

# Einführung in Artificial Intelligence SS 2024, 4.0 VU, 192.027

## Exercise Sheet 2 – Problem Solving and Search

For the discussion part of this exercise, mark and upload your solved exercises in **TUWEL** until Wednesday, June 5, 23:55 CEST. The registration for a solution discussion ends on Friday, June 7, 23:55 CEST. Be sure that you tick only those exercises that you can solve and explain!

In the discussion, students will be asked questions about their solutions of examples they checked. The discussion will be evaluated with 0–25 points, which are weighted with the fraction of checked examples and rounded to the next integer. There is *no minimum number of points* needed for a positive grade (i.e., you do not need to participate for a positive grade, but you can get at most  $\approx 80\%$  without doing exercises).

Note, however, that *your registration is binding*. Thus, if you register for a solution discussion, then it is *mandatory* to show up. No-show after a registration without plausible excuse leads to a penalty. Such students will not have the possibility to participate in another exam once they have gotten a certificate. If you registered but cannot show up due to unpredictable and unavoidable obstacles, either deregister or send us a confirmation (doctor's note, etc) and you will be excused. Please ask questions in the **TUWEL** forum or visit our tutors during the tutor hours (see **TUWEL**).

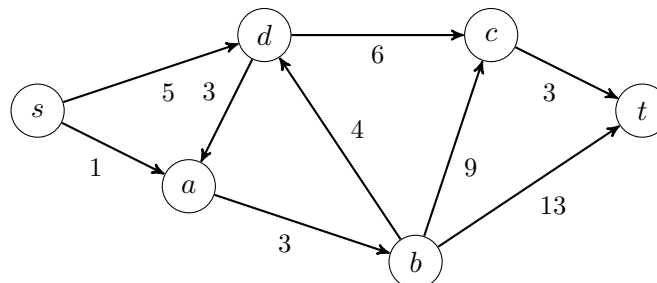
---

**Exercise 2.1:** Consider again the 8-Puzzle discussed in the lecture. Consider the discussed heuristics

- (a)  $h_1(n)$ : number of misplaced tiles,
- (b)  $h_2(n)$ : Manhattan distance.

Show whether the two suggested heuristics are admissible and/or consistent (monotonic).

**Exercise 2.2:** Perform the greedy algorithm and the A\* algorithm using the given heuristic function  $h$  on the following graph in order to find a shortest path from  $s$  to  $t$ . In which order are the nodes expanded? Show the contents of the priority queue at each iteration. If multiple nodes have the same priority, expand the one that comes first alphabetically. Compare the shortest paths returned by the two algorithms, argue whether these solutions are optimal? Justify your answer.



$$h(s) = 14, h(a) = 7, h(b) = 6, h(c) = 2, h(d) = 8, h(t) = 0$$

**Exercise 2.3:** Assume  $h_a, h_b$  are admissible heuristics. Which of the following heuristics is admissible? Justify your answer.

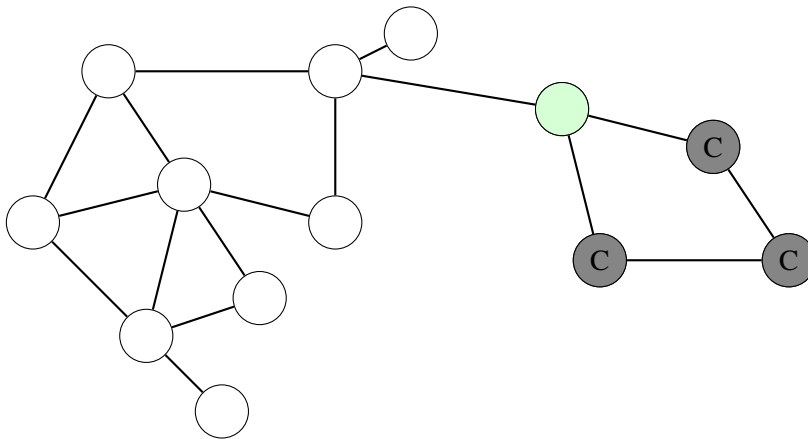
- (a)  $h_1(n) = \frac{h_a(n)}{h_b(n)}$ .
- (b)  $h_2(n) = 2^{(h_a+h_b)/c} - 1 \quad c > 0$

**Exercise 2.4:** Let  $f(n) = c \cdot g(n) + d \cdot h(n)$  be an evaluation function, where  $c$  and  $d$  are constants.

- (1) Define  $c, d, h(\cdot), g(\cdot)$  such that A\* with this evaluation function acts as a breadth-first search.
- (2) Define  $c, d, h(\cdot), g(\cdot)$  such that A\* with this evaluation function acts as a depth-first search.
- (3) Define  $c, d, h(\cdot), g(\cdot)$  such that A\* with this evaluation function acts as a uniform cost search.

You may assume that nodes contain all the information that we discussed in the lecture.

