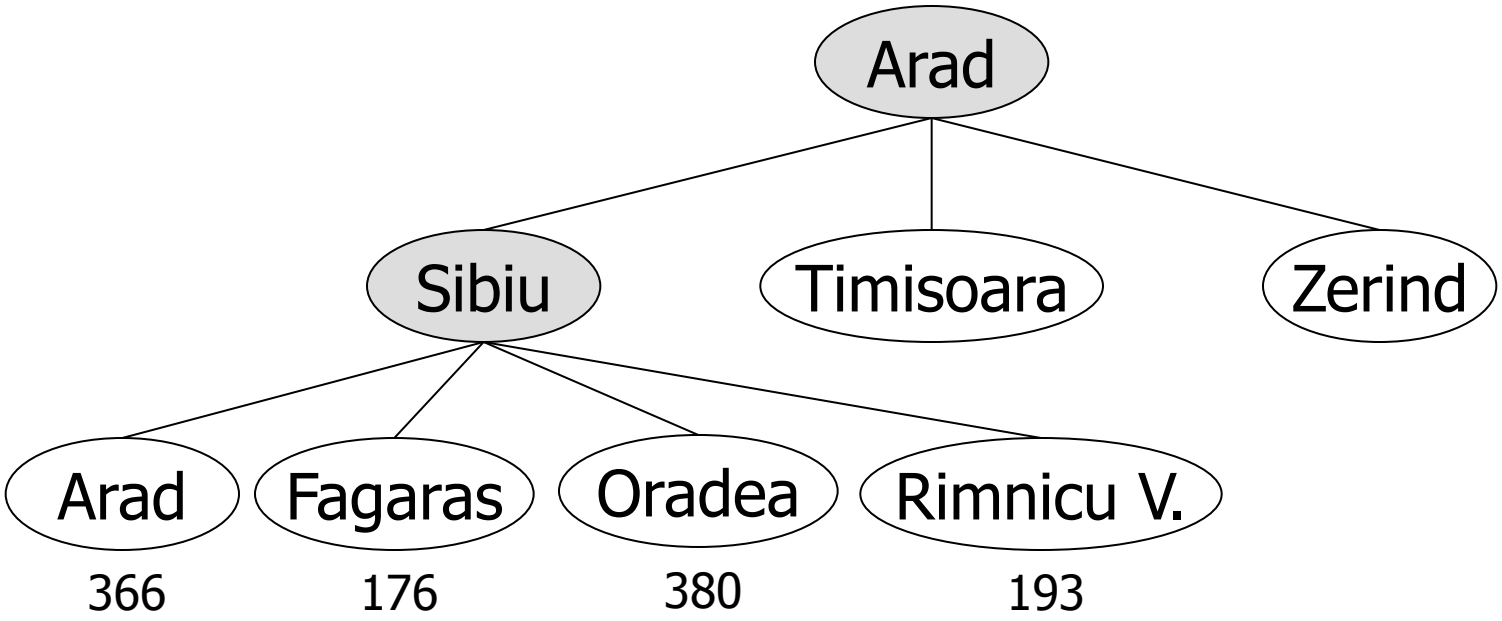
Arad	**366**
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu V.	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca Gulosa – Exemplo



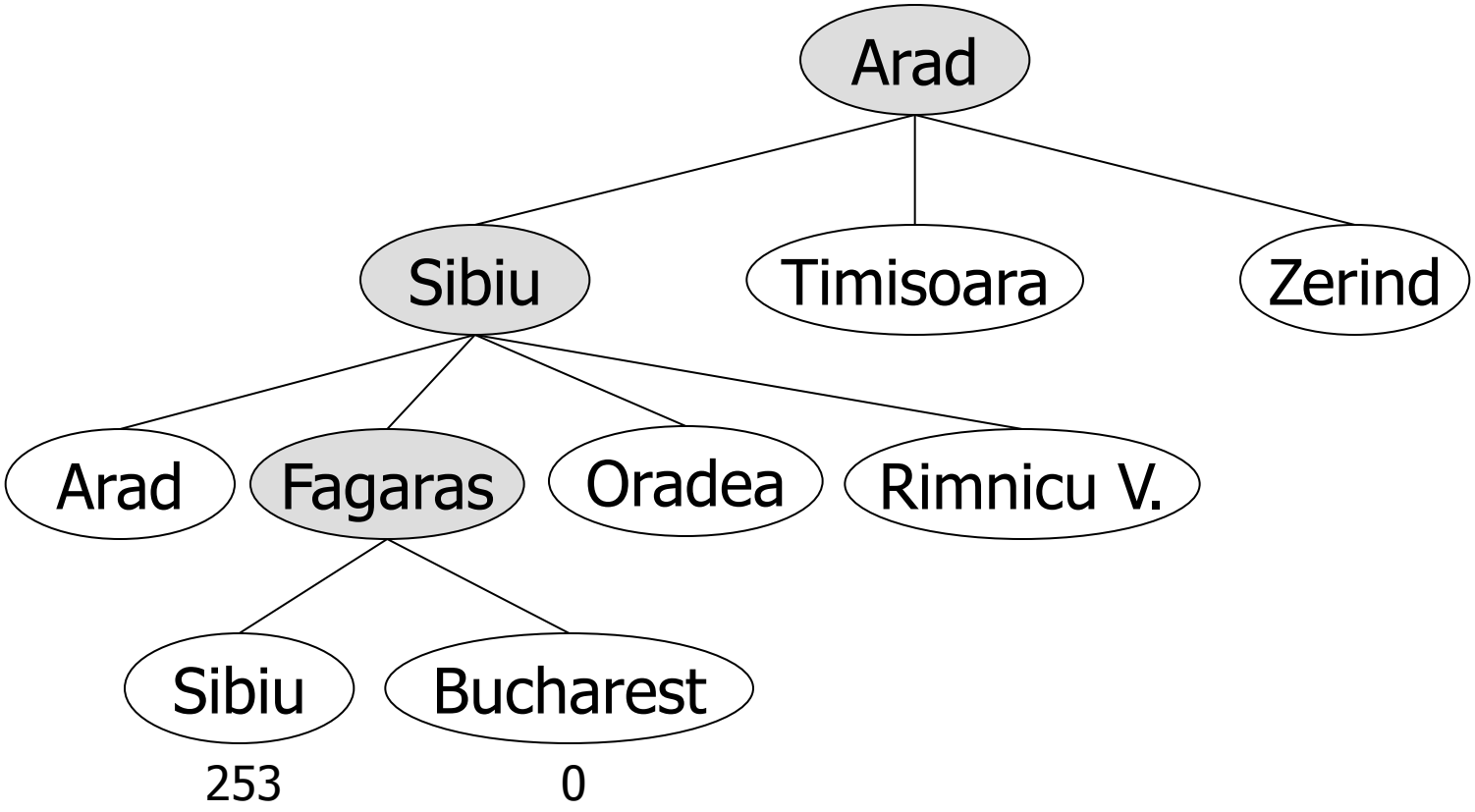
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu V.	193
Sibiu	**253**
Timisoara	**329**
Urziceni	80
Vaslui	199
Zerind	**374**

