Efficient memory-bounded search methods

Mikhail Simin – Arjang Fahim

CSCE 580: Artificial Intelligence – Fall 2011
Dr. Marco Voltorta

Outline of The Presentation

- **Motivations and Objectives**

- Background
  - BFS and DFS search
  - Depth-First Iterative-Deepening (DFID)

- IDA* (Iterative-Deepening A*)
  - A* heuristic search and its properties
  - IDA* algorithm

- SMA*
  - SMA* algorithm
  - SMA* properties
  - SMA* vs IDA*

- IE
  - IE algorithm

Motivations

- **Many problems in AI can be solved in theory by intelligently searching through many possible solutions.**
- **The performance of most AI systems is dominated by the complexity of a search algorithm.**
- **The standard algorithms, breadth-first and depth-first search both have limitations.**
- **Breadth-first search uses too much space.**
- **Depth-first search uses too much time and is not guaranteed to find a shortest path to a solution.**

Objectives

- **Using Iterative-Deepening algorithm in heuristic search such as A* and introducing IDA* is optimal along time, space and cost of solution.**
- **Advantage and Disadvantage of IDA***
- **Introducing memory bounded algorithms SMA* and IE**

Outline of The Presentation

- Motivations and Objectives

- **Background**
  - **BFS and DFS search**
  - **Depth-First Iterative-Deepening (DFID)**

- IDA* (Iterative-Deepening A*)
  - A* heuristic search and its properties
  - IDA* algorithm

- SMA*
  - SMA* algorithm
  - SMA* properties
  - SMA* vs IDA*

- IE
  - IE algorithm

- Conclusion

Background – BFS and DFS searches

- **Breadth-First-Search**
  - **BFS exhaustively searches entire graph by expanding nodes on each level until a goal state is reached.**

  - **In the worst case BFS expands all nodes up to depth *d* therefore the time complexity for BFS is $O(b^d)$ (*b* is branching factor) ($b + b^2 + b^3 + \ldots b^d$)**

  - **All nodes should be stored until their child nodes in the next level have been generated, therefore the space complexity is proportional to the number of nodes at the deepest level which is $O(b^d)$**
  - **In the real world BFS search will exhaust the available memory.**

Background – BFS and DFS searches

- **Depth-First-Search**
  - **DFS is an uniform search that works by generating descendant of the most recently expanded node, until some depth cutoff is reached and then backtracking to one of the most recently expanded node.**

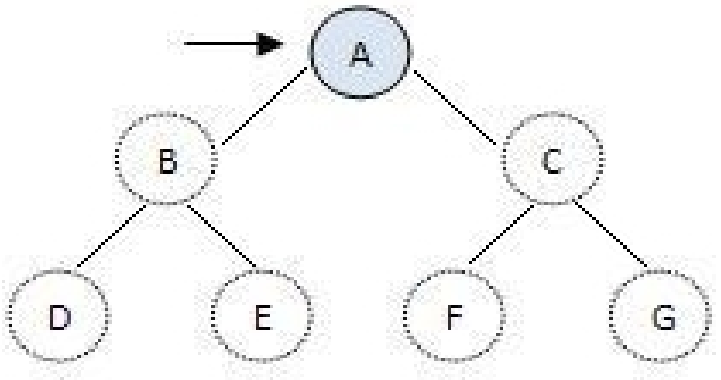
  - **In DFS only the path from initial node to the current node need to be stored. If the cutoff is *d*, the space requirement is *O(d)*.**

  - **The time complexity of depth first search is $O(e^d)$**
    e is branching factor

Depth-First Iterative-Deepening (DFID)

- **DFID starts performing depth-first search to the depth zero, then if that fails it searches to depth one, then depth 2, etc.**

- **DFID expands all nodes at a given depth, it is guaranteed to find a shortest-length solution**



- **The disadvantage of the DFID is that it performs wasted computation before reaching to the goal.**

Outline of The Presentation

- Motivations and Objectives

- Background
  - BFS and DFS search
  - Depth-First Iterative-Deepening (DFID)

- **IDA* (Iterative-Deepening A*)**
  - **A* heuristic search and its properties**
  - **IDA* algorithm**

- SMA*
  - SMA* algorithm
  - SMA* properties
  - SMA* vs IDA*

- IE
  - IE algorithm

IDA* (Iterative-Deepening A*)

- **A* algorithm**

  - A* uses best-first search and finds the least-cost path from a given initial node to the goal node (out of one or more possible goals)