**Exercise 2.5:** Darth Vader has recently conquered a new sector of the galaxy. However, he now faces the problem that the Emperor has only provided him with 3 mobile command centers to secure the planets in this area. A planet is considered secure once either a command center is located on it or a neighboring planet has a command center. The hyperspace routes connecting the planets are depicted as edges in the following graph, with the planets as nodes. Darth Vader randomly distributed the command centers among the planets. Now it's your turn to execute a hill-climbing local search to find a goal state where each planet is covered. A step in the search process involves transferring one command center from one planet to another, the number of secured planets represents the value of each state. *Note: A command center can be transferred to any planet during a step, however only one can be transferred at a time.*



*Covered Planets: 4*

**Exercise 2.6:** Decide and explain which of the following statements are true and which are false? Back up your answers with proofs or counterexamples.

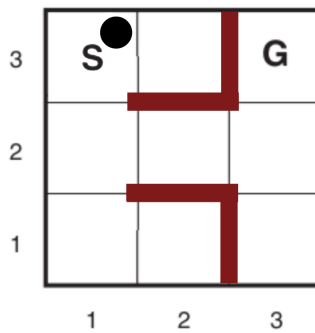
- (1) Consider the basic implementation of local beam search as discussed in the lecture. A local search using this approach always outperforms  $k$  individual local searches running basic hill-climbing.
- (2) Consider the basic implementation of the genetic algorithm as discussed in the lecture. To streamline the algorithm, one may choose to skip the mutation step at the end. This decision may prolong the computation time, but it is still possible for the algorithm to cover the entire search space.

**Exercise 2.7:** Consider the vacuum-cleaner world example discussed in the lecture. This time, a mischievous puppy is introduced into the world, presenting some challenges for our little robot friend. The following nondeterministic action effects have been incorporated into the base world:

- (1) *Playmate*. The puppy sees the robot as its playmate, which introduces the possibility of our robot getting pushed around *after* it vacuums a space. For instance, the "Suck" action in the state [A, Dirt, Dirt] results in either [A, , Dirt] or [B, , Dirt]. *Note that the puppy pushes the robot, after it has already cleaned the space.*
- (2) *Dog Hair*. As with most dogs, the puppy also sheds, which is why hair can get caught in the robot's tires if it moves without having vacuumed beforehand. Whenever the robot takes a "Move" action and moves away from a space that contains dirt, it is possible that the robot will spread the dirt on the new space. For example, the "Move-Left" action in the state [B, , Dirt] results in either [A, , Dirt] or [A, Dirt, Dirt].

Decide whether this search problem with nondeterministic action effects can be solved with an *And-Or Tree* or if a *Cyclic Solution* is required. Illustrate the corresponding solution visually.

**Exercise 2.8:** Perform the Online Search algorithm, as outlined in the lecture, on the following maze. Show the agent's location in the maze for each state along with the next move and the updates made to the *untried*, *result*, and *unbacktracked* arrays. Note that the preferred moves are in the order of Up (U), Right (R), Left (L), and Down (D).



## 2.1

### Definition 1

A heuristic  $h$  is **admissible**, if for every node  $n$  the following holds:

1.  $h(n) \leq h^*(n)$  where  $h^*(n)$  is the **true** cost from  $n$  ( $h$  is "optimistic");
2.  $h(n) \geq 0$ ;
3.  $h(g) = 0$  for every goal  $g$  (follows from 1 and 2).

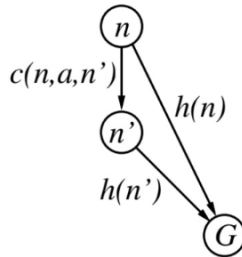
### Optimality of A\* using graph search

#### Definition 3

A heuristic is **consistent** if, for every node  $n$ , every successor  $n'$  of  $n$  and every operator  $a$ ,

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

holds, where  $c(n, a, n')$  are the path costs for  $a$ .



#### Lemma 4

If  $h$  is consistent, then  $f$  is non-decreasing along every path:

$$\begin{aligned}
 f(n') &= g(n') + h(n') \\
 &= g(n) + c(n, a, n') + h(n') \\
 &\geq g(n) + h(n) = f(n).
 \end{aligned}$$

#### Theorem 5

If  $h$  is **consistent**, then A\* using **graph search** is optimal.