Busca Gulosa – Exemplo



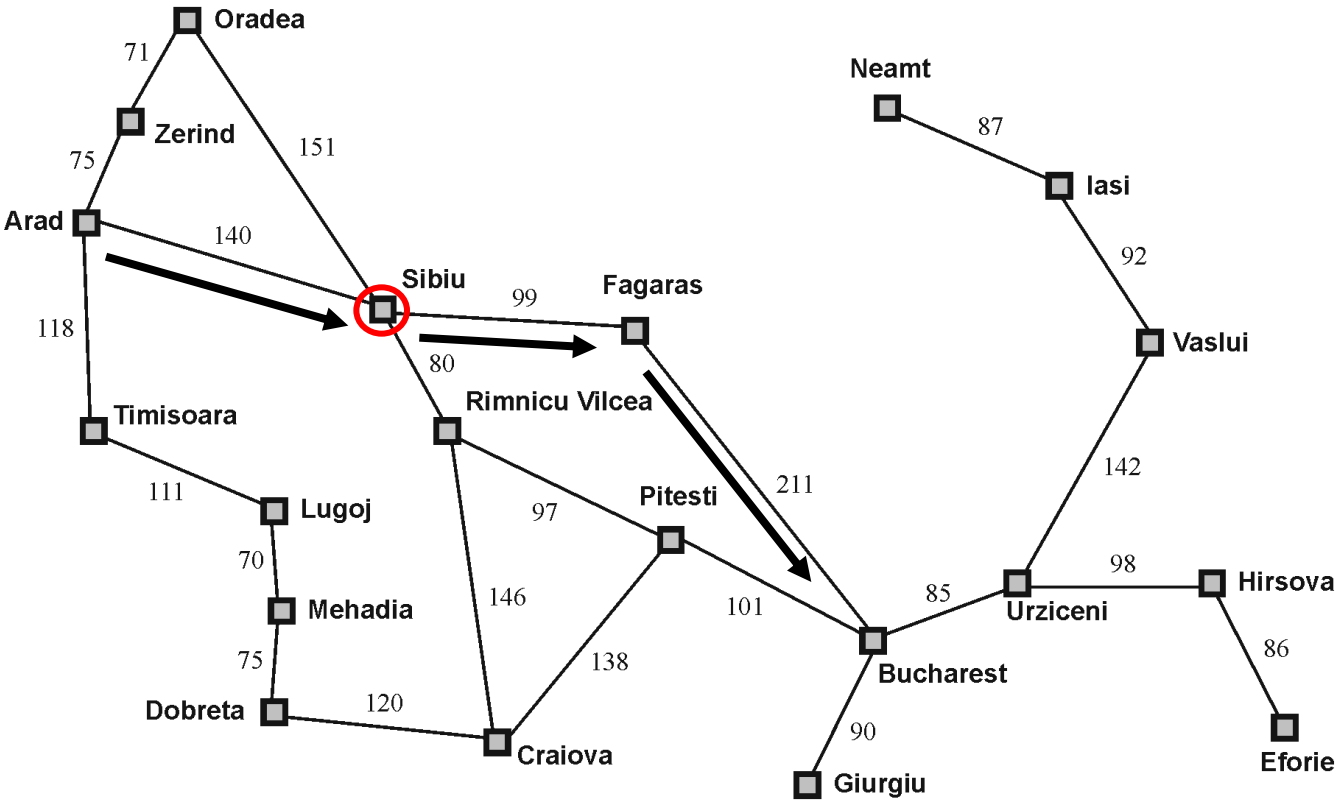
Arad	**366**
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	**176**
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	**380**
Pitesti	100
Rimnicu V.	**193**
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca Gulosa – Exemplo



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

Busca Gulosa – Exemplo



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

- É a melhor solução?
  - O custo computacional é mínimo
  - Mas a solução não é ótima

Busca Gulosa

- Ótimo: não
- Completo: não (pode entrar em loop)
- Complexidade de tempo e espaço $O(b^m)$, onde b é o fator de expansão e m a profundidade máxima da árvore de busca

- A escolha de uma função heurística adequada é fundamental para que o algoritmo seja eficiente

Algoritmo A*

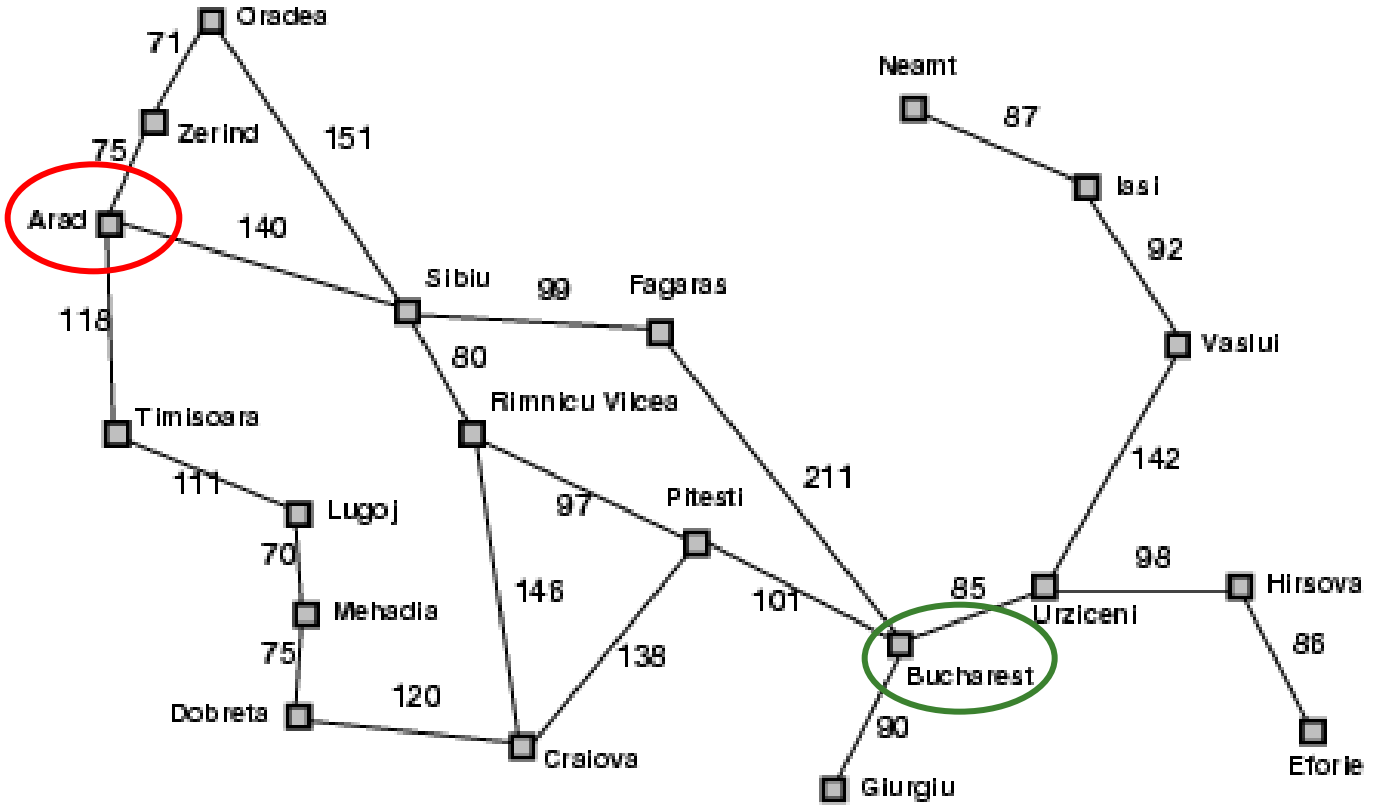
- A escolha do nodo a ser investigado é feita considerando-se:

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

  - g(n): **custo real** de chegar ao nó atual
  - h(n): **estimativa do custo** do nó atual para o gol

- Portanto, f(n) vai ser a estimativa do de custo da melhor solução que passa por n

