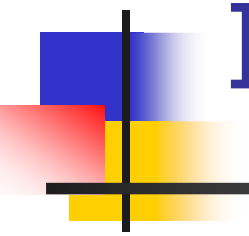


Inteligência Artificial

Prof. Tiago A. E. Ferreira

Aula 9 – Busca com Heurísticas

(Parte I)

Busca Heurística - Informada

- Estratégias de Busca Exaustiva (Cega)
  - Encontram soluções para problemas pela geração *sistemática* de novos estados, que são comparados ao objetivo;
  - São *ineficientes* na maioria dos casos:
    - São capazes de calcular *apenas* o *custo de caminho* do nó atual ao nó inicial (função *g*), para decidir qual o próximo nó da fronteira a ser expandido.
    - Essa medida não necessariamente conduz a busca na direção do objetivo.
  - Como encontrar um barco perdido?
    - Não podemos procurar no oceano inteiro...
    - Observamos as correntes marítimas, o vento, etc...

Busca Heurística - Informada

- Estratégias de Busca Heurística
  - Utilizam *conhecimento específico* do problema na escolha do próximo nó a ser expandido
  - Barco perdido
    - correntes marítimas, vento, etc...
- Aplica uma *função de avaliação* a cada nó na fronteira do espaço de estados
  - Essa função estima o *custo de caminho* do nó atual até o objetivo mais próximo utilizando uma *função heurística*.
- Classes de algoritmos para busca heurística:

1. Busca pela melhor escolha (Best-First search)
2. Busca com limite de memória
3. Busca com melhora iterativa

Funções Heurísticas

- Função heurística - *h*
  - Estima o custo do caminho mais barato do estado atual até o estado final mais próximo.
- Funções heurísticas são específicas para cada problema
- Exemplo: encontrar a rota mais barata de Jeremoabo a Cajazeiras
  - $h_{dd}(n)$ = distância direta entre o nó *n* e o nó final.
- Como escolher uma boa função heurística?
  - Ela deve ser admissível
  - I.E., nunca *superestimar* o custo real da solução
- Distância direta (h_{dd}) é *admissível* porque o caminho mais curto entre dois pontos é sempre uma linha reta (Métrica do Espaço Euclidiano)

Busca pela Melhor Escolha (BME)
Best-First Search

- Busca genérica onde o nó de *menor custo* "*aparente*" na fronteira do espaço de estados é expandido primeiro
- Duas abordagens básicas:
  1. Busca Gulosa (Greedy Search)
  2. Algoritmo A*
- Algoritmo:

Função-Insere - ordena nós com base na Função-Avaliação

função **Busca-Melhor-Escolha** (*problema*,**Função-Avaliação**)

retorna **uma solução**

Busca-Genérica (*problema*, Função-Insere)

BME: Busca Gulosa

- Semelhante à busca em profundidade com *backtracking*
- Tenta expandir o nó mais próximo do nó final com base na estimativa feita pela *função heurística h*
- Algoritmo:

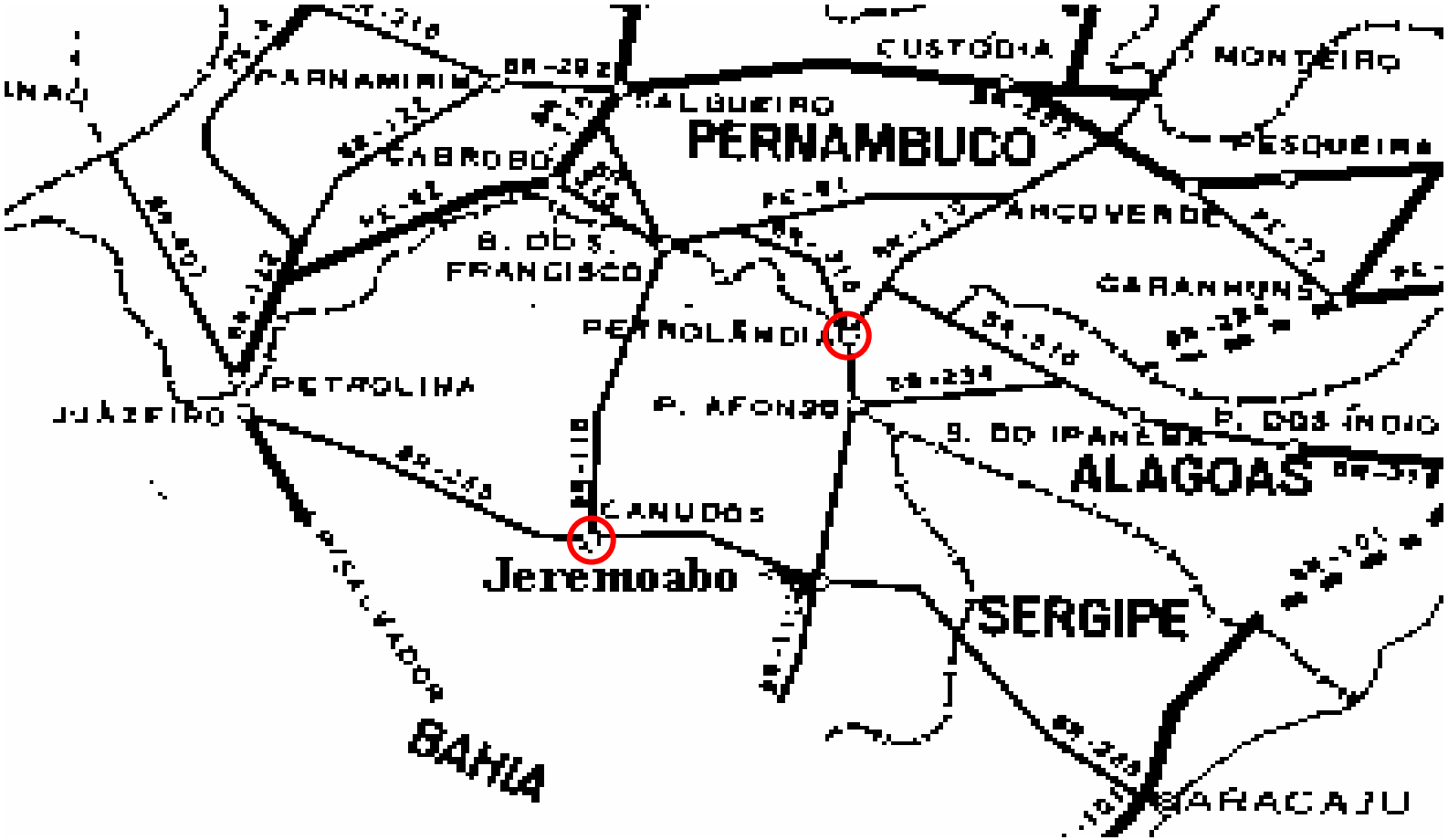
  função **Busca-Gulosa** (***problema***)

  retorna **uma solução ou falha**

  Busca-Melhor-Escolha (*problema*, *h*)

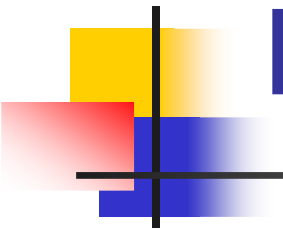
- Exemplo: **encontrar a rota mais barata de *Canudos* a *Petrolândia***
  - $h_{dd}(n)$ = distância direta entre o nó *n* e o nó final
  - h_{dd} é admissível!

Busca Gulosa



Busca Gulosa

- Custo de busca mínimo!
  - não expande nós fora do caminho
- Porém *não* é ótima:
  - escolhe o caminho que é mais econômico à primeira vista
    - Belém do S. Francisco, Petrolândia = 4,4 unidades
  - porém, existe um caminho mais curto de Canudos a Petrolândia
    - Jeremoabo, P. Afonso, Petrolândia = 4 unidades
- A solução via Belém do S. Francisco foi escolhida por este algoritmo porque
  - $h_{dd}(BSF) = 1,5$ *u.*, enquanto $h_{dd}(Jer) = 2,1$ u.



Busca Gulosa

- Não é completa:
  - pode entrar em looping se não detectar a expansão de estados repetidos
  - pode tentar desenvolver um caminho infinito

- Custo de tempo e memória: $O(b^d)$
  - guarda todos os nós expandidos na memória

BME: Algoritmo A*

- Tenta minimizar o custo total da solução combinando:

  Busca Gulosa
  - econômica, porém não é completa nem ótima

  Busca de Custo Uniforme
  - ineficiente, porém completa e ótima
- Função de avaliação:
  - $f(n) = g(n) + h(n)$
  - $g(n)$ = distância de n ao nó inicial
  - $h(n)$ = distância estimada de n ao nó final
- A* expande o nó de menor valor de f na fronteira do espaço de estados.