## 2.1a

### Admissibility:

$h_1$  must be admissible, because it is a lower bound for  $h^*(n)$ . Each tile that is misplaced must be moved at least once (cost per move = 1). The minimal distance to move a tile to its correct position is the Manhattan Distance so we know that:

$$h_1(n) \leq h_2(n) \quad (h_2(n) \text{ dominates } h_1(n))$$

Because  $h_2(n)$  is admissible,  $h_1(n)$  must also be  
see 2.1b

### Consistency:

Cost for a move  $c(n, m) = 1$  from state  $m$  to  $n$

Best-case scenario the move  $c(n, m)$  reduces the number of incorrect tiles by 1 (our heuristic)

$x$ ... number of incorrect tiles

$$h(n) = x$$

$n \rightarrow m$  via  $c(n, m) = 1$  gives us  $h(m) = x - 1$  (no change in overall cost)

$$h(n) \leq c(n, m) + h(m) \quad \left. \begin{array}{l} \text{substitute with } x \\ \end{array} \right\}$$

$$x \leq 1 + x - 1$$

$$x \leq x \quad \checkmark$$



2.7b

### Admissibility:

$h_2$  must be admissible, because the Manhattan Distance is a lower bound for the real cost of the 8-Puzzle.

$$\underbrace{h^*(n)}_{\text{real-cost}} = \underbrace{h_2(n)}_{\text{minimal cost to move tiles to destination}} + \underbrace{g^*(n)}_{\text{cost to move other tiles out of the way}}$$

In a best-case scenario there are no other tiles to move so our true cost  $h^*(n)$  is exactly the Manhattan distance, as it is the shortest path.

Best-case:  $h^*(n) = h_2(n) + g^*(n)$ ,  $g^*(n) = 0$ , because of no obstacles

$$h^*(n) = h_2(n) \checkmark$$

Other-case:

$$h^*(n) = h_2(n) + g^*(n), \quad g^*(n) \geq 0$$

$$h_2(n) = h^*(n) - g^*(n)$$

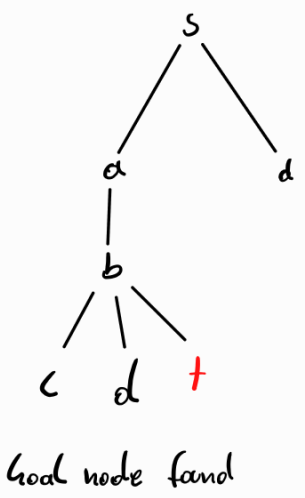
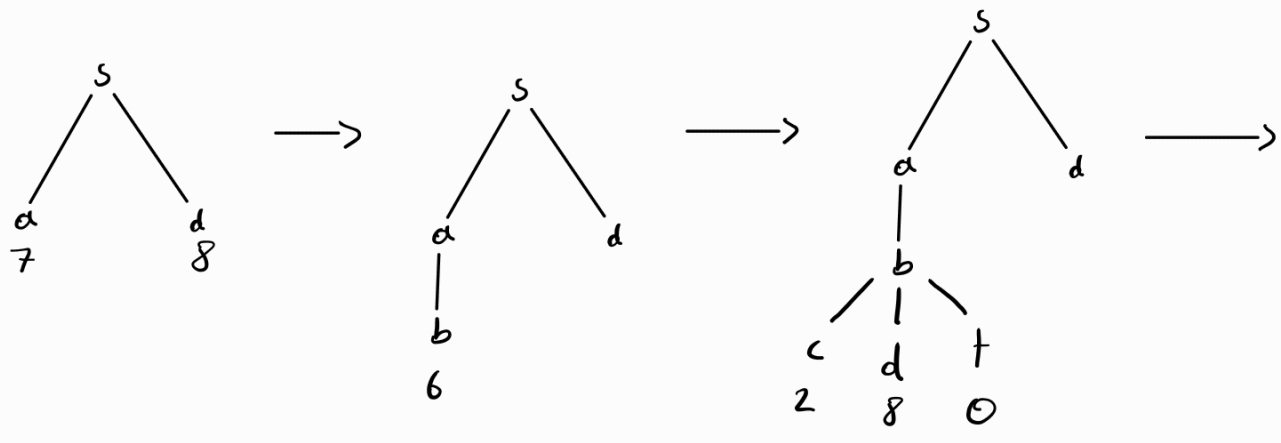
$$h_2(n) \leq h^*(n) \checkmark$$

### Consistent:

Same as 2.7a because each move to another state  $h(m)$  can only decrement the overall Manhattan distance by 1, because tiles can only be moved 1 square  $\leftrightarrow$

so best-case  $h(m) = h(n) - 1$

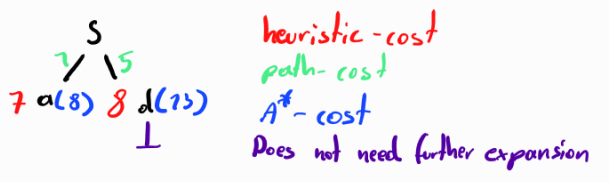
Greedy-Algorithm:



path = s → a → b → t  
 red-path-cost = 17

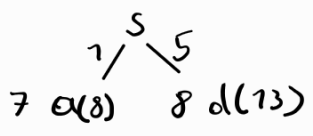
A\* - Algorithm:

Legend:



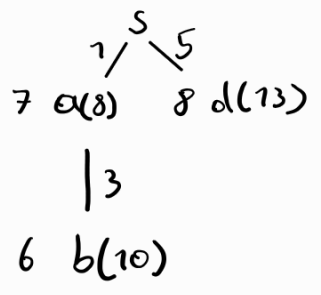
Visited = {s(14)}

Queue = a(8), d(13)



