Beyond Classical Search

Chapter 4, Section 5

Outline

◊ **On-line search in unknown environments**

◊ On-line search problem formulation

◊ Learning Real-Time A*

On-line search and unknown environments

Off-line search commits to a solution and executes it

On-line search algorithms interleave computation and action:
search $\rightarrow$ execute $\rightarrow$ observe $\rightarrow$ search $\rightarrow$ . . .

When is it useful???

◊ good idea in dynamic environments

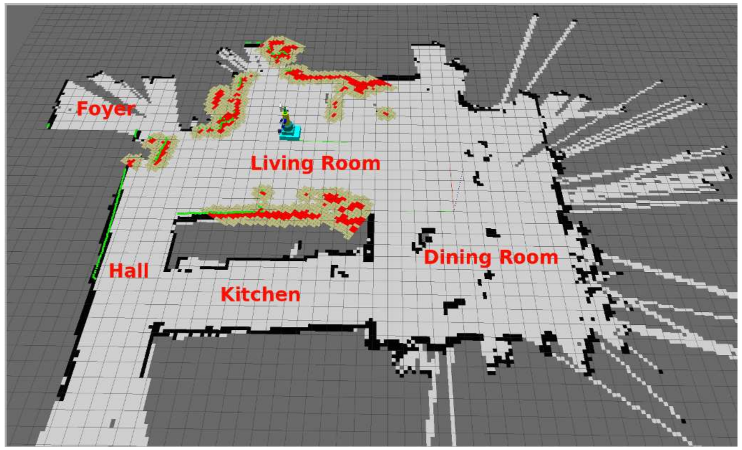
◊ good idea in time-constrained environments

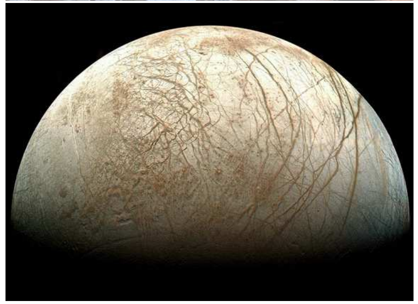
◊ necessary in unknown environments [exploring & learning]

Think of baby gradually discovering how the world works

Applications

- building a map of an unknown building
- search and rescue applications
- submarine robot exploration of Europa





On-line search problem formulation

An on-line search problem consists of:

◇ ACTIONS(s): returns the set of legal actions in state s [effects not known]

◇ The step cost function $c(s, a, s')$: returns the cost of going from s to s' via action a. This cannot be used until the agent has tried a in s and knows that s' is the outcome. [learnt]

◇ GOAL-TEST(s): returns true if s is a goal state

Observability: the current percept identifies the current state [Assume percept is good enough]

The agent cannot determine RESULT(s, a) **except by being in s, executing a and observing the result!** Degree of ignorance might be reduced for some applications then memorize

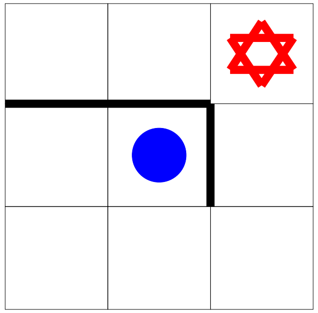
The agent might have access to some (admissible) heuristic function $h(s)$ [assume it has]

estimate

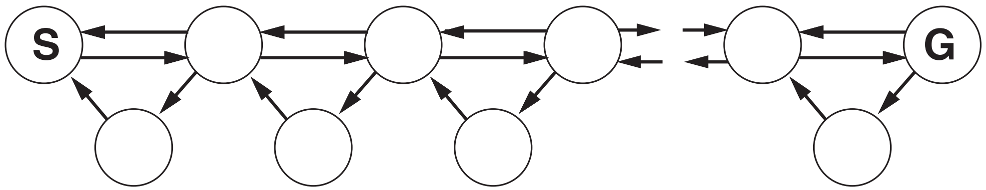
[Assume] No dead-end: the goal must be reachable from every state
no algorithm can avoid dead-ends in all environments