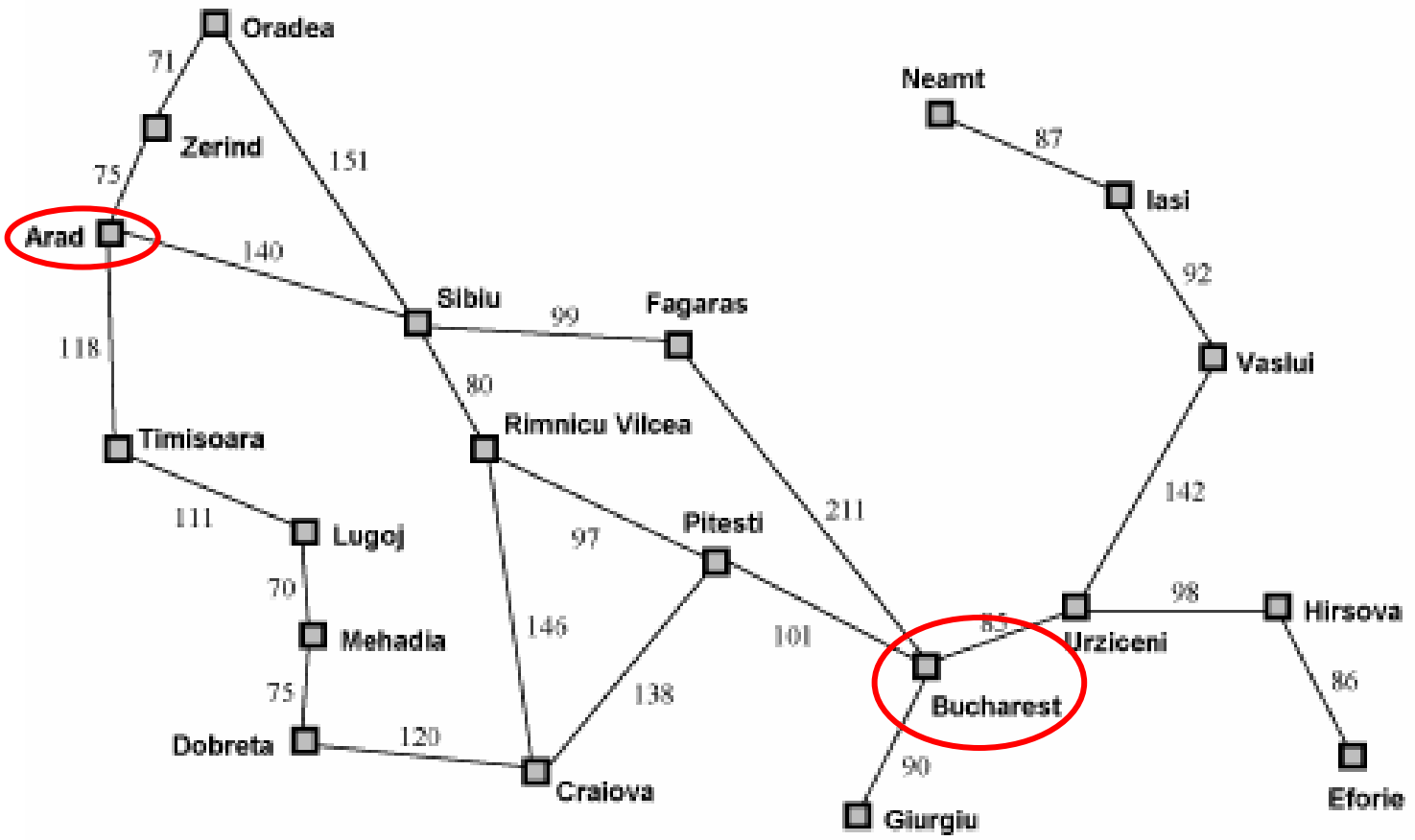
Algoritmo A*

- Se *h* é *admissível*, *f (n)* nunca irá superestimar o custo real da melhor solução através de *n.*
- Neste caso, a rota escolhida entre *Canudos* e *Petrolândia* é de fato a mais curta, uma vez que:
  - *f (BSF) = 2,5 u + 1,5 u = 4 u*
  - *f (Jeremoabo) = 1,5 u + 2,1 u = 3,6 u*
- Algoritmo:
  função **Busca-A* (*problema*)**
  retorna **uma solução ou falha**