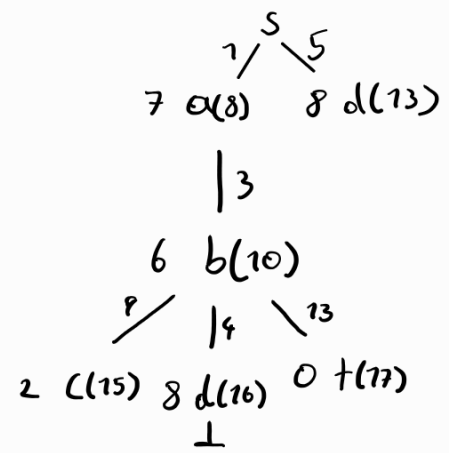
Visited = {s(14), a(8)}

Queue = b(10), d(13)



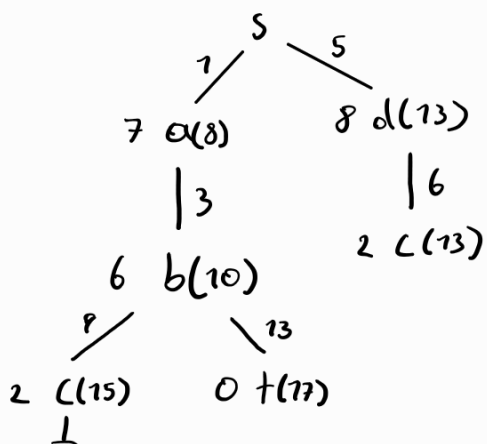
Visited = {s(14), a(8), b(10)}

Queue = d(13), c(15), t(17)



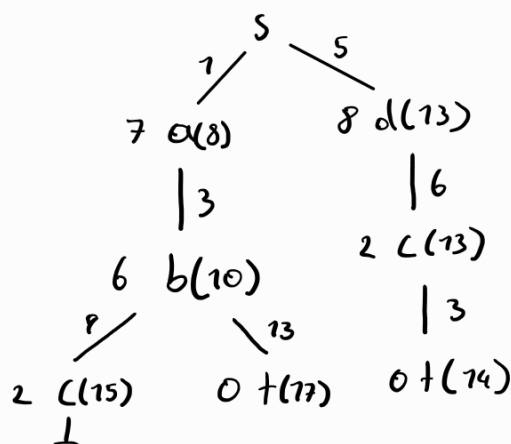
Visited = {s(14), a(8), b(70), d(13)}

Queue = c(13), t(17)



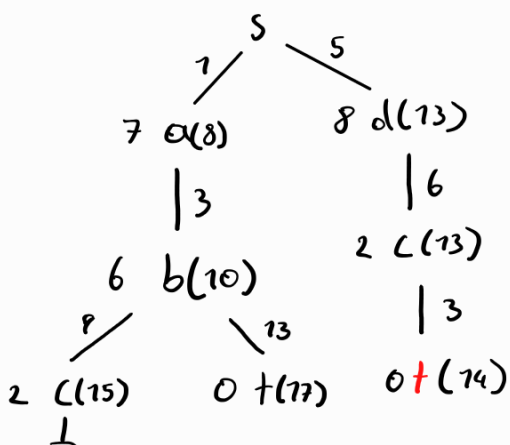
Visited = {s(14), a(8), b(70), d(13), c(13)}

Queue = t(14)



Visited = {s(14), a(8), b(70), d(13), c(13), t(14)}

Queue = t(14)



Goal state t visited  
 path = s → d → c → t  
 path-cost = 5 + 6 + 3 = 14

The solution found via A\* is optimal and can be verified using Dijkstra's algorithm, but optimality is in general not guaranteed.

Justification:

A\*-Search in a graph is optimal if the heuristic used is admissible and consistent.

Admissibility: ✓  
 s → t = 14    h(s) = 14 ✓  
 a → t = 15    h(a) = 7 ✓  
 b → t = 12    h(b) = 6 ✓  
 c → t = 3    h(c) = 2 ✓  
 d → t = 9    h(d) = 8 ✓

Consistency: X  
 The triangle inequality  $h(n) \leq C(n, a, m) + h(m)$  is not satisfied for  
 $h(s) \leq C(s, a) + h(a)$   
 $14 \leq 1 + 7$   
 $14 \leq 8$  X  
 optimality is not guaranteed!

Heuristic never over-estimates

2.3a

$$h_1(n) = \frac{h_a(n)}{h_b(n)}$$

Not admissible, because  $h_b(n) = 0$ , causes  $h_1(n)$  to be undefined.

if  $h_b(n) \neq 0$ :

Admissible, because  $h_1(n) \leq h_a(n)$ .  $h_a(n)$  dominates  $h_1(n)$ , because  $h_a(n)$  is admissible per specification

2.3b

$$h_2(n) = 2^{\frac{h_a + h_b}{c}} - 1, \quad c > 0$$

Not admissible, as  $c$  could be  $c \in (0, 1)$ . This makes it possible for  $h_2(n)$  to be significantly larger than  $h_a$  or  $h_b$ .

