15-Puzzle Problem

School of Data and Computer Science
Sun Yat-sen University

Overview

1. Search
2. IDA*
3. Tasks

A Search Problem

To formulate a problem as a search problem we need the following components:

- Formulate a **state space** over which to search. The state space necessarily involves abstracting the real problem.
- Formulate **actions** that allow one to move between different states. The actions are abstractions of actions you could actually perform.
- Identify the **initial state** that best represents your current state
- Identify the **goal** or **desired condition** one wants to achieve.
- Formulate various **heuristics** to help guide the search process.

The formalism

- Once the problem has been formulated as a state space search, various algorithms can be utilized to solve the problem.
- A solution to the problem will be a sequence of actions/moves that can transform your current state into state where your desired condition holds.

Algorithms for search

Inputs:

- a specified initial state (a specific world state or a set of world states representing the agent's knowledge, etc.)
- a successor function $S(x)$ = set of states that can be reached from state x via a single action.
- a goal test a function that can be applied to a state and returns true if the state is satisfies the goal condition.
- A step cost function $C(x,a,y)$ which determines the cost of moving from state x to state y using action a. ($C(x,a,y) = \infty$ if a does not yield y from x)

Output:

- a sequence of states leading from the initial state to a state satisfying the goal test.

Uninformed search strategies

- These are strategies that adopt a fixed rule for selecting the next state to be expanded.
- The rule does not change irrespective of the search problem being solved.
- These strategies do not take into account any domain specific information about the particular search problem.

Popular uninformed search techniques

- Breadth-First
- Uniform-Cost
- Depth-First
- Depth-Limited
- **Iterative-Deepening search**

Iterative deepening search

- Solve the problems of depth-first and breadth-first by extending depth limited search
- Starting at depth limit $L = 0$, we iteratively increase the depth limit, performing a depth limited search for each depth limit
- Stop if a solution is found, or if the depth limited search failed without cutting off any nodes because of the depth limit
- If no nodes were cut off, the search examined all paths in the state space and found no solution, hence no solution exists.

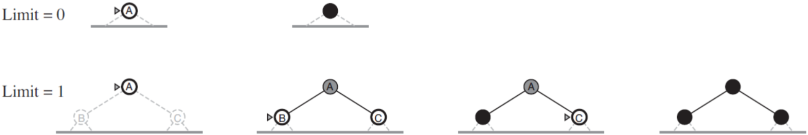
Iterative deepening search (cont'd)

Iterative deepening search

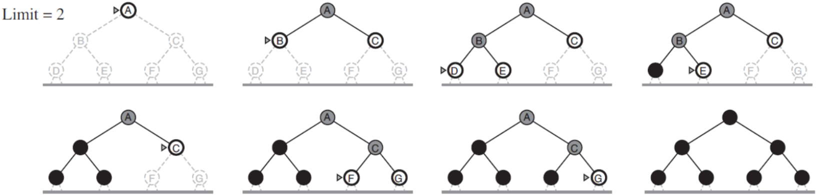
function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for $depth = 0$ **to** ∞ **do**
    $result \leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if $result \neq$ cutoff **then return** *result*

- The iterative deepening search algorithm, which repeatedly applies depth limited search with increasing limits.
- It terminates when a solution is found or if the depth limited search returns failure, meaning that no solution exists.

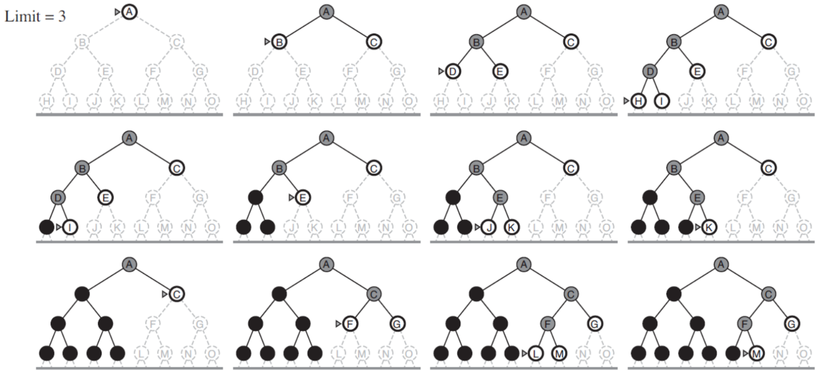
Iterative deepening search (cont'd)



Iterative deepening search (cont'd)



Iterative deepening search (cont'd)



Heuristic search

- The idea is to develop a domain specific heuristic function h(n), guessing the cost of getting to the goal from node n
- We require that $h(n) = 0$ for every node n whose state satisfies the goal
- There are different ways of guessing this cost in different domains. i.e., heuristics are domains specific.

A* search

- Define an evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ is the cost of the path to node n
  - $h(n)$ is the heuristic estimate of the cost of getting to a goal node from n
- So $f(n)$ is an estimate of the cost of getting to the goal via node n.
- We use $f(n)$ to order the nodes on the frontier.