Local search algorithms

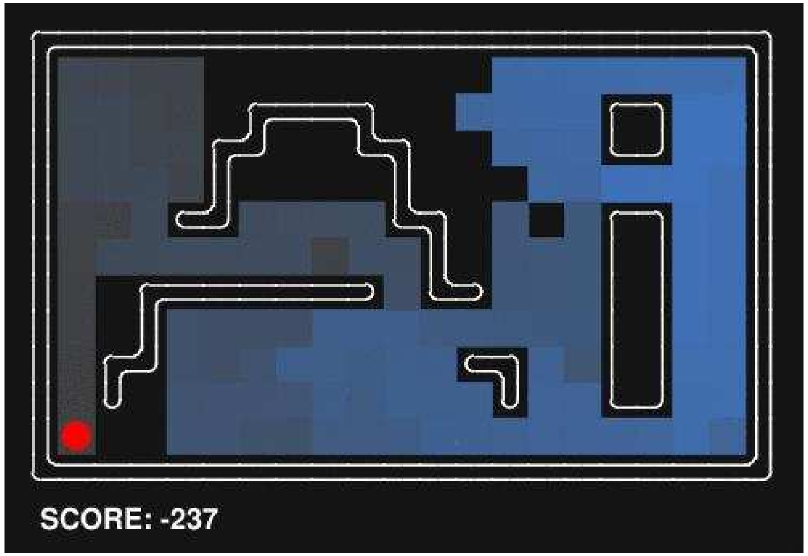
Hill-climbing using $h(s)$ can quickly gets stuck in a local minimum.

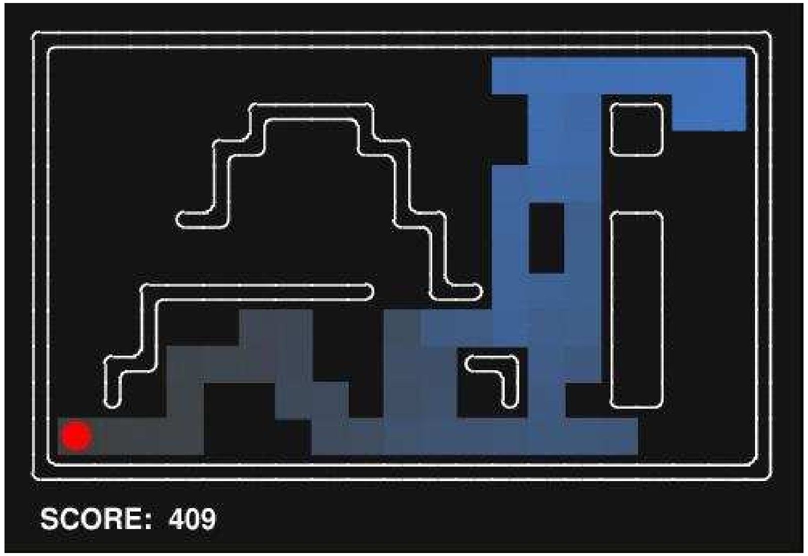


Random walk will eventually find a goal but will often take exponentially many steps.



Random vs LRTA*





LRTA*: Learning Real-Time A*

Learning action results

Store a table $result(s, a)$ recording the result of actions tried

When a is first tried in s, record the resulting state

Learning estimated cost to the goal ⇄ heuristic

Store and update a table $H(s)$ recording the estimate of visited states

When s is first met, initialise $H(s) = h(s)$ for some admissible $h(s)$

When s is left, update its cost estimate $H(s)$ to $\min_a C(s, a)$

estimate when met. replace this est. when we leave it

A* [just like A* which the initial state of path is not the start state]

$C(s, a)$ is the estimated cost of reaching the goal from state s via action a:

$$C(s,a) = \begin{cases} c(s,a,s') + H(s') & \text{if } result(s,a) \text{ is known to be } s' \\ h(s) & \text{otherwise} \end{cases}$$

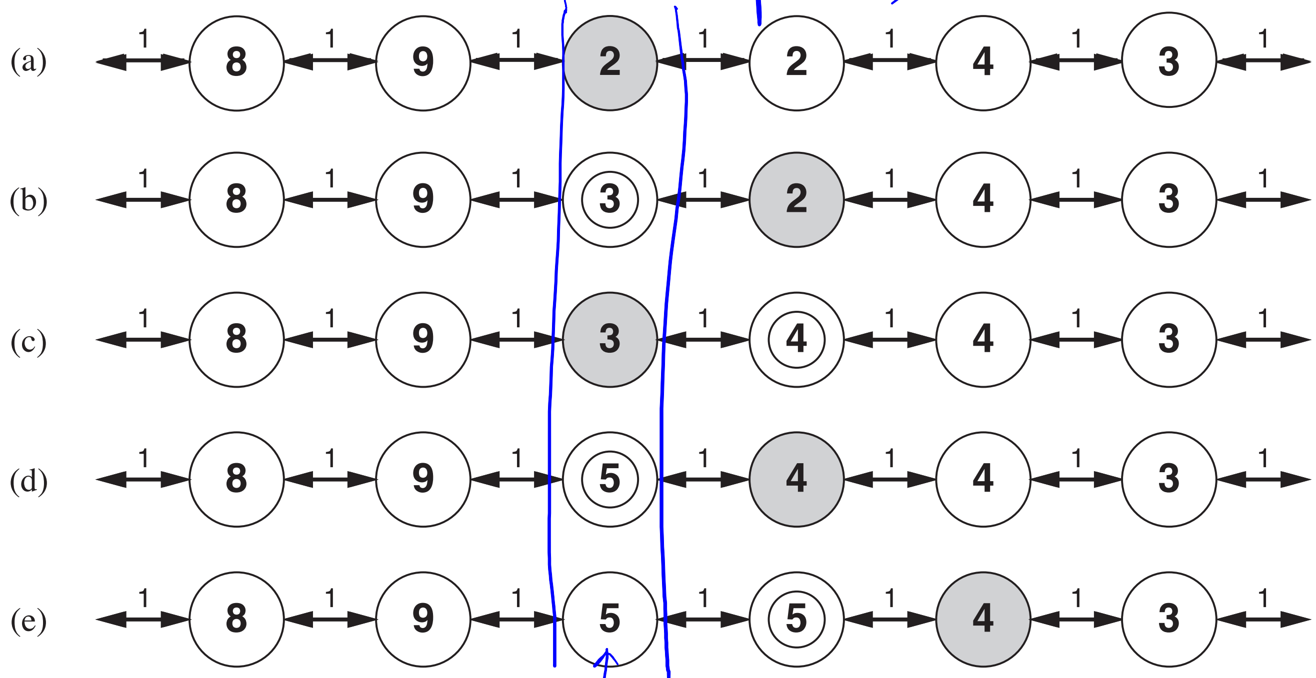
Take the best local move (real-time)

In state s select the action a minimising the estimated cost $C(s, a)$

Update of cost estimates

Cost estimates improve and converge to the true cost (over a number of trials) This allows escaping local maxima.

updating



big h value

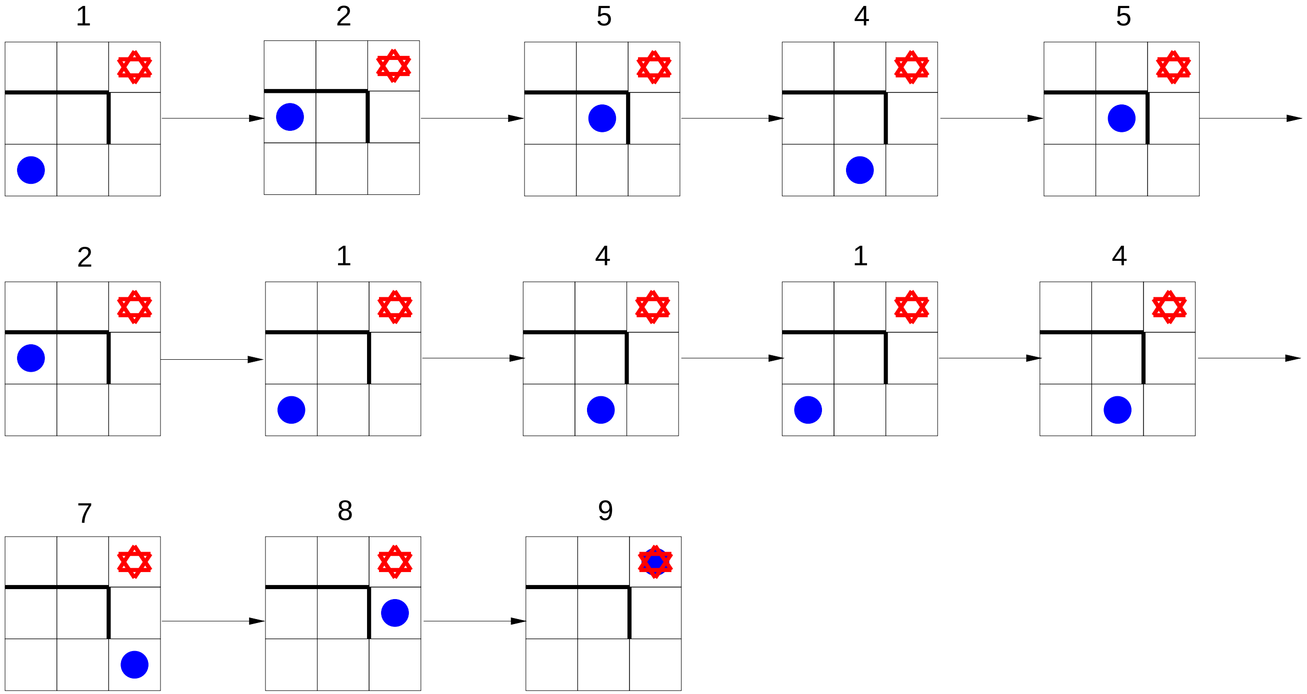
LRTA* Algorithm

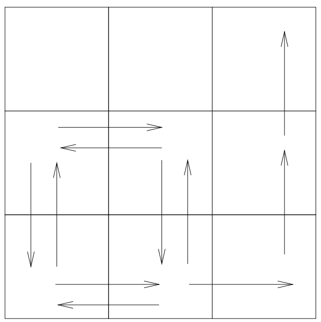
function LRTA*-AGENT(*cur*) **returns** an action
  inputs: *cur*, a percept that identifies the current state
  static: *prev*, *action*, the previous state and action, initially null
    result, table indexed by state and action, initially empty
    H, table of cost estimates indexed by state, initially empty

  if GOAL-TEST(*cur*) **then return** *stop*
  if *cur* is a new state **then** $H[cur] \leftarrow h(cur)$ / heuristic
  if *prev* is not null **then**
    $result[prev, action] \leftarrow cur$
    $H[prev] \leftarrow \min_{a \in \text{ACTIONS}(prev)} \text{EST-COST}(prev, a, result[prev, a], H)$ ≪ update
  $action \leftarrow \text{argmin}_{a \in \text{ACTIONS}(cur)} \text{EST-COST}(cur, a, result[cur, a], H)$
  $prev \leftarrow cur$
  return *action*

function EST-COST(*s*, *a*, *s'*, H) **returns** a cost estimate
  if *s'* is undefined **then return** $h(s)$
  else return $c(s,a,s') + H[s']$ Δ

Example





result function

		0
5	4	1
4	3	2

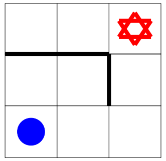
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example

1



"ah I have actions A & B, take one at random"

result function

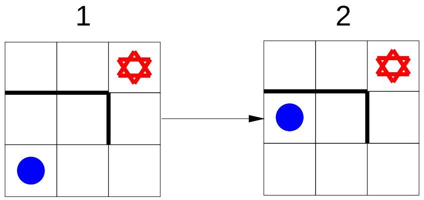
4		

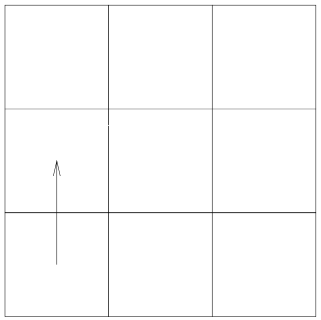
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

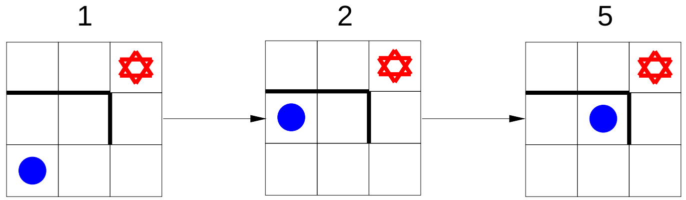
3		
4		

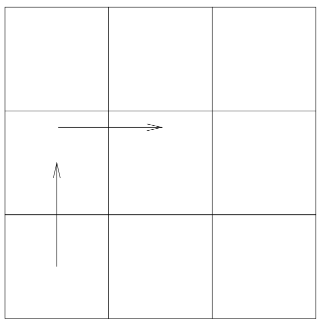
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

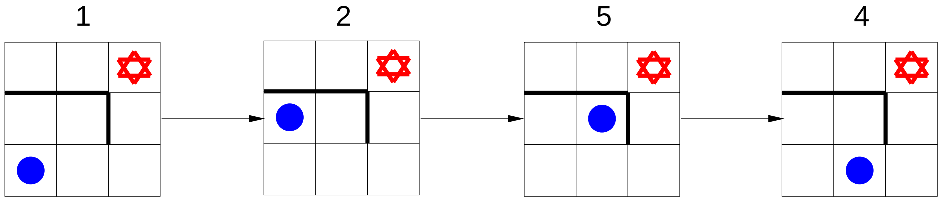
3	2	
4		

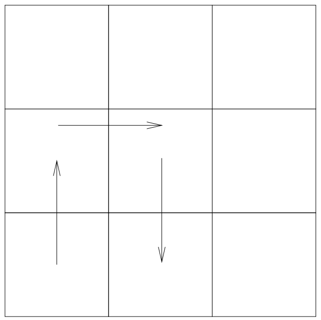
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

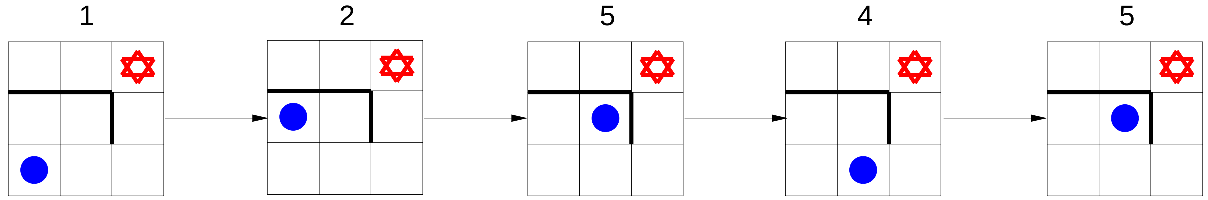
3	2	
4	3	

cost estimation H

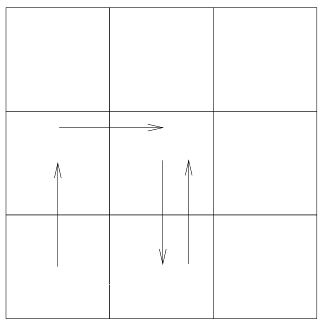
2	1	0
3	2	1
4	3	2

heuristic h

Example



"now I have estimations"



result function

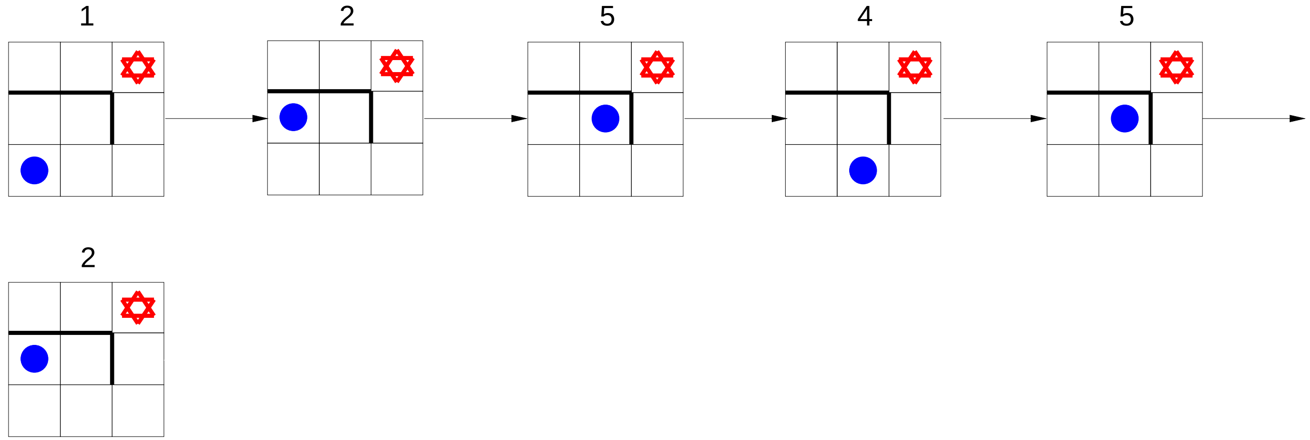
3	2	
4	3	

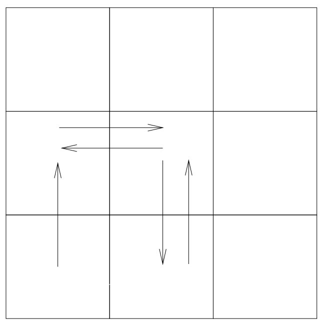
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

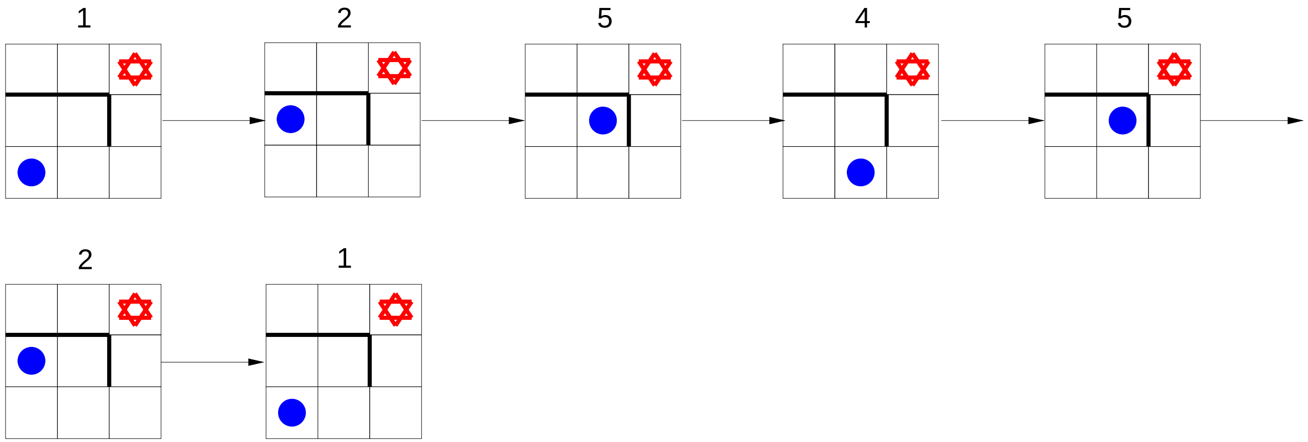
3	4	
4	3	

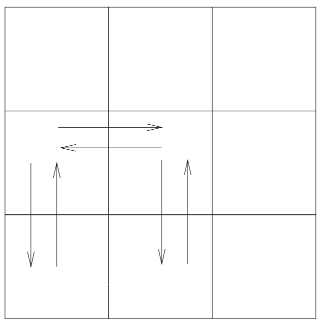
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