2.4

$A^*$ : Use evaluation function  $f(n) = g(n) + h(n)$   
(and avoid expanding paths that are already expensive)

- $g(n)$ : path costs from start to  $n$ , (i.e., costs so far up to  $n$ )
- $h(n)$ : estimated cost to goal from  $n$  (like in greedy search)
- $f(n)$ : estimated total cost of path through  $n$  to goal

2.4a

$$c = 0$$

$$d = 1$$

$h(n)$  = distance from node  $n$  to start node  $S$ .

↳ always chose node with minimal  $h(n)$ . This searches in layers (like BFS)

$$f(n) = h(n) \cdot d = h(n)$$

2.4b

$$c = 0$$

$$d = 1$$

$h(n)$  = distance from node  $n$  to goal node

$$f(n) = h(n) \cdot d = h(n)$$

2.4c

$$c = 1$$

$$h = 0$$

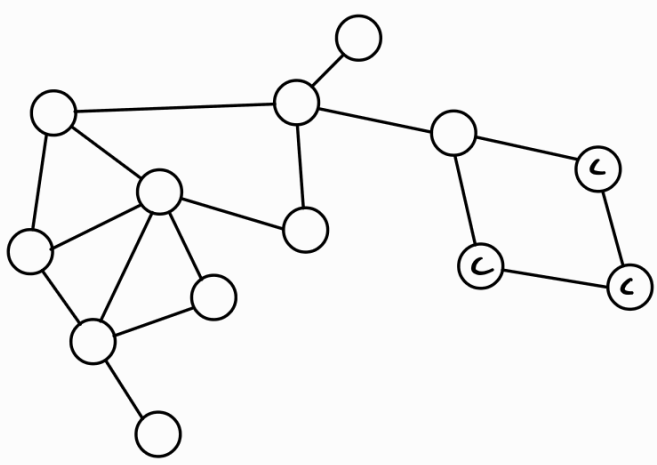
$g(n)$  = path cost from start node to  $n$ .

$$f(n) = g(n)$$

2.5

Initial state

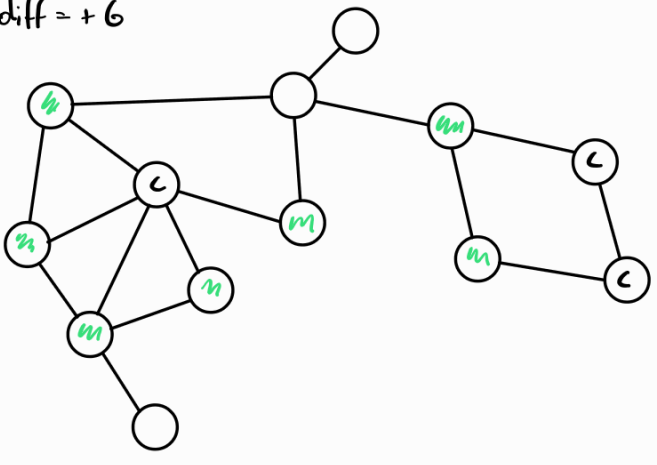
covered nodes = 4



Iteration 1:

covered nodes = 10

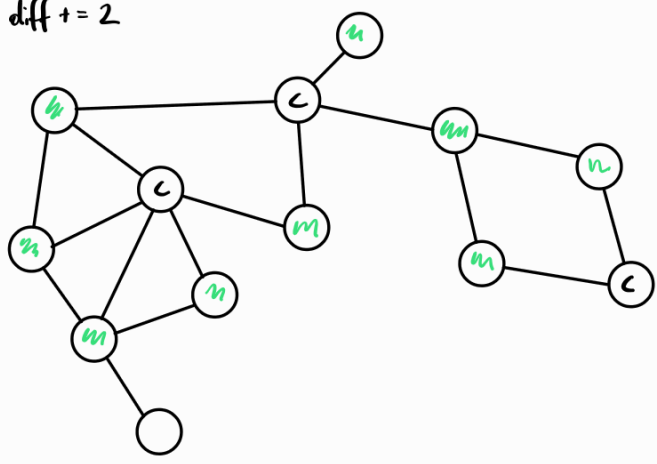
diff = +6



Iteration 2:

covered nodes = 12

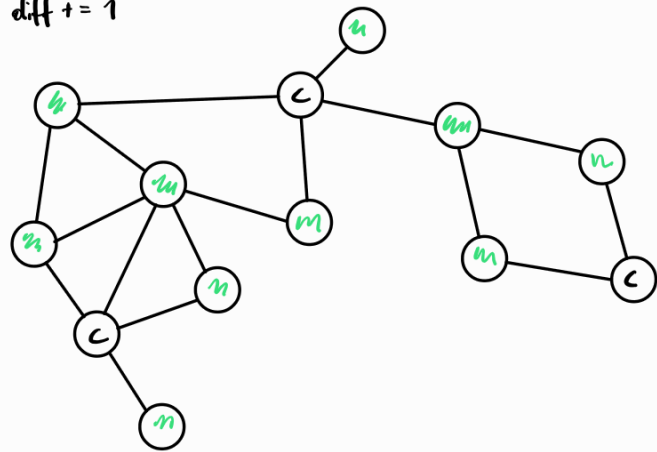
diff = +2



Iteration 3

covered nodes = 13

diff += 1



Local maxima reached  $\rightarrow$  terminate

2.6

(1)