  Busca-Melhor-Escolha (*problema*, *g+h*)

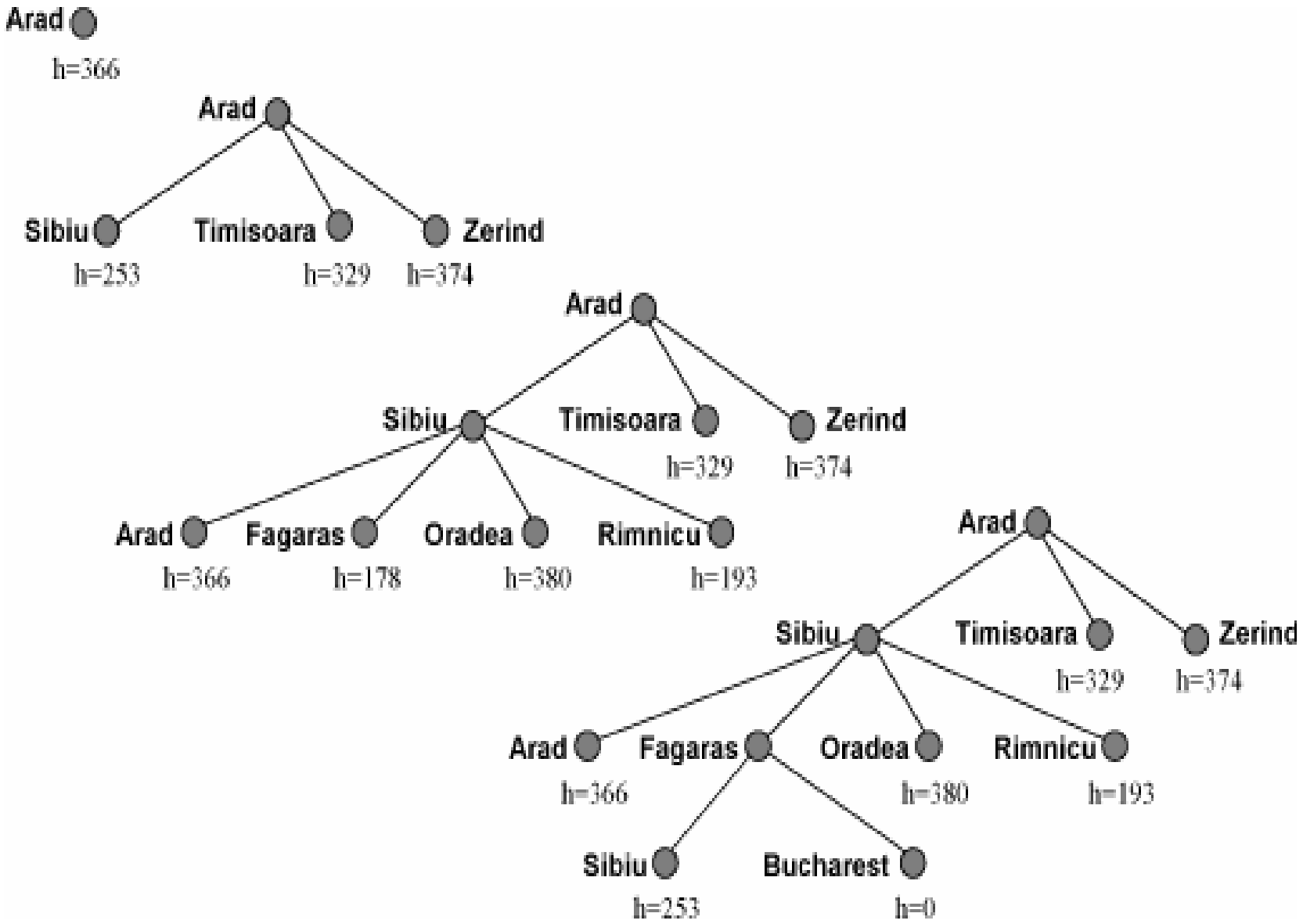
Algoritmo A* : exemplo

Ir de Arad a Bucharest