  - It uses a distance-plus-cost heuristic function (denoted by *f(n)*)
  f(n) = g(n) + h(n)

  - Heuristic function should be admissible (not over estimate) and monotone

  - A* always finds a cheapest solution path if the heuristic is admissible

IDA* (Iterative-Deepening A*)

- **IDA* algorithm**
  - **IDA* uses the same idea of DFID algorithm and uses the admissibility of the heuristic function by limiting the *f-cost* of the nodes examined by the DFS search, rather than the depth**

Algorithm IDA*(n):
$limit \leftarrow f(n)$;
do until *success* or $limit$ unchanged
  $limit \leftarrow DFS(n, limit)$;

Function DFS(n,limit):
if $f(n) > limit$ return $f(n)$;
if $goal(n)$ then return with success
else return lowest value of $DFS(s, limit)$ for $s \in S(n)$.

S(n) denotes n's successors

IDA* (Iterative-Deepening A*)

- A* always finds a cheapest solution path if the heuristic is admissible , this property also holds for IDA* as well

- The worst case of the IDA* is that every node has a different *f-cost* in this case if A* examine k node to solve a problem, IDA* examines 1+2+3+...+k ($O(K^2)$) nodes

- IDA* (memory-bounded) algorithm does not keep any previously explored path (the same as A*). It needs to re-expand path if it is necessary and this will be a costly operation

Outline of The Presentation

- Motivations and Objectives

- Background
  - BFS and DFS search
  - Depth-First Iterative-Deepening (DFID)

- IDA* (Iterative-Deepening A*)
  - A* heuristic search and its properties
  - IDA* algorithm

- **SMA***
  - **SMA* algorithm**
  - **SMA* properties**
  - **SMA* vs IDA***

- IE
  - IE algorithm

Memory-bound A*

MA* is near-identical to A* aside from key differences:

- When the number of nodes in OPEN and CLOSED reaches some preset limit, MA* prunes the OPEN list by removing the leaf-node with highest f-cost.
- For each new successor the f-cost is propagated back up the tree.
- This keeps the tree very “informed” allowing the search to make better decisions, at the cost of some overhead.

Simplified MA*

1. Because the backing up of f-costs means that many nodes have the same f-cost, and because the algorithms need to select deepest and shallowest leaves of lowest and highest f-cost, SMA* uses a binary tree of binary trees to store OPEN, sorted by f and depth respectively. MA*'s data structures are less efficient.
2. SMA* is easier to implement and understand than MA*, since it maintains just two f-cost quantities for each node rather than four. Also, SMA* backs up once per fully-expanded node, rather than once per node generated.
3. When MA* begins pruning, it continues until only the current 'principal variations' — nodes with the best f-cost — remain. As argued above, only the worst node should be pruned, and only when space is needed for a better one. SMA* adds and prunes only one node at a time.
4. MA* loses information by not using pathmax with the backed-up f-costs. This is the most crucial improvement in SMA*.

SMA* Algorithm

Simplified Memory-Bounded A* can use of all available memory to carry out the search.