A* - Exemplo

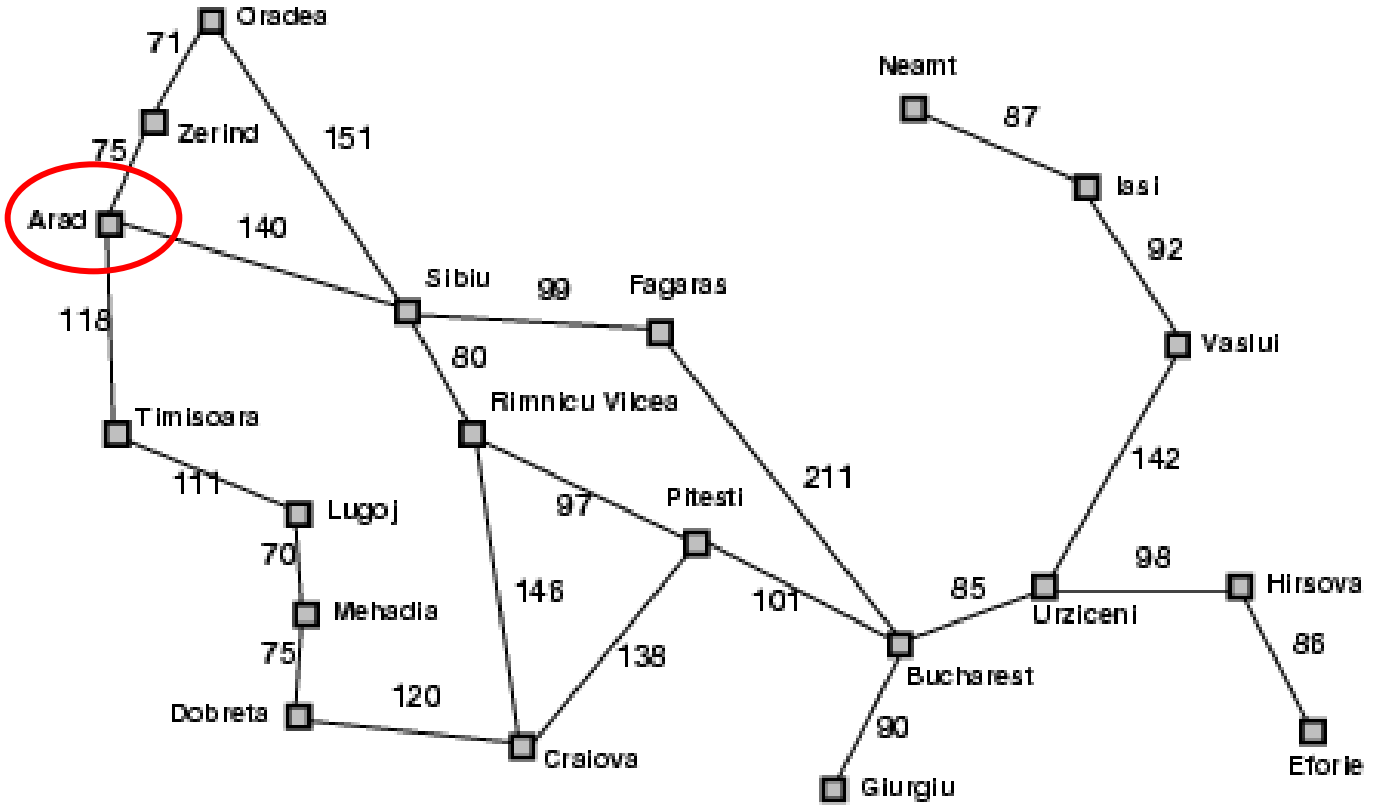


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* - Exemplo

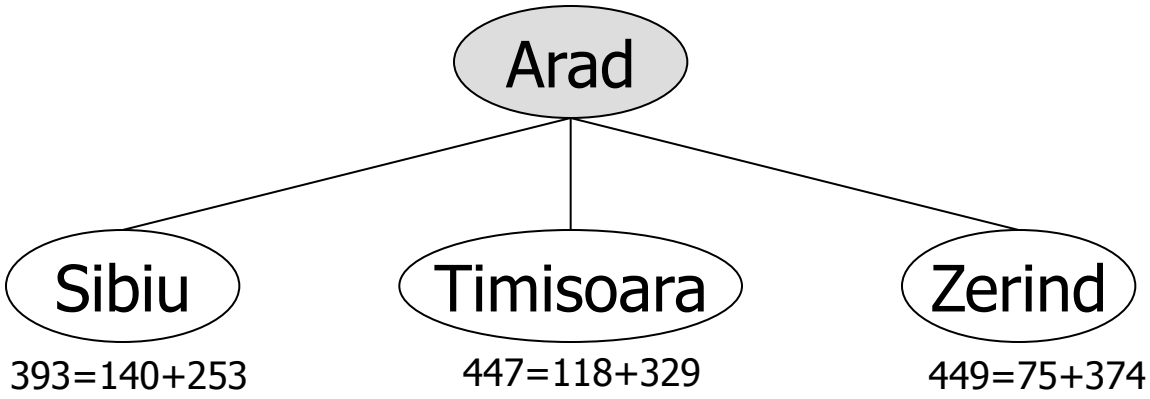
366=0+366

A* - Exemplo

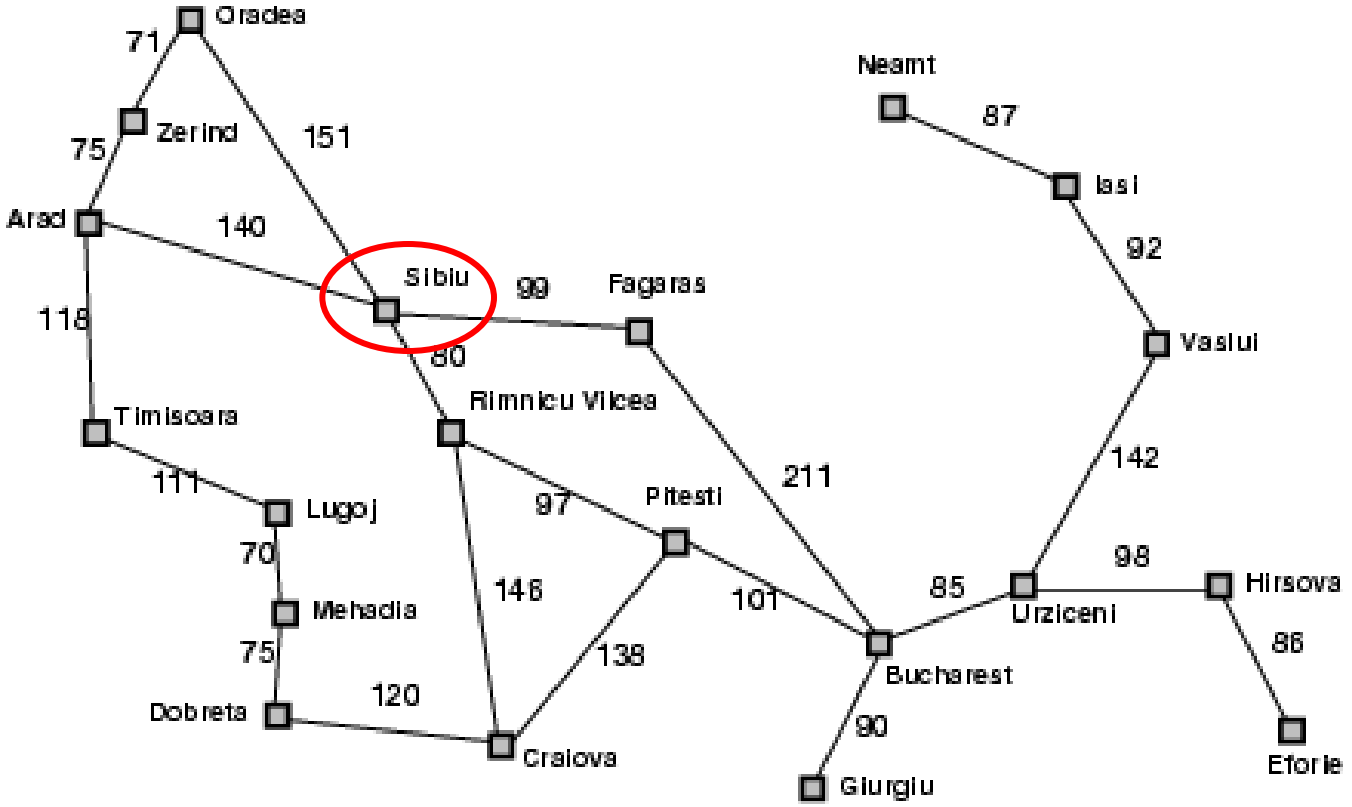


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* - Exemplo

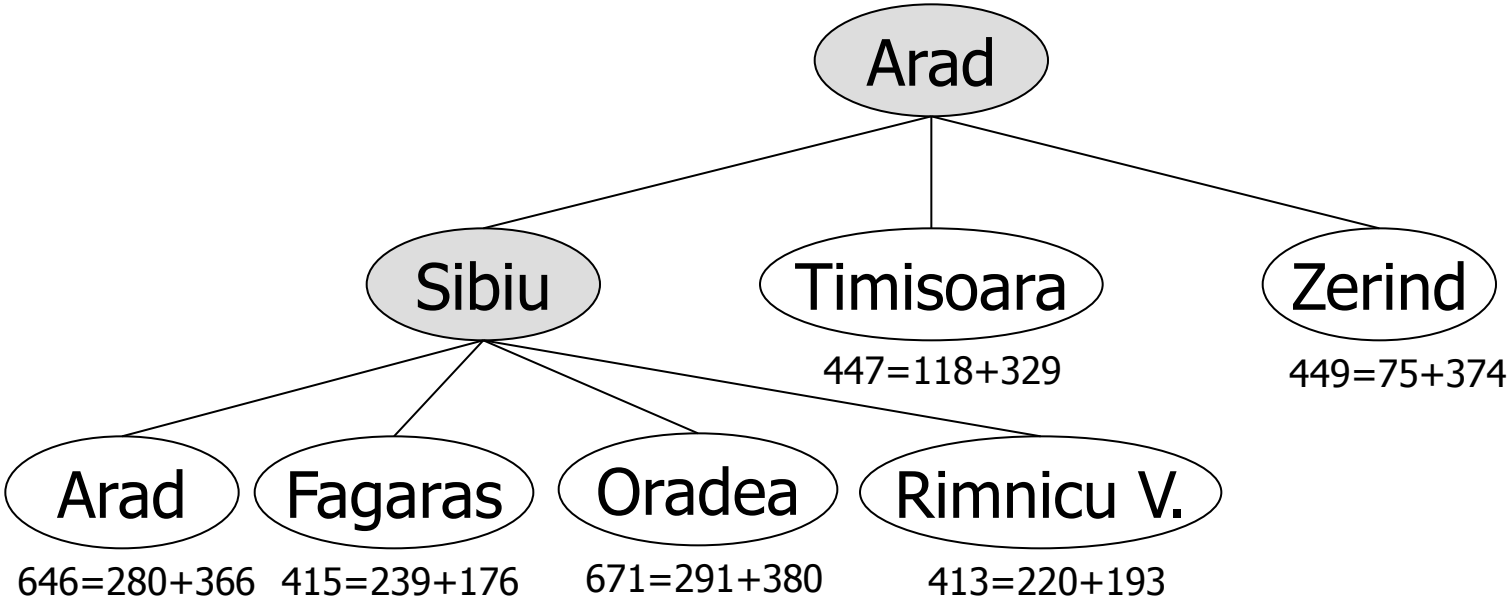