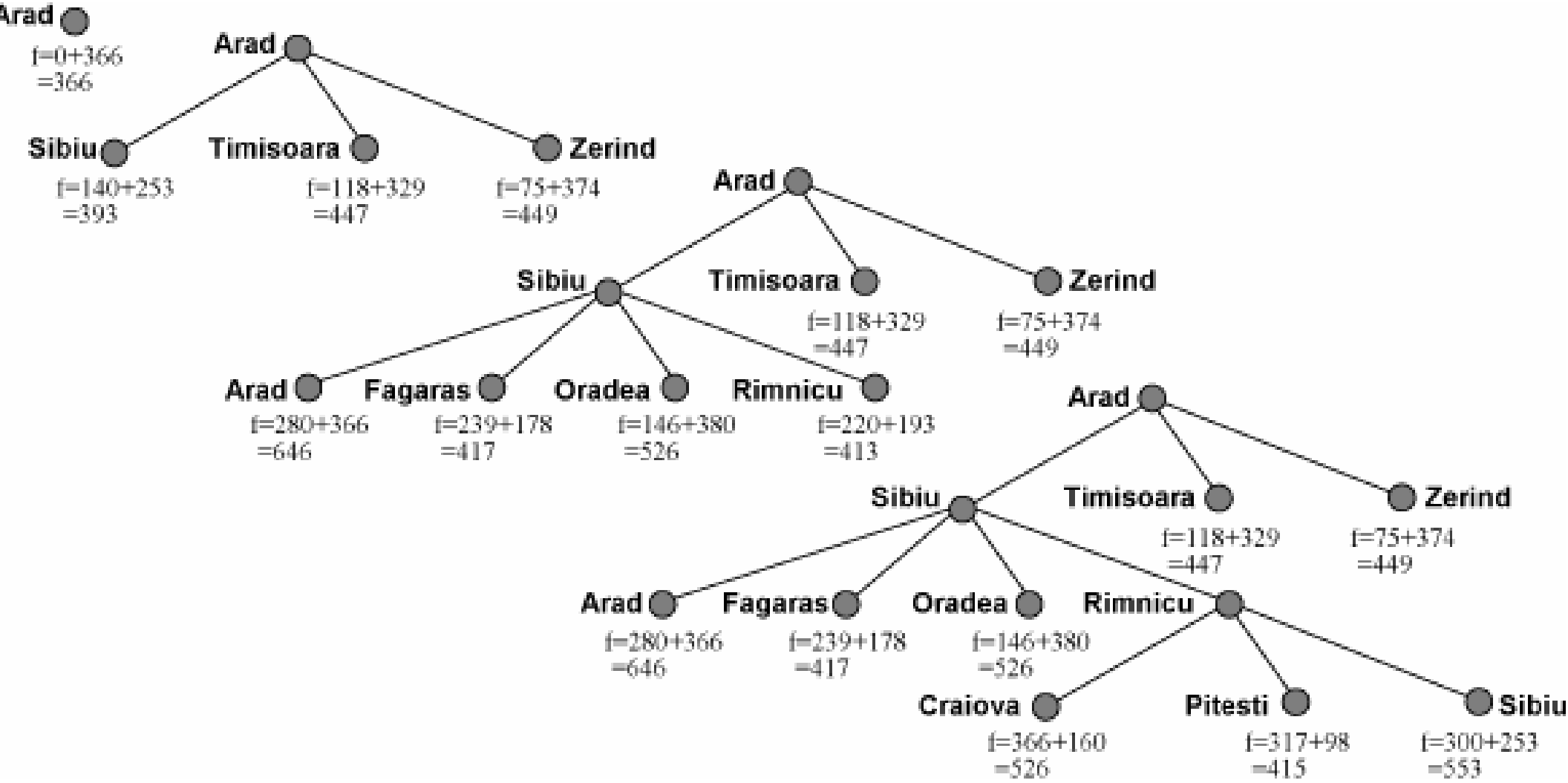


Straight line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Se fosse pela Busca Gulosa...