3.1 Algorithm description

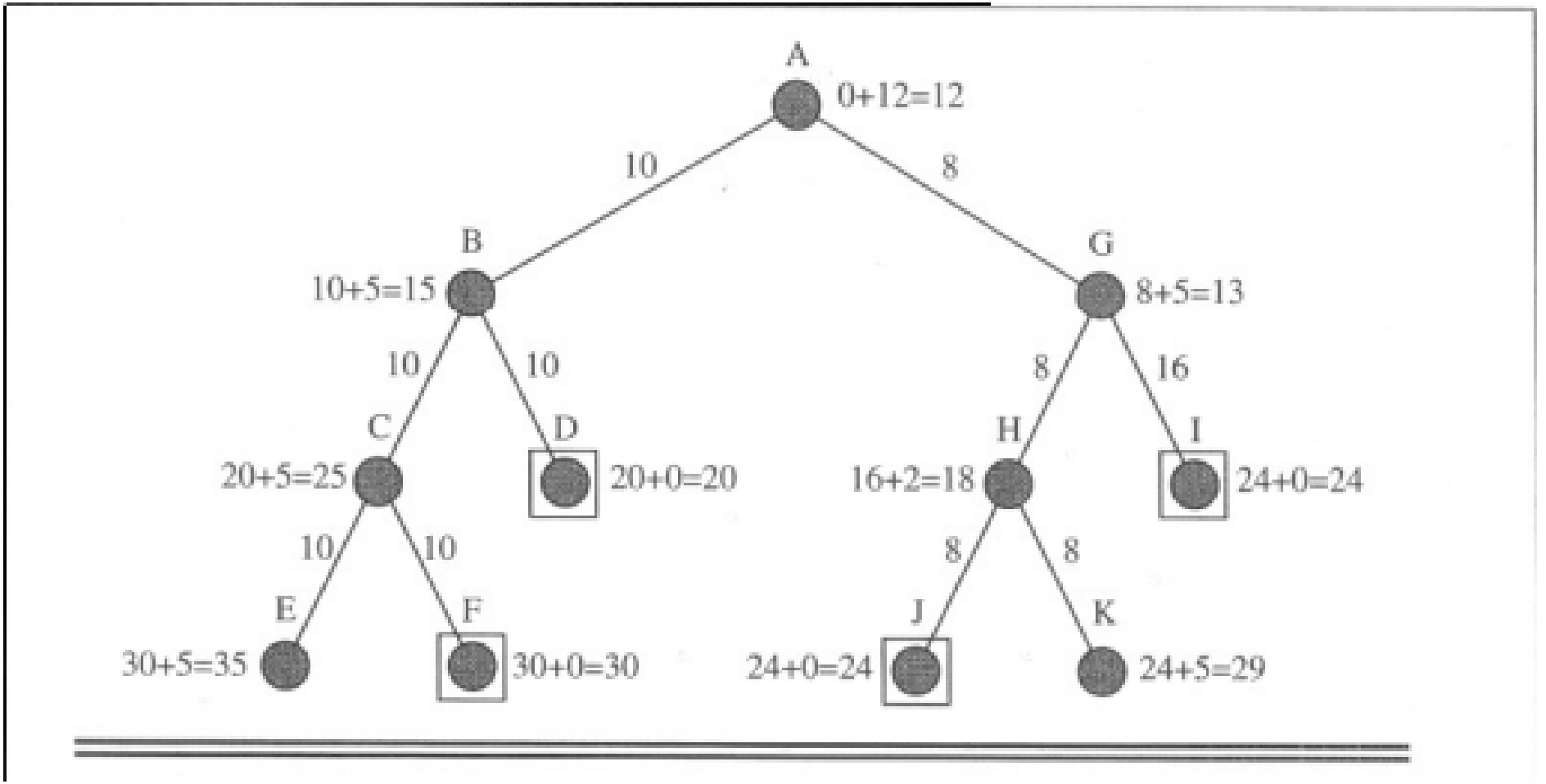
Algorithm SMA*(start):

put *start* on OPEN; USED ← 1;
loop
  if *empty*(OPEN) return with failure;
  best ← deepest least-f-cost leaf in OPEN;
  if *goal*(*best*) then return with success;
  succ ← *next-successor*(*best*);
  $f(succ) \leftarrow max(f(best), g(succ) + h(succ))$;
  if *completed*(*best*), $BACKUP(best)$;
  if $S(best)$ all in memory, remove *best* from OPEN.
  USED ← USED+1;
  if USED > MAX then
    delete shallowest, highest-f-cost node in OPEN;
    remove it from its parent's successor list;
    insert its parent on OPEN if necessary;
    USED ← USED-1;
  insert *succ* on OPEN.