Arad
Sibiu
Timisoara
Zerind
393=140+253
447=118+329
449=75+374

A* - Exemplo

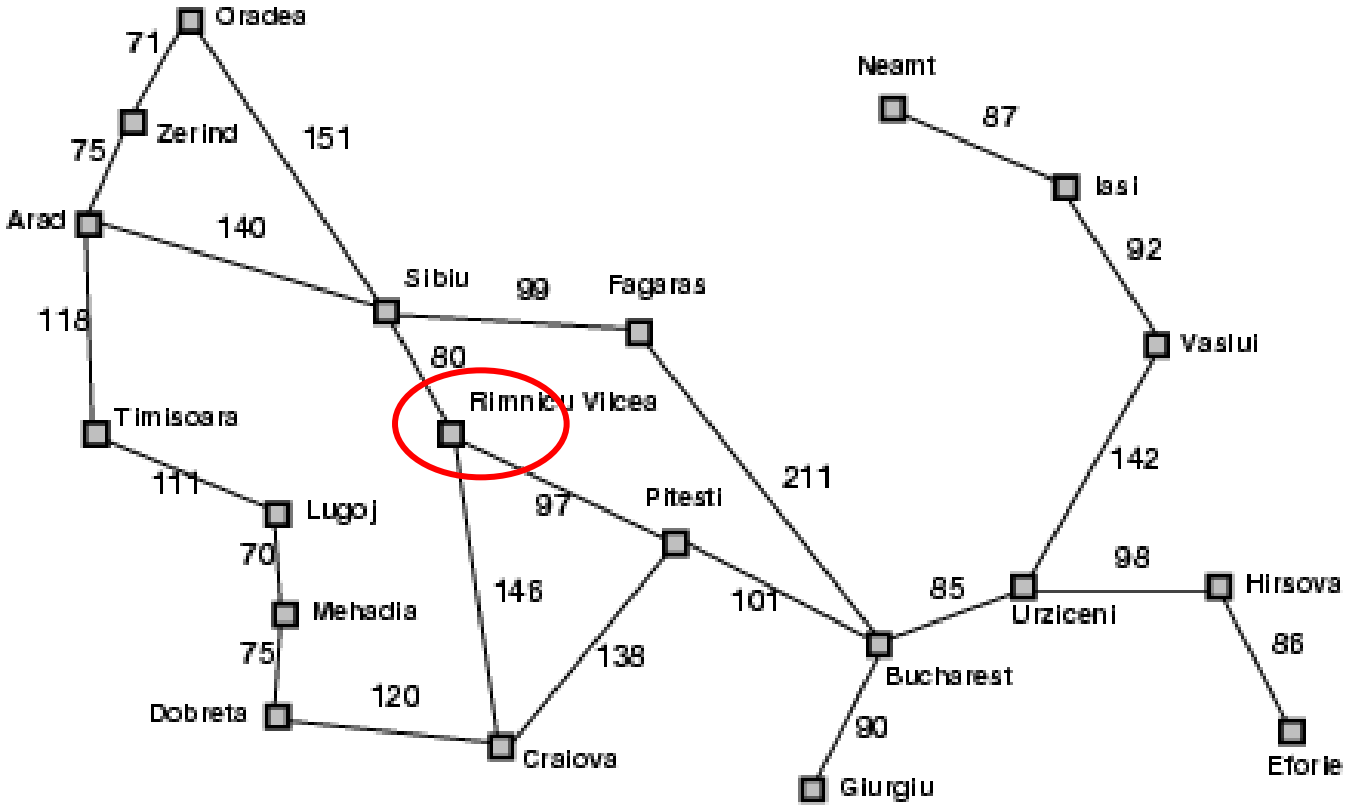


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* - Exemplo

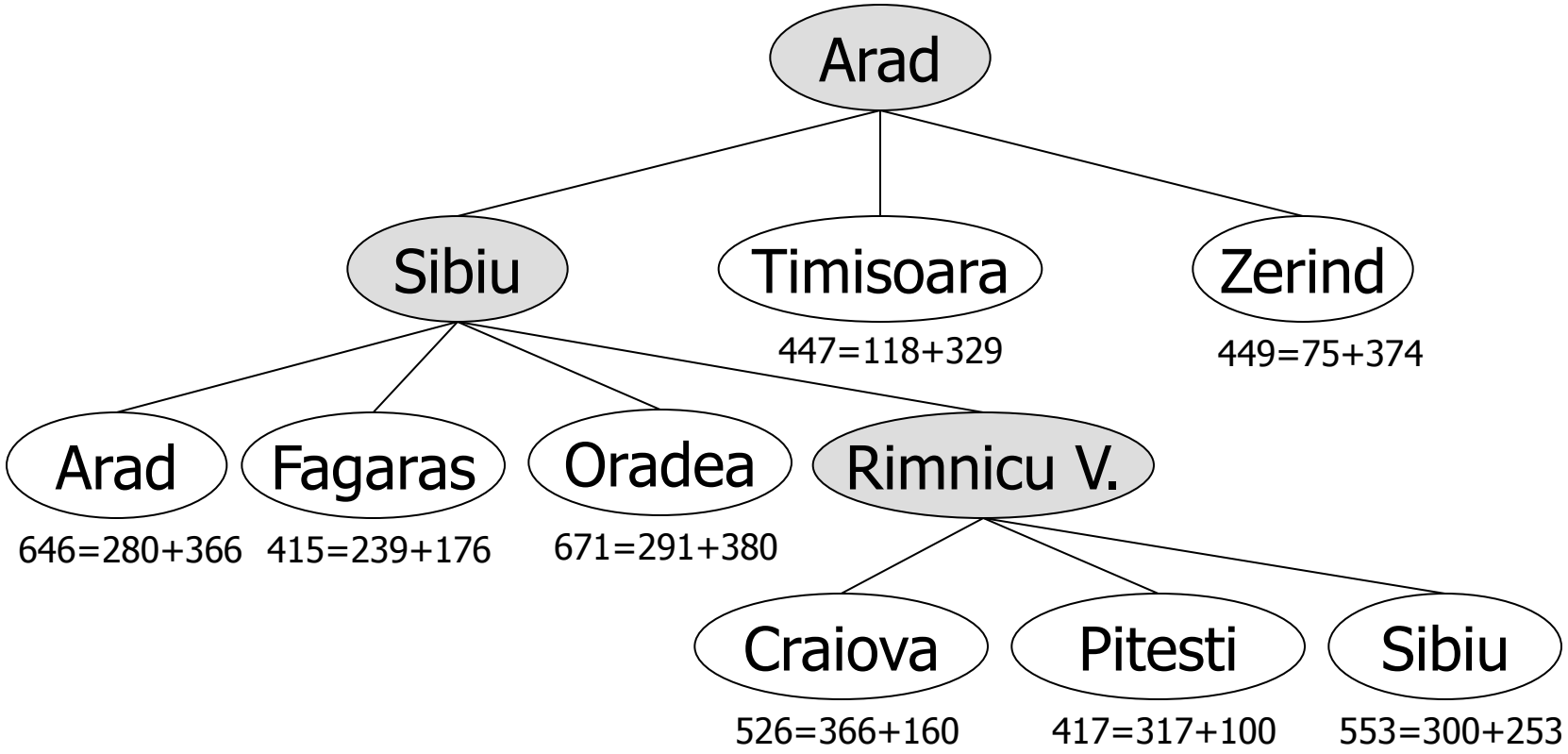

Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu V.
646=280+366