Usando A*


Arad
f=0+366
=366
Arad
Sibiu
Timisoara
Zerind
f=140+253
=393
f=118+329
=447
f=75+374
=449
Arad
Sibiu
Timisoara
Zerind
f=118+329
=447
f=75+374
=449
Arad
Fagaras
Oradea
Rimnicu
f=280+366
=646
f=239+178
=417
f=146+380
=526
f=220+193
=413
Arad
Sibiu
Timisoara
Zerind
f=118+329
=447
f=75+374
=449
Arad
Fagaras
Oradea
Rimnicu
f=280+366
=646
f=239+178
=417
f=146+380
=526
Craiova
Pitesti
Sibiu
f=366+160
=526
f=317+98
=415
f=300+253
=553

Algoritmo A*: função de avaliação

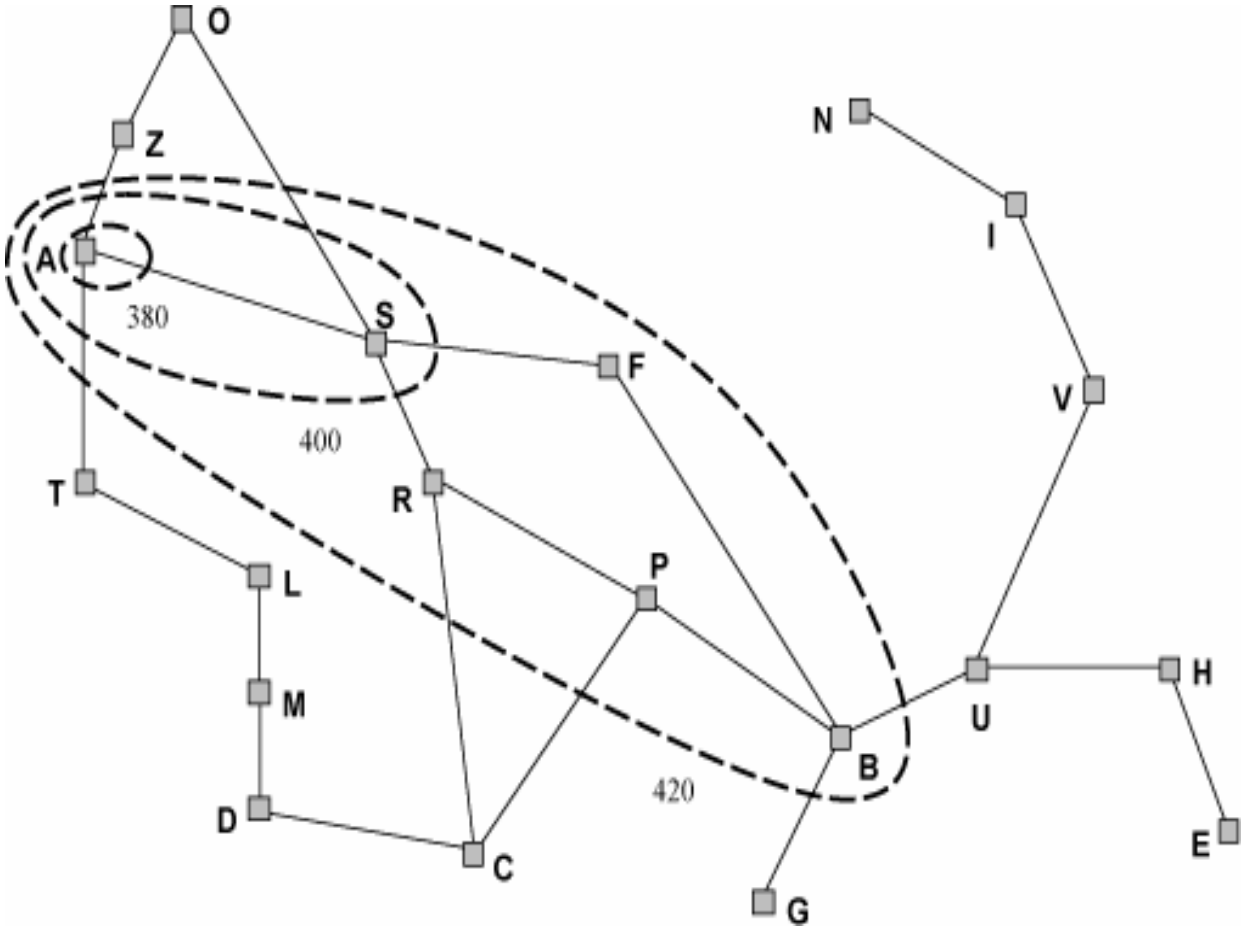
- A função heurística h é monotônica se
  - h (n) ≥ h (sucessor(n))
    - isso vale para a maioria das funções heurísticas
- Transferindo-se esse comportamento para a função de avaliação *f =g+h*, temos que
  - f (sucessor(n)) ≥ f(n)
    - uma vez que g é não decrescente
  - Em outras palavras
    - o custo de cada nó gerado no mesmo caminho nunca diminui
- Se *h* é não monotônica, para se garantir a monotonicidade de *f*, temos:
  - quando $f(suc(n)) < f(n)$
  - usa-se $f(suc(n)) = max\ (\ f(n),\ g(suc(n)) + h(suc(n))\ )$