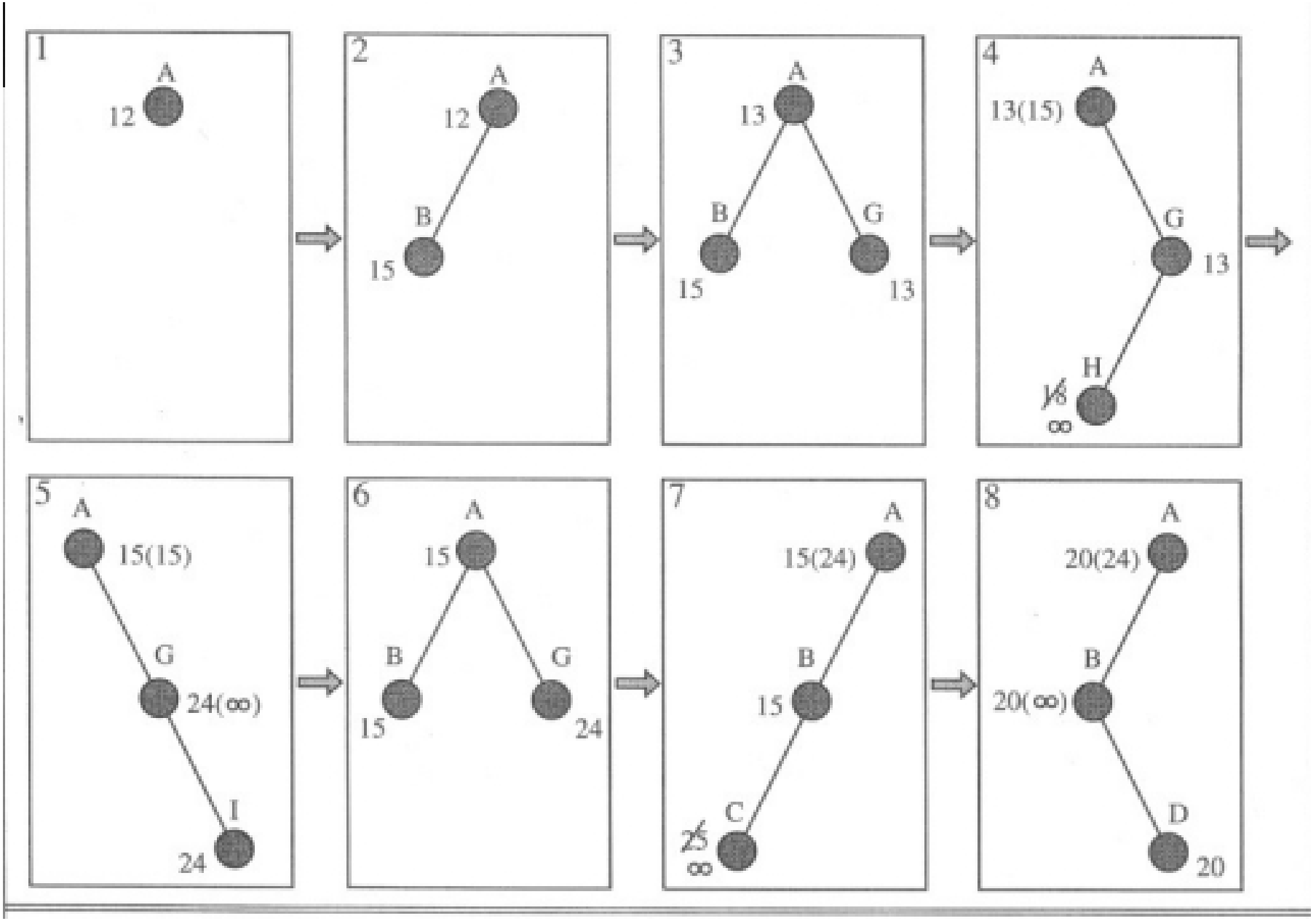
Procedure BACKUP(n):

if n is completed and has a parent then
  $f(n)$ ← least f-cost of all successors;
  if $f(n)$ changed, $BACKUP(parent(n))$.

SMA* Example


A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
D
H
I
20+5=25
20+0=20
16+2=18
24+0=24
10
10
8
8
E
F
J
K
30+5=35
30+0=30
24+0=24
24+5=29

SMA* Example


1
A
12
2
A
12
B
15
3
A
13
B
15
G
13
4
A
13(15)
G
13
H
∞
5
A
15(15)
G
24(∞)
I
24
6
A
15
B
15
G
24
7
A
15(24)
B
15
C
∞
8
A
20(24)
B
20(∞)
D
20