FALSE:

- if  $k=1$  then local beam search and hill climbing are identical, thus one cannot outperform the other

- Also: it is possible for a hill-climbing search to find a better solution than local beam search, if the search space is very rugged (many peaks, valleys, cliff, etc)

This is because one of the  $k$ -hill climbing searches might start in a state which might look unpromising but be close to the global maxima. The beam search is almost guaranteed to not traverse such a path.

(2)

In general FALSE:

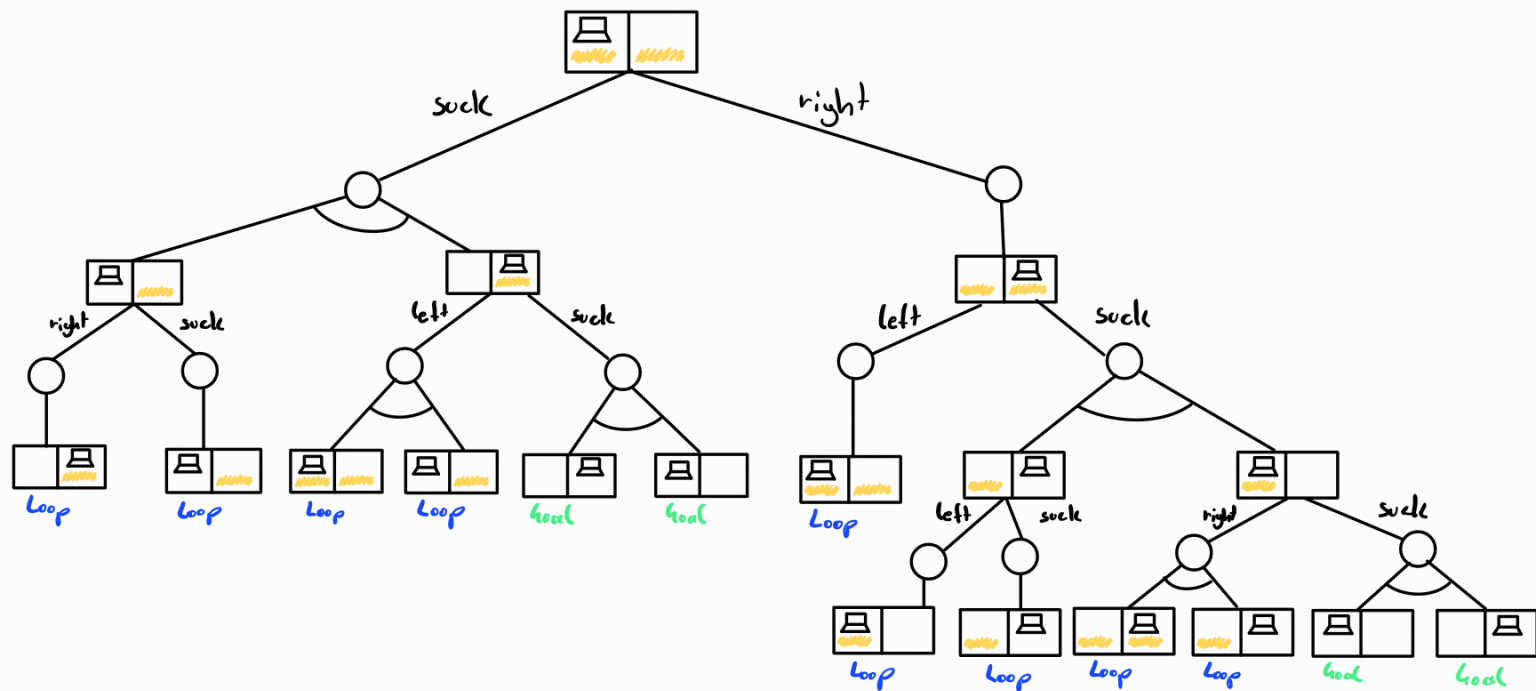
In general though this is not the cause, because the number of combinations via crossover is limited. (needed search space might be larger)

TRUE, in some scenarios:

It is possible, but highly unlikely. Because of the crossover step it is possible, if the combinations match the criteria needed to cover the entire search space.