415=239+176
671=291+380
413=220+193

A* - Exemplo

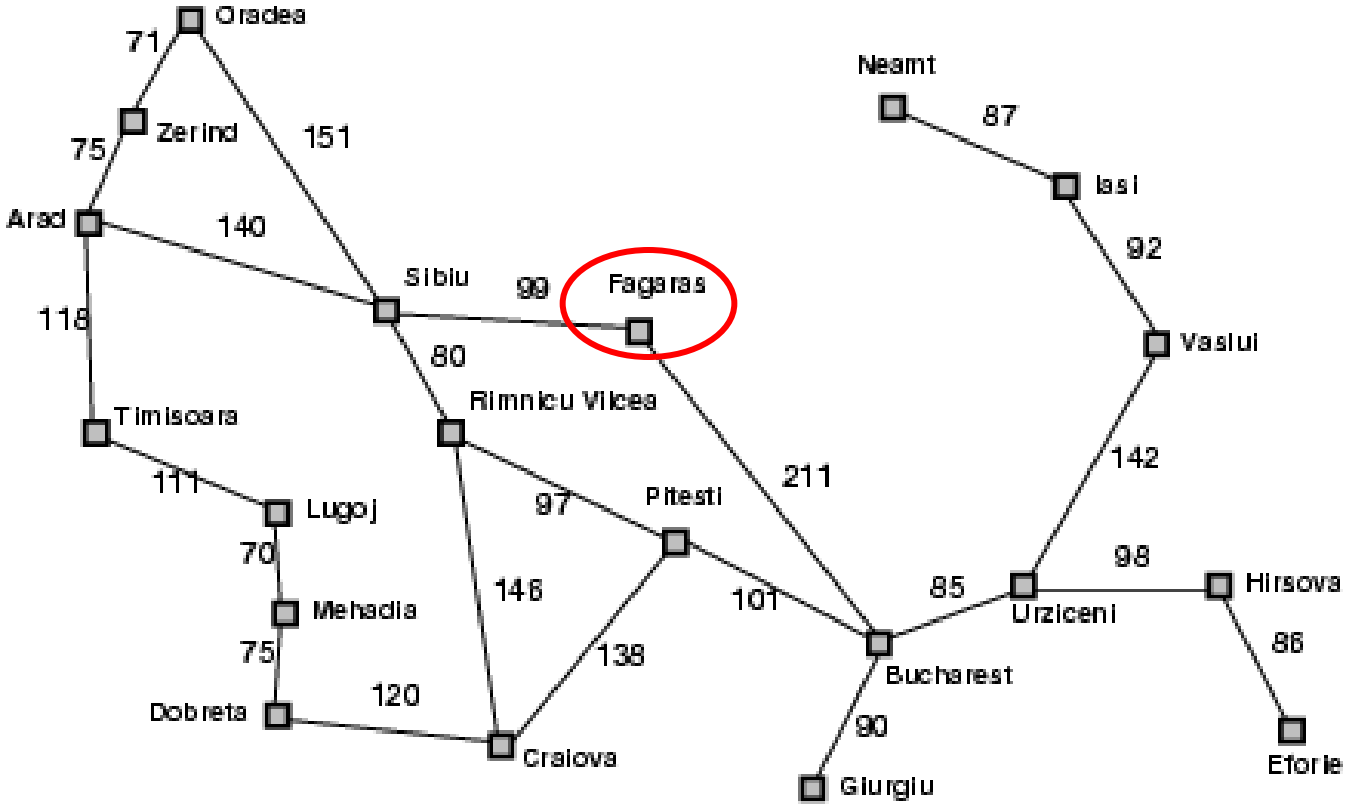


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* - Exemplo

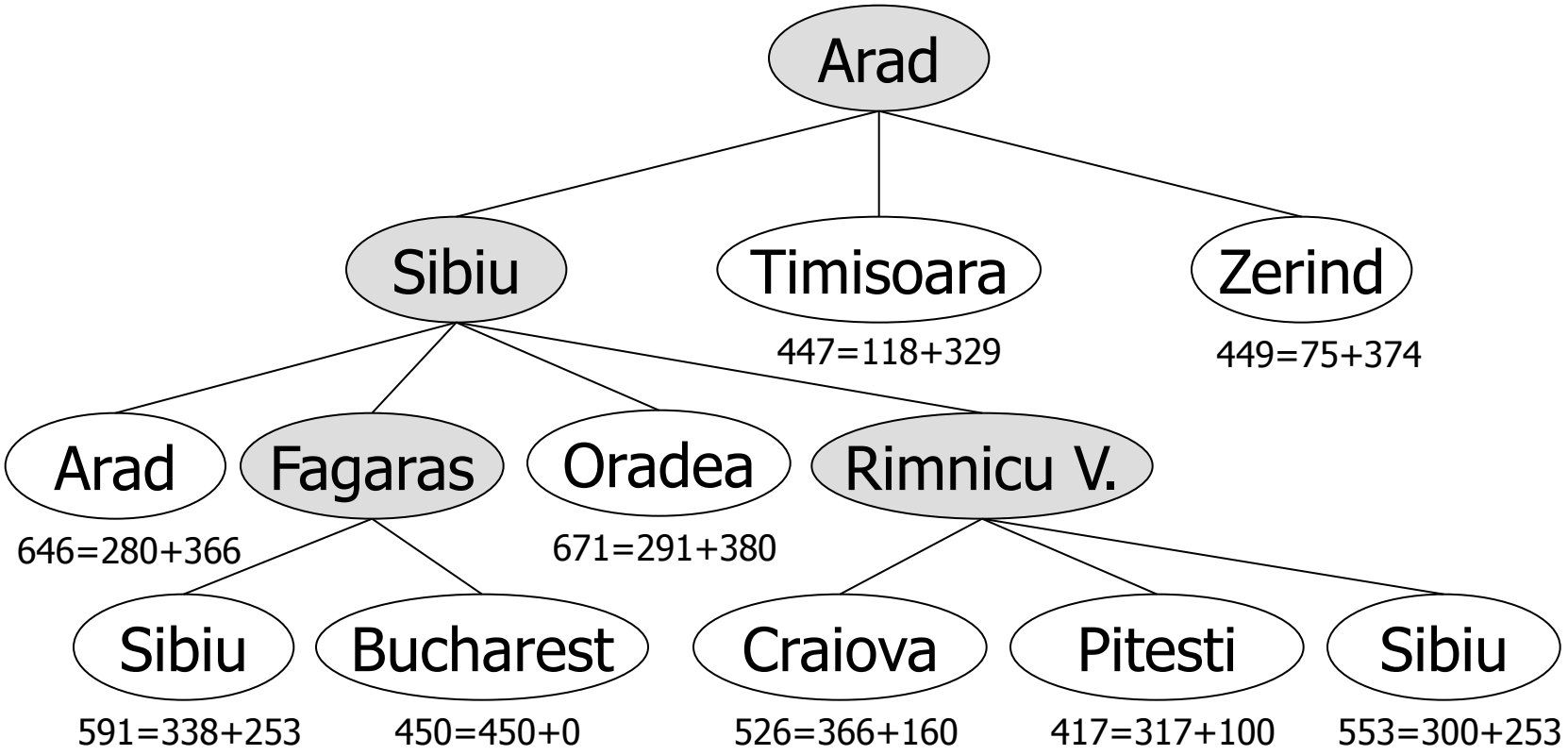

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu V.
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* - Exemplo