SMA* Algorithm

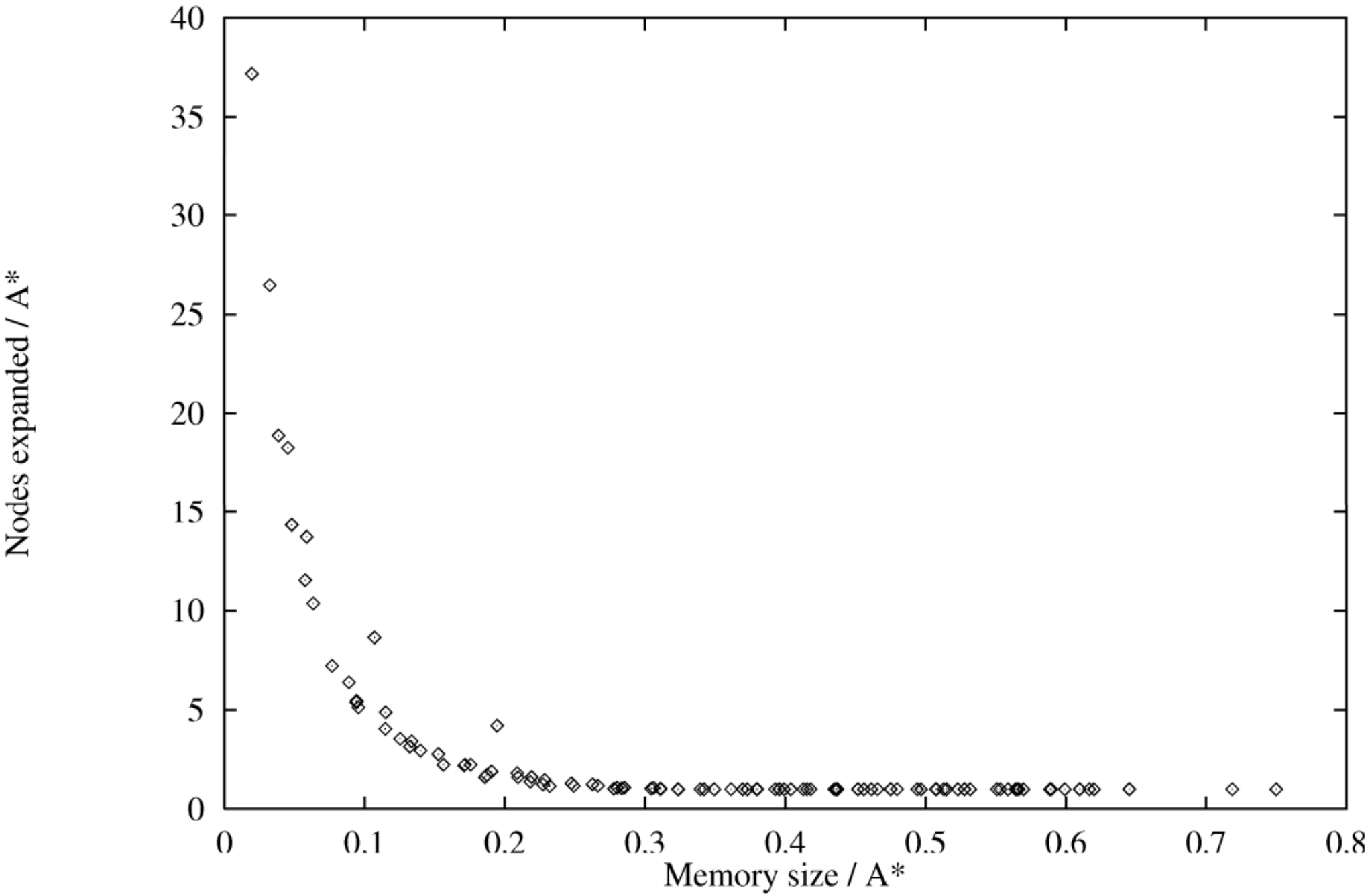
- It will utilize whatever memory is made available to it.

- It avoids repeated states as far as its memory allows.

- It is complete if the available memory sufficient to store the shallowest solution path.

- It is optimal if enough memory is available to store the shallowest optimal solution path. Otherwise, it returns the best solution that can be reached , with the available memory.

SMA* vs. A*


Nodes expanded / A*
Memory size / A*
0
5
10
15
20
25
30
35
40
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