(1) + (2)





2.8


```

function ONLINE-DFS-AGENT(problem, s') returns an action
    s, a, the previous state and action, initially null
    result, a table mapping (s, a) to s', initially empty
    untried, a table mapping s to a list of untried actions
    unbacktracked, a table mapping s to a list of states never backtracked to

if problem.IS-GOAL(s') then return stop
if s' is a new state (not in untried) then untried[s'] ← problem.ACTIONS(s')
if s is not null then
    result[s, a] ← s'
    add s to the front of unbacktracked[s']
if untried[s'] is empty then
    if unbacktracked[s'] is empty then return stop
    a ← an action b such that result[s', b] = POP(unbacktracked[s'])s' ← null
else a ← POP(untried[s'])
    s ← s'
return a
    
```

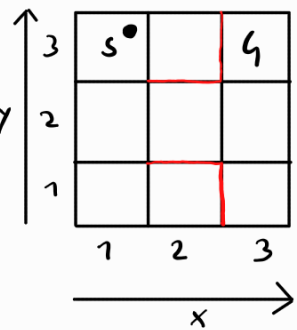
preferred moves order: U > R > L > D

Legend:

$s_{xy} = \begin{cases} \text{state } s \text{ at coordinates} \\ x \\ y \end{cases} \Rightarrow s_{23} =$ 


①

next action = R



untried:

state <i>s</i>	untried actions
$s_{13}$	R, D

unbacktracked:

state <i>s</i>	states never backtracked to

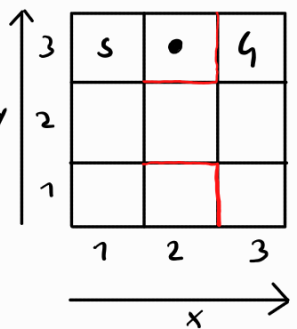
result:

state <i>s</i>	action <i>a</i>	resulting state <i>s'</i>

②

next action = L

(always take from untried actions if possible, only take from backtracked if untried actions [*s'*] = {})



untried:

state <i>s</i>	untried actions
$s_{13}$	D
$s_{23}$	L

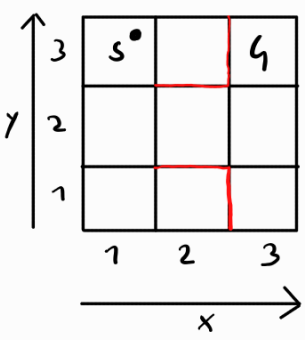
unbacktracked:

state <i>s</i>	states never backtracked to
$s_{23}$	$s_{13}$

result:

state <i>s</i>	action <i>a</i>	resulting state <i>s'</i>
$s_{13}$	R	$s_{23}$

③  
next action = D



untried:

state s	untried actions
S <sub>13</sub>	D
S <sub>23</sub>	∅

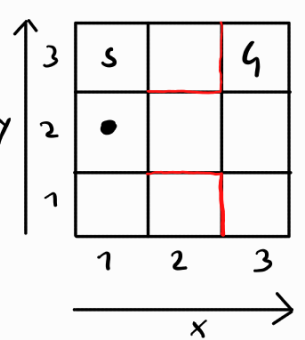
unbacktracked:

state s	states never backtracked to
S <sub>23</sub>	S <sub>13</sub>
S <sub>13</sub>	S <sub>23</sub>

result:

state s	action a <sub>1</sub>	resulting state s'
S <sub>13</sub>	R	S <sub>23</sub>
S <sub>23</sub>	L	S <sub>13</sub>

④  
next action = U



untried:

state s	untried actions
S <sub>13</sub>	∅
S <sub>23</sub>	∅
S <sub>12</sub>	U, R, D

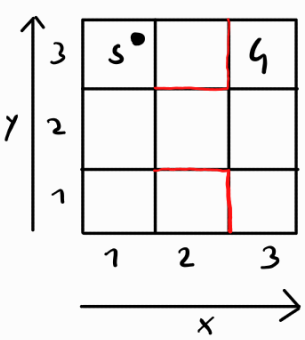
unbacktracked:

state s	states never backtracked to
S <sub>23</sub>	S <sub>13</sub>
S <sub>13</sub>	S <sub>23</sub>
S <sub>12</sub>	S <sub>13</sub>

result:

state s	action a <sub>1</sub>	resulting state s'
S <sub>13</sub>	R	S <sub>23</sub>
S <sub>23</sub>	L	S <sub>13</sub>
S <sub>13</sub>	D	S <sub>12</sub>

⑤  
next action = D (Backtrack to S<sub>12</sub>)



untried:

state s	untried actions
S <sub>13</sub>	∅
S <sub>23</sub>	∅
S <sub>12</sub>	R, D

unbacktracked:

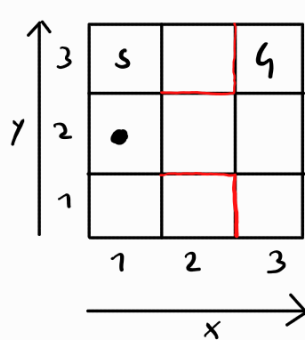
state s	states never backtracked to
S <sub>23</sub>	S <sub>13</sub>
S <sub>13</sub>	S <sub>12</sub> , S <sub>23</sub>
S <sub>12</sub>	S <sub>13</sub>

result:

state s	action a <sub>1</sub>	resulting state s'
S <sub>13</sub>	R	S <sub>23</sub>
S <sub>23</sub>	L	S <sub>13</sub>
S <sub>13</sub>	D	S <sub>12</sub>
S <sub>12</sub>	U	S <sub>13</sub>