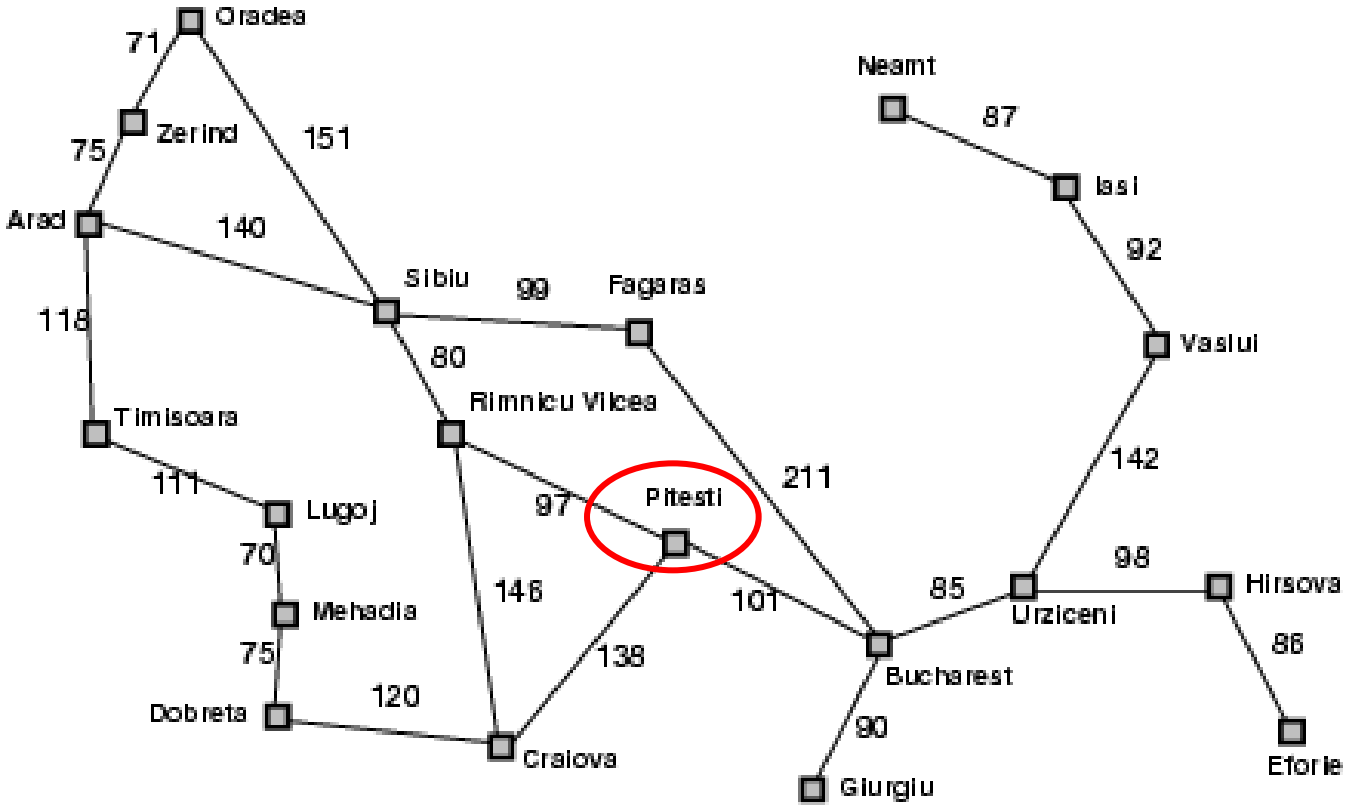


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* - Exemplo

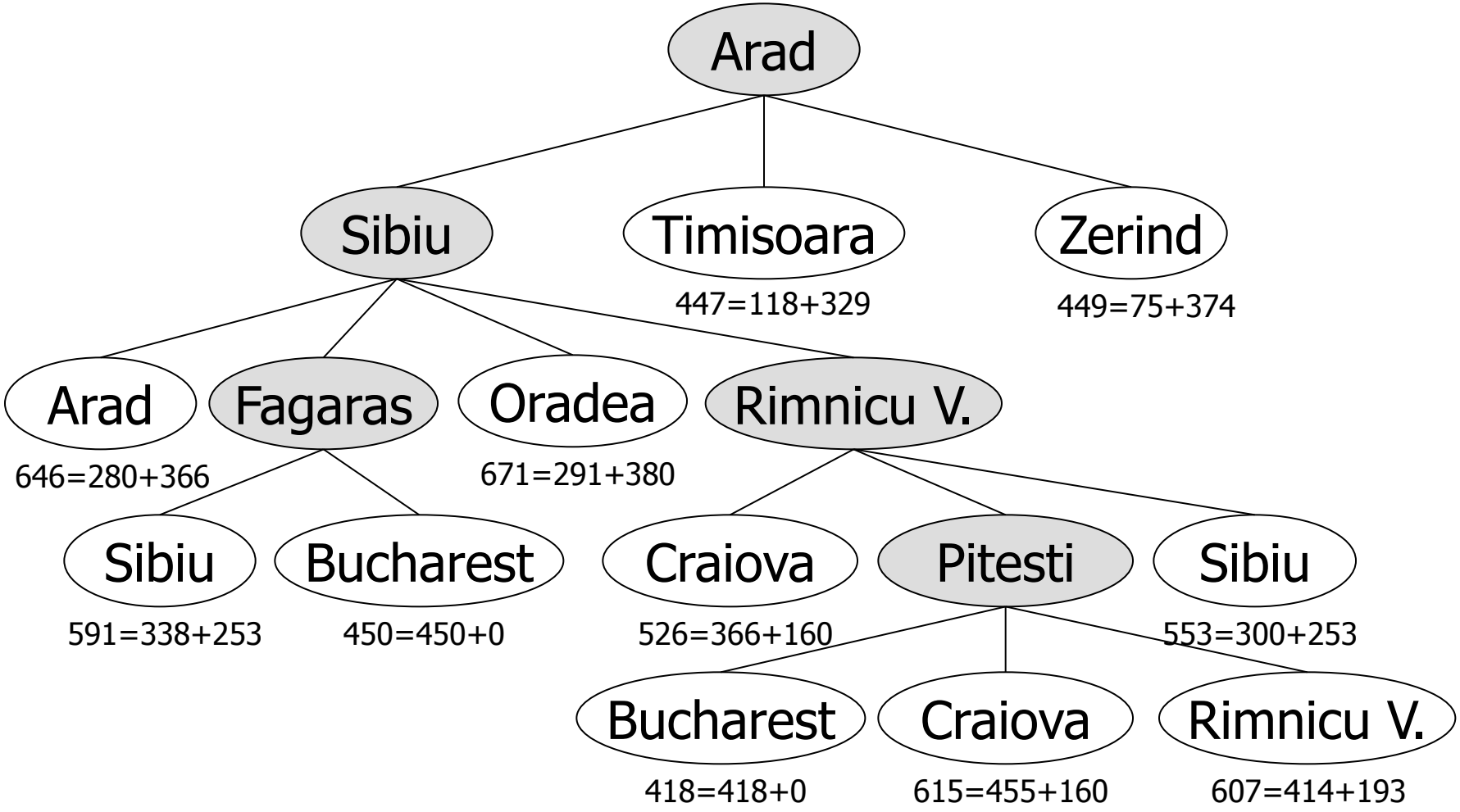

Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu V.
646=280+366
671=291+380
Sibiu
Bucharest
Craiova
Pitesti
Sibiu
591=338+253
450=450+0
526=366+160
417=317+100
553=300+253

A* - Exemplo



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Busca A* – Solução



Algoritmo A*

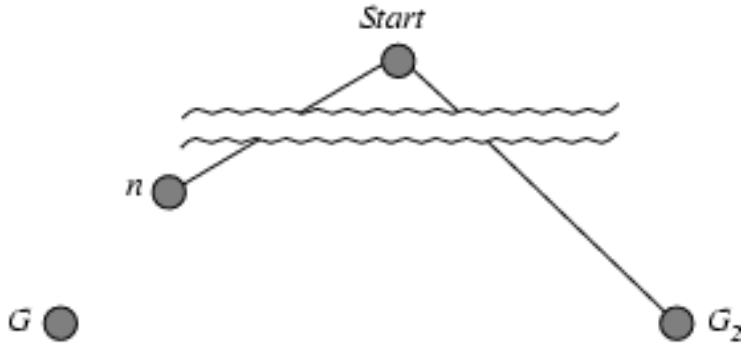
- Provadamente fornece a solução ótima se a heurística é **admissível**
  - Custo indicado pela heurística é menor ou igual ao custo real para o gol
  - No exemplo, a distância direta entre duas cidades é sempre menor ou igual a distância através das estradas. Logo, a solução encontrada é ótima

Algoritmo A*

```
1  Procedimento A*
2      g(s_0) ← 0;
3      f(s_0) ← g(s_0) + h(s_0);
4      parent(s_0) ← NULL;
5      open ← {s_0};
6      closed ← ∅;
7      while open ≠ ∅ do
8          current ← argmin_{n∈open}(f(n));
9          if current = s_goal then
10             return solução ótima;
11         end
12         open ← open \ {current};
13         closed ← closed ∪ {current};
14         foreach n ∈ Succ(n) do
15             if n ∉ closed then
16                 if n ∉ open then
17                     g(n) ← g(current) + c(current, n);
18                     f(n) ← g(n) + h(n);
19                     parent(n) ← current;
20                     open ← open ∪ {n};
21                 else if g(current) + c(current, n) < g(n) then
22                     g(n) ← g(current) + c(current, n);
23                     f(n) ← g(n) + h(n);
24                     parent(n) ← current;
25                 end
26             end
27         end
28     end
29     return não existe solução;
30 end
```

Otimalidade do A* (Tree Search)

- **Seja G_2 um nó gol gerado através de um caminho subótimo e que está na borda. Considere também que *n* seja um nó na borda, ainda não expandido, tal que n esteja no caminho ótimo para o gol. Seja C* o custo desse caminho ótimo.**



- ***f*(G_2) > *C**** **caminho que gerou G_2 é subótimo**
- ***f*(*n*) = *g*(*n*) + *h*(*n*) ≤ *C**** **h é admissível**

Logo,

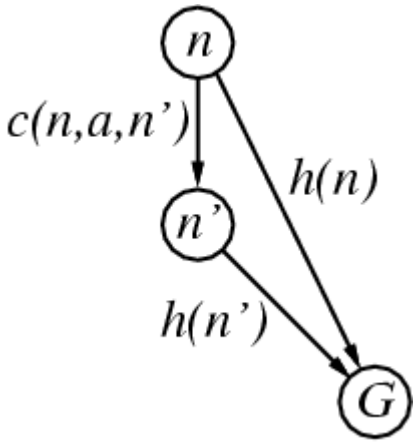
- **f(n) ≤ *C** < *f*(G_2).**

O nó G_2 não será expandido e o A* vai fornecer o resultado ótimo

Otimalidade do A*

- Pode falhar usando-se *Graph-Search* com eliminação de estados repetidos
  - Tomar cuidado na eliminação
  - Garantir que quando um nó é expandido, o caminho ótimo até ele foi encontrado
  - Isso é garantido se a heurística **é consistente**

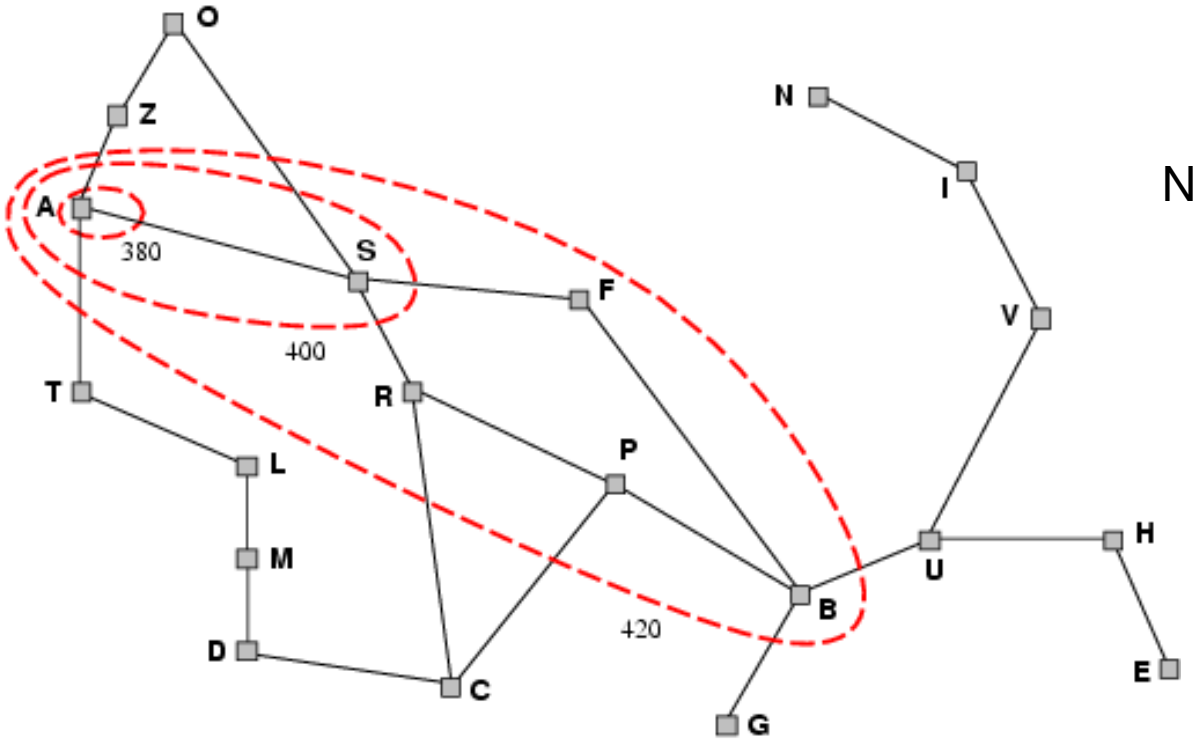
Uma heurística é consistente se, para um nó *n* e os seus sucessores *n'* gerados por uma ação *a,* o custo estimado de atingir o gol a partir de n não é maior que o custo de chegar a n' somado ao custo estimado de n' para o gol.



$$h(n) \leq c(n,a,n') + h(n')$$

Otimalidade do A*

- Uma outra consequência importante é:
  - Se a heurística é consistente, os valores de *f(n)* ao longo de um caminho são não decrescentes