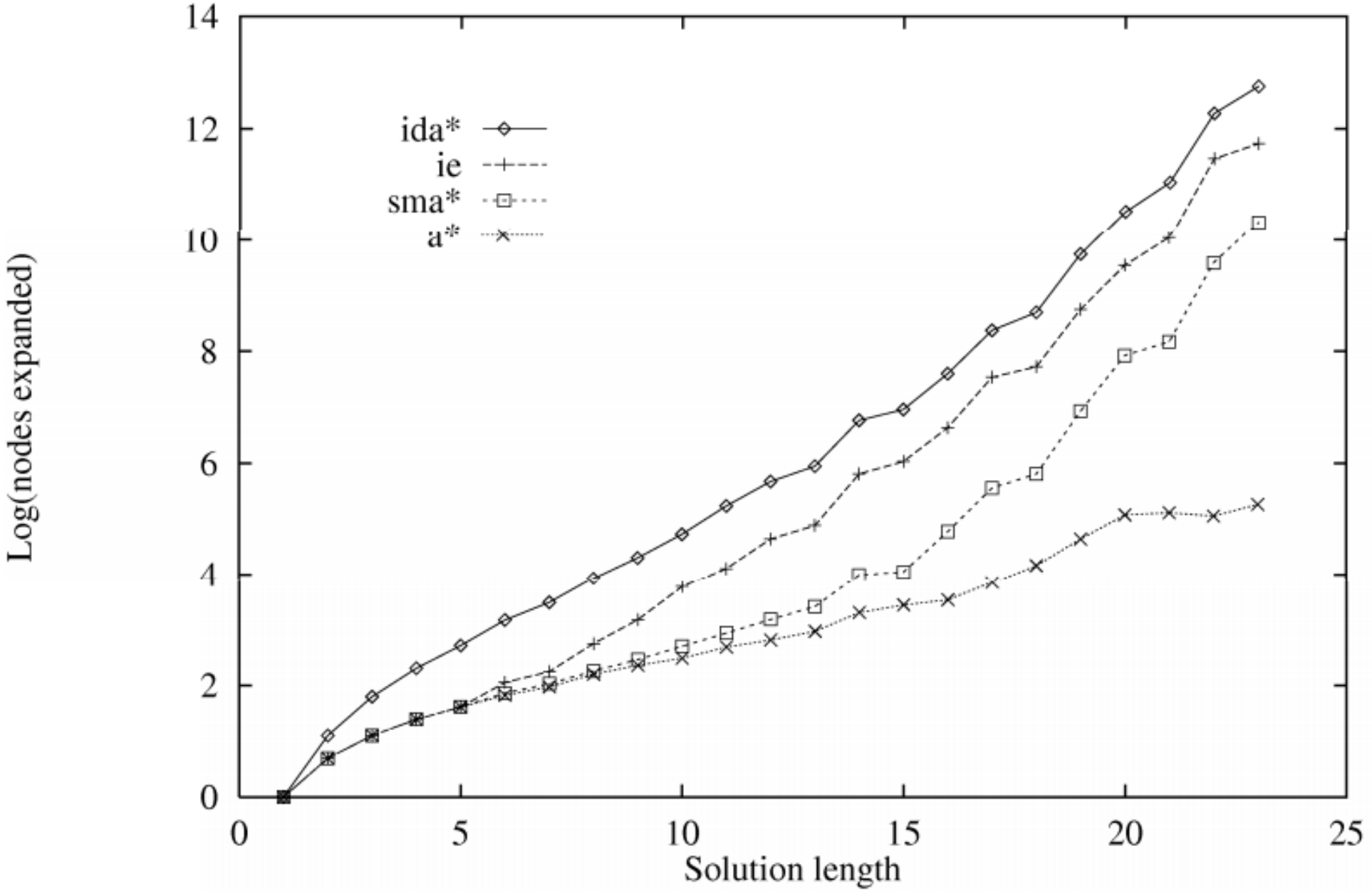
Iterative Expansion

- With an admissible heuristic the backed-up f-cost of a child of the root can only increase, and therefore search should only be carried out under the current best child, until the cost exceeds the current second best child.

- IE is called on a node with a bound equal to the backed-up f-cost of a second best path from any ancestors of that node.

Algorithm IE(n,bound):

if $f(n) > bound$ then return;
if $goal(n)$ then return with success;
generate $S(n)$, assign f-costs using pathmax;
if $S(n) = \{\}$ return ∞;
do until success or while $f(n) \leq bound$
    $best \leftarrow$ node in $S(n)$ with lowest f-cost;
    $newbound \leftarrow min(bound,$ other f-costs in $S(n))$;
    call $IE(best, newbound)$;
    $f(n) \leftarrow$ lowest f-cost in $S(n)$.


Log(nodes expanded)
Solution length
ida*
ie
sma*
a*
0
2
4
6
8
10
12
14
0
5
10
15
20
25

Questions and Answers

References

- **Depth-First Iterative-Deepening: An Optimal Admissible Tree Search***
Richard E. Korf
Depatment of Computer Science, Collumbia University,
New York, NY 10027, USA

- **Efficient memory-bounded search methods**
Stuart Russell
Computer Science Division University of California, Berkeley,
CA

- **Artificial Intelligence a modern approach –** *Second Edition*
Stuart Russell-Peter Norvig