Lookup result table for action

⑥  
next action = R



untried:

state s	untried actions
S <sub>13</sub>	∅
S <sub>23</sub>	∅
S <sub>12</sub>	R, D

unbacktracked:

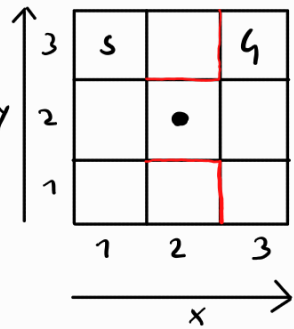
state s	states never backtracked to
S <sub>23</sub>	S <sub>13</sub>
S <sub>13</sub>	S <sub>23</sub>
S <sub>12</sub>	S <sub>13</sub>

result:

state s	action a <sub>1</sub>	resulting state s'
S <sub>13</sub>	R	S <sub>23</sub>
S <sub>23</sub>	L	S <sub>13</sub>
S <sub>13</sub>	D	S <sub>12</sub>
S <sub>12</sub>	U	S <sub>13</sub>

7

next action = R



untried:

state s	untried actions
S <sub>13</sub>	∅
S <sub>23</sub>	∅
S <sub>12</sub>	D
S <sub>22</sub>	R, L

unbacktracked:

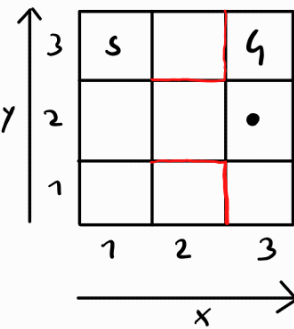
state s	states never backtracked to
S <sub>23</sub>	S <sub>13</sub>
S <sub>13</sub>	S <sub>23</sub>
S <sub>12</sub>	S <sub>13</sub>
S <sub>22</sub>	S <sub>12</sub>

result:

state s	action a <sub>1</sub>	resulting state s'
S <sub>13</sub>	R	S <sub>23</sub>
S <sub>23</sub>	L	S <sub>13</sub>
S <sub>13</sub>	D	S <sub>12</sub>
S <sub>12</sub>	U	S <sub>13</sub>
S <sub>12</sub>	R	S <sub>22</sub>

8

next action = U



untried:

state s	untried actions
S <sub>13</sub>	∅
S <sub>23</sub>	∅
S <sub>12</sub>	R, D
S <sub>22</sub>	R, L
S <sub>32</sub>	U, L, D

unbacktracked:

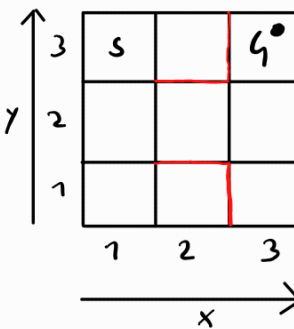
state s	states never backtracked to
S <sub>23</sub>	S <sub>13</sub>
S <sub>13</sub>	S <sub>23</sub>
S <sub>12</sub>	S <sub>13</sub>
S <sub>22</sub>	S <sub>12</sub>
S <sub>32</sub>	S <sub>22</sub>

result:

state s	action a <sub>1</sub>	resulting state s'
S <sub>13</sub>	R	S <sub>23</sub>
S <sub>23</sub>	L	S <sub>13</sub>
S <sub>13</sub>	D	S <sub>12</sub>
S <sub>12</sub>	U	S <sub>13</sub>
S <sub>12</sub>	R	S <sub>22</sub>
S <sub>22</sub>	R	S <sub>32</sub>

9

next action = ∅ Goal node found!



untried:

state s	untried actions
S <sub>13</sub>	∅
S <sub>23</sub>	∅
S <sub>12</sub>	R, D
S <sub>22</sub>	R, L
S <sub>32</sub>	L, D

unbacktracked:

state s	states never backtracked to
S <sub>23</sub>	S <sub>13</sub>
S <sub>13</sub>	S <sub>23</sub>
S <sub>12</sub>	S <sub>13</sub>
S <sub>22</sub>	S <sub>12</sub>
S <sub>32</sub>	S <sub>22</sub>

result:

state s	action a <sub>1</sub>	resulting state s'
S <sub>13</sub>	R	S <sub>23</sub>
S <sub>23</sub>	L	S <sub>13</sub>
S <sub>13</sub>	D	S <sub>12</sub>
S <sub>12</sub>	U	S <sub>13</sub>
S <sub>12</sub>	R	S <sub>22</sub>
S <sub>22</sub>	R	S <sub>32</sub>