IDA*

- A* has the same potential space problems as BFS or UCS
- IDA* - Iterative Deepening A* is similar to Iterative Deepening Search and similarly addresses space issues.
- Like iterative deepening, but now the cutoff is the f-value (g+h) rather than the depth
- At each iteration, the cutoff value is the smallest f-value of any node that exceeded the cutoff on the previous iteration

Pseudocode

```
path                //current search path (acts like a stack)
node                //current node (last node in current path)
g                   //the cost to reach current node
f                   //estimated cost of the cheapest path (root.. node.. goal)
h(node)             //estimated cost of the cheapest path (node.. goal)
cost(node, succ)    //step cost function
is_goal(node)       //goal test
successors(node)    //node expanding function, expand nodes ordered by g + h(node)
ida_star(root)      //return either NOT_FOUND
                    //or a pair with the best path and its cost

procedure ida_star(root)
    bound := h(root)
    path := [root]
    loop
        t := search(path, 0, bound)
        if t = FOUND then
            return (path, bound)
        if t = ∞ then
            return NOT_FOUND
        bound := t
    end loop
end procedure
```

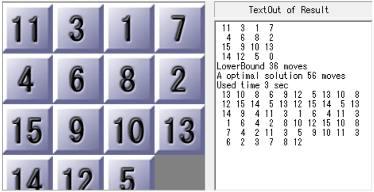
Pseudocode (cont'd)

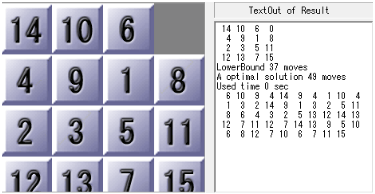
```
function search(path, g, bound)
    node := path.last
    f := g + h(node)
    if f > bound then
        return f
    if is_goal(node) then
        return FOUND
    min := ∞
    for succ in successors(node) do
        if succ not in path then
            path.push(succ)
            t := search(path, g + cost(node, succ), bound)
            if t = FOUND then
                return FOUND
            if t < min then
                min : =t
            path.pop()
        end if
    end for
    return min
end function
```

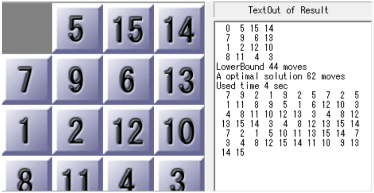
Tasks

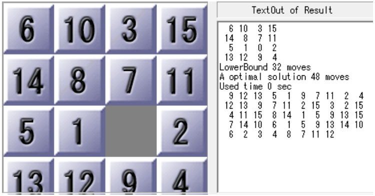
- Please solve 15-Puzzle problem by using IDA* (MATLAB).
- You can use one of the two commonly used heuristic functions:
  - h1 = the number of misplaced tiles.
  - h2 = the sum of the distances of the tiles from their goal positions.
- Here are 4 test cases for you to verify your algorithm correctness.

Tasks (cont'd)


11 3 1 7
4 6 8 2
15 9 10 13
14 12 5
TextOut of Result
11 3 1 7
4 6 8 2
15 9 10 13
14 12 5 0
LowerBound 36 moves
A optimal solution 56 moves
Used time 3 sec
13 10 8 6 9 12 5 13 10 8
12 15 14 5 13 12 15 14 5 13
14 9 4 11 3 1 6 4 11 3
1 6 4 2 8 10 12 15 10 8
7 4 2 11 3 5 9 10 11 3
6 2 3 7 8 12


14 10 6
4 9 1 8
2 3 5 11
12 13 7 15
TextOut of Result
14 10 6 0
4 9 1 8
2 3 5 11
12 13 7 15
LowerBound 37 moves
A optimal solution 49 moves
Used time 0 sec
6 10 9 4 14 9 4 1 10 4
1 3 2 14 9 1 3 2 5 11
8 6 4 3 2 5 13 12 14 13
12 7 11 12 7 14 13 9 5 10
6 8 12 7 10 6 7 11 15


5 15 14
7 9 6 13
1 2 12 10
8 11 4 3
TextOut of Result
0 5 15 14
7 9 6 13
1 2 12 10
8 11 4 3
LowerBound 44 moves
A optimal solution 62 moves
Used time 4 sec
7 9 2 1 9 2 5 7 2 5
1 11 8 9 5 1 6 12 10 3
4 8 11 10 12 13 3 4 8 12
13 15 14 3 4 8 12 13 15 14
7 2 1 5 10 11 13 15 14 7
3 4 8 12 15 14 11 10 9 13
14 15


6 10 3 15
14 8 7 11
5 1 2
13 12 9 4
TextOut of Result
6 10 3 15
14 8 7 11
5 1 0 2
13 12 9 4
LowerBound 32 moves
A optimal solution 48 moves
Used time 0 sec
9 12 13 5 1 9 7 11 2 4
12 13 9 7 11 2 15 3 2 15
4 11 15 8 14 1 5 9 13 15
7 14 10 6 1 5 9 13 14 10
6 2 3 4 8 7 11 12

More...

- This program uses IDA* as the basic search algorithm, and this uses WD(Walking Distance) to improve the efficiency of the search.
- WD is a sophisticated lower bound for how many moves are needed to solve an arbitrary board configuration.
- WD gives severe distance than MD(Manhattan Distance).
- As for the details of WD, please read `http://www.ic-net.or.jp/home/takaken/nt/slide/solve15.html`

Reference

- Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach.
- IDA*, Wikipedia.
- "15puzzle Optimal solver" for windows, `http://www.ic-net.or.jp/home/takaken/e/15pz/index.html`

The End