Algoritmo A* : análise do comportamento

- A estratégia é completa e ótima
- Custo de tempo:
  - exponencial com o comprimento da solução, porém boas funções heurísticas diminuem significativamente esse custo
    - o fator de expansão fica próximo de 1
- Custo memória: $O(b^d)$
  - guarda todos os nós expandidos na memória
    - para possibilitar o backtracking
- Eficiência ótima
  - só expande nós com $f(n) \leq f^*$, onde f^* é o custo do caminho ótimo
    - f é não decrescente
  - nenhum outro algoritmo ótimo garante expandir menos nós

A* define Contornos

- **f(n) ≤ f* (f é admissível)**
- **fator de expansão próximo de 1**



Busca com Limite de Memória
Memory Bounded Search

- IDA* (Iterative Deepening A*)
  - igual ao aprofundamento iterativo, porém seu limite é dado pela função de avaliação (f) , e não pela profundidade (d).
  - necessita de menos memória do que A*

- SMA* (Simplified Memory-Bounded A*)
  - O número de nós guardados em memória é fixado previamente

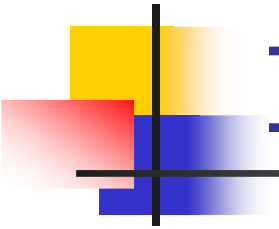
IDA* : Algoritmo

```
function IDA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: f-limit, the current f- COST limit
          root, a node

  root ← MAKE-NODE(INITIAL-STATE[problem])
  f-limit ← f- COST(root)
  loop do
      solution, f-limit ← DFS-CONTOUR(root, f-limit)
      if solution is non-null then return solution
      if f-limit = ∞ then return failure; end
```

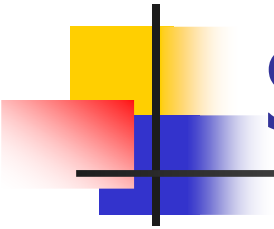
```
function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f- COST limit
  inputs: node, a node
          f-limit, the current f- COST limit
  static: next-f, the f- COST limit for the next contour, initially ∞

  if f- COST[node] > f-limit then return null, f- COST[node]
  if GOAL-TEST[problem](STATE[node]) then return node, f-limit
  for each node s in SUCCESSORS(node) do
      solution, new-f ← DFS-CONTOUR(s, f-limit)
      if solution is non-null then return solution, f-limit
      next-f ← MIN(next-f, new-f); end
  return null, next-f
```