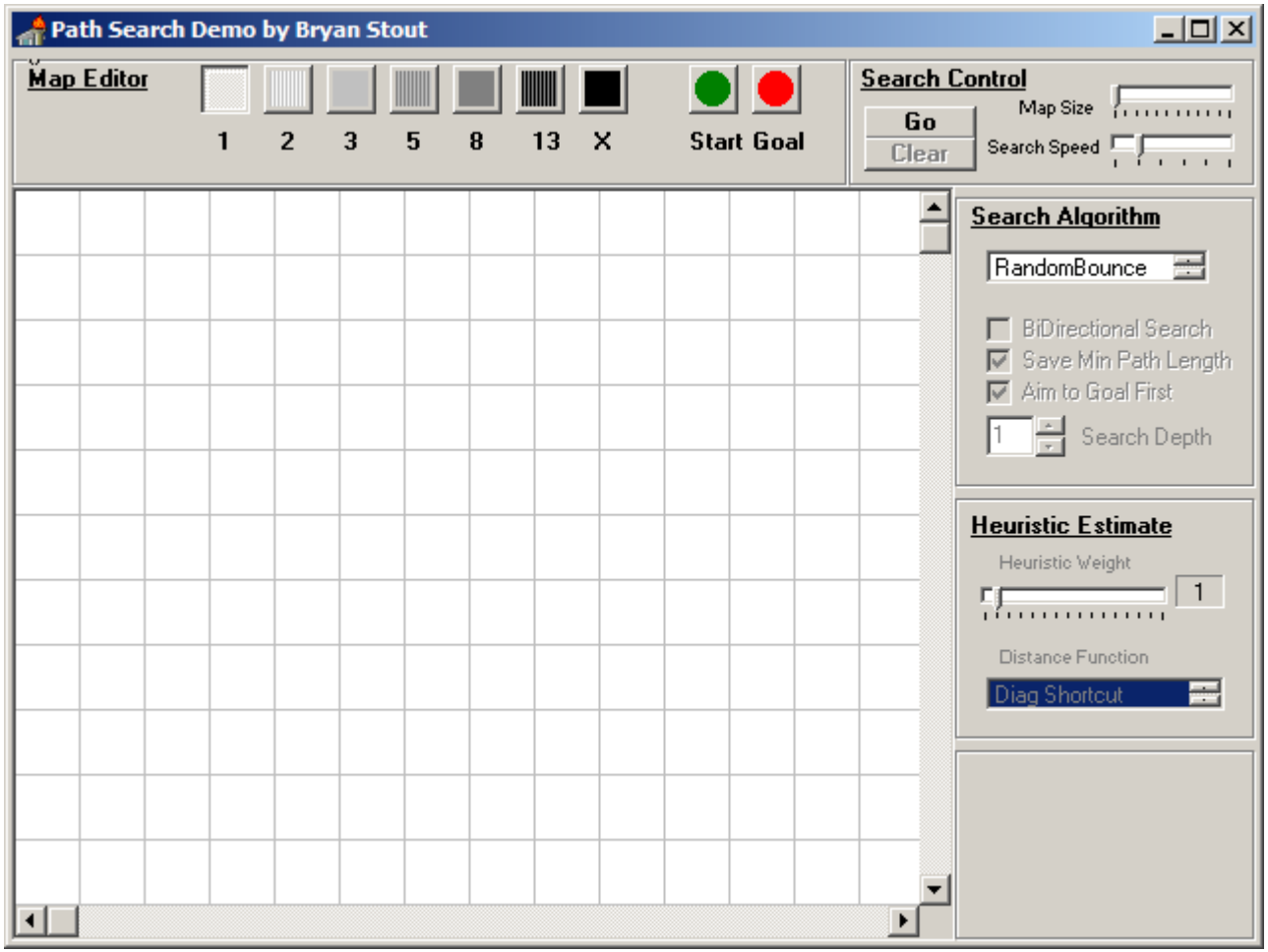


Não expande nós com f(n) > C*

Algoritmo A*

- Ótimo: sim (se a heurística é admissível)
  - Também é eficientemente ótimo (expande o menor número de nodos possível dentre os algoritmos ótimos)
- Completo: sim
- Complexidade
  - Tempo: Apesar disso tudo, dependendo da heurística, o número de nodos expandidos ainda pode ser uma função exponencial da solução.
  - Espaço: Nodos são mantidos na memória
    - Soluções: IDA*, SMA*, etc,

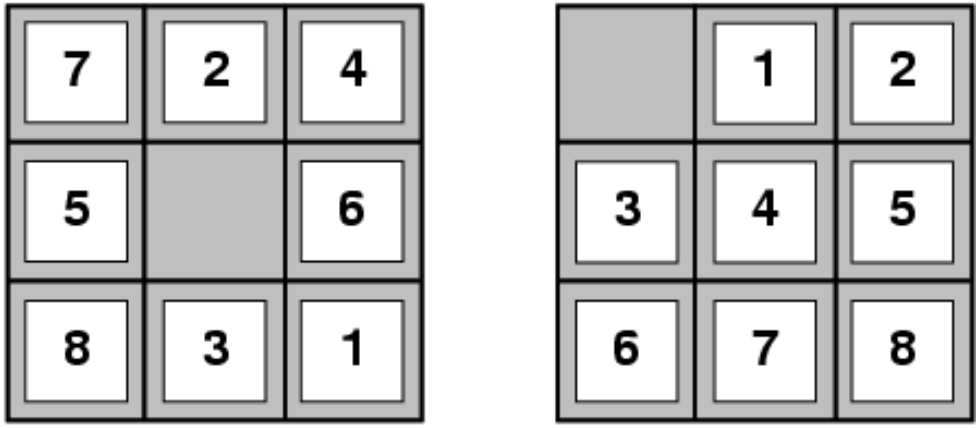
Comparação "Visual" entre os algoritmos de busca



Funções Heurísticas

- A escolha de uma heurística adequada é fundamental para o bom funcionamento dos algoritmos de busca com informação
- Como escolher uma boa heurística?
- Alguns conceitos importantes através de Exemplos do 8-puzzle

8-Puzzle



- Solução média requer 22 movimentos
- Considerando um *branching factor* médio igual a 3
  - $3^{22} = 3.1 \times 10^{10}$ estados investigados
  - Eliminando estados repetidos: 170.000
  - Não é muito, mas e o 15 puzzle?

Heurísticas para o 8-puzzle

- h_1: número de peças na posição errada
  - Na figura, h1=8
  - É admissível porque cada peça deverá ser movimentada pelo menos uma vez
- h_2: soma das distâncias de cada peça às suas posição correta
  - Não considera diagonais (*Manhattan Distance)*
  - Na figura, h2= 3+1+2+2+2+3+3+2 = 18
  - É admissível porque cada ação movimenta cada peça apenas um passo mais próximo do objetivo

Qualidade da Heurística

- *Effective Branching Factor (b*)*
  - Seja n o número de nodos gerados pelo algoritmo e d a profundidade da solução
  - b* é o branching factor de uma árvore uniforme com n+1 nodos
    - $n+1 = 1+b^*+(b^*)^2+...+(b^*)^d$
  - Uma boa heurística deve apresentar b* o mais próximo possível de 1

Qualidade da Heurística

	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

Dominação

- Para qualquer nodo *n*, $h_2(n) \geq h_1(n)$
- É dito que h_2 ***domina*** h_1
- Logo, h_2 expande menos nodos que h_1
  - Todos os nodos com $f(n) < C^*$ (ou de outra forma, com $h(n) < C^* - g(n)$) serão expandidos

- É vantajoso escolher heurísticas com valores grandes, desde que elas não ultrapassem o custo real e o tempo para computá-las não seja muito grande

Criação de Heurísticas

- Heurísticas admissíveis podem ser obtidas a partir de soluções ótimas de versões **relaxadas** do problema
- Problema relaxado é um problema com menos restrições que o original
- É importante que a solução do problema relaxado seja simples de se obter

Criação de Heurísticas

- 8-Puzzle original
  - Pode-se mover uma peça de A para B **se** A é adjacente a B **e** B está vazio
- 8-Puzzle relaxado
  - Uma peça pode ser movida de A para B (h_1)
  - Uma peça pode ser movida de A para B caso A seja adjacente a B (h_2)
  - Uma peça pode ser movida de A para B caso B esteja vazia (*Heurística de Gaschnig*)

Criação de Heurísticas

- Como fazer se for difícil de determinar qual é a melhor heurística?

- $h(n) = \max\{h_1, h_2, ..., h_m\}$
  - h(n) é admissível se todas as outras forem
  - h(n) domina todas as outras

- Nesse caso, a melhor heurística é utilizada em cada situação específica

Criação de Heurísticas

- Heurísticas também podem ser criadas a partir da solução de subproblemas
  - Ex. posicionar apenas as peças 1, 2, 3 e 4
  - Se os subproblemas forem disjuntos pode-se somar as heurística obtidas
- Heurísticas também podem ser criadas por "experiência"
  - Resolve-se várias instâncias do problema e, com "sorte", consegue-se estimar o custo de outras instâncias.
    - Aprendizado por indução (ex. Redes Neurais)