IDA* : Custos

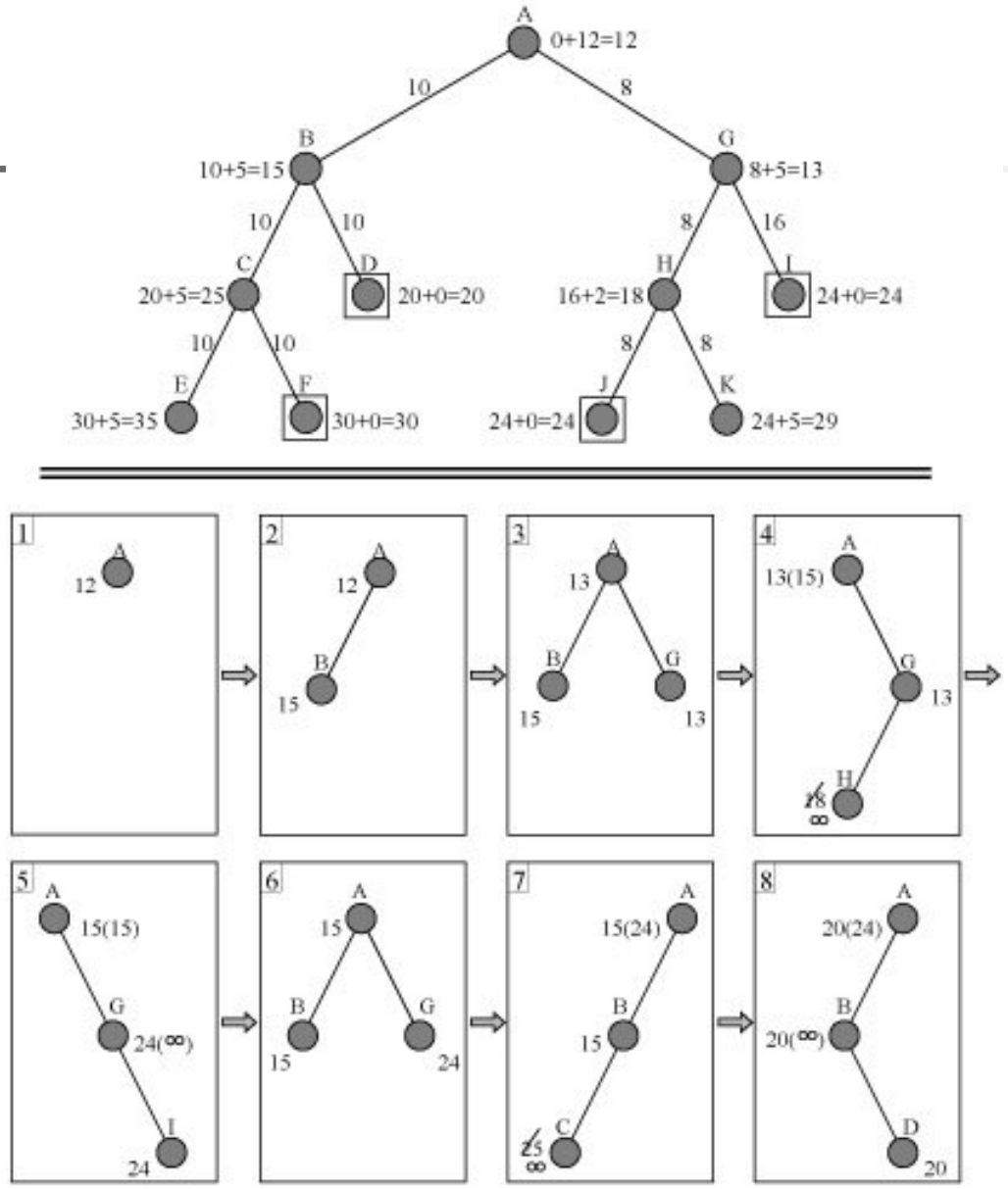
- O IDA* é completo e ótimo como o A*
- Se d é o custo por operação, f* é o custo da solução ótima e b é o fator de expansão, no pior caso temos um custo em memória de bf*/d
  - Na maioria dos casos bd é uma boa estimativa do custo
- A complexidade em tempo do IDA* depende fortemente do problema em análise
  - Em sistemas muitos complexos, o IDA* apresenta muita dificuldade.



SMA*

- Para problemas complexos o IDA* pode ter um custo de memória da ordem de N^2, onde n é o número de nós.
- A busca SMA* tem as propriedades:
  - Este utilizará toda a memória que lhe for disponível
  - Este permite repetir estados desde que a memória suporte
  - Este é completo se a memória permitir estocar a solução de rota mais rasa, se não, retornará a melhor solução que poder ser alcançada com a memória avaliada
  - Quando existe memória suficiente a busca é otimamente eficiente.

SMA* (Simplified Memory-Bounded A*)



SMA*: Algoritmo

```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
    if Queue is empty then return failure
    n ← deepest least-f-cost node in Queue
    if GOAL-TEST(n) then return success
    s ← NEXT-SUCCESSOR(n)
    if s is not a goal and is at maximum depth then
        f(s) ← ∞
    else
        f(s) ← MAX(f(n), g(s)+h(s))
    if all of n's successors have been generated then
        update n's f-cost and those of its ancestors if necessary
    if SUCCESSORS(n) all in memory then remove n from Queue
    if memory is full then
        delete shallowest, highest-f-cost node in Queue
        remove it from its parent's successor list
        insert its parent on Queue if necessary
    insert s on Queue
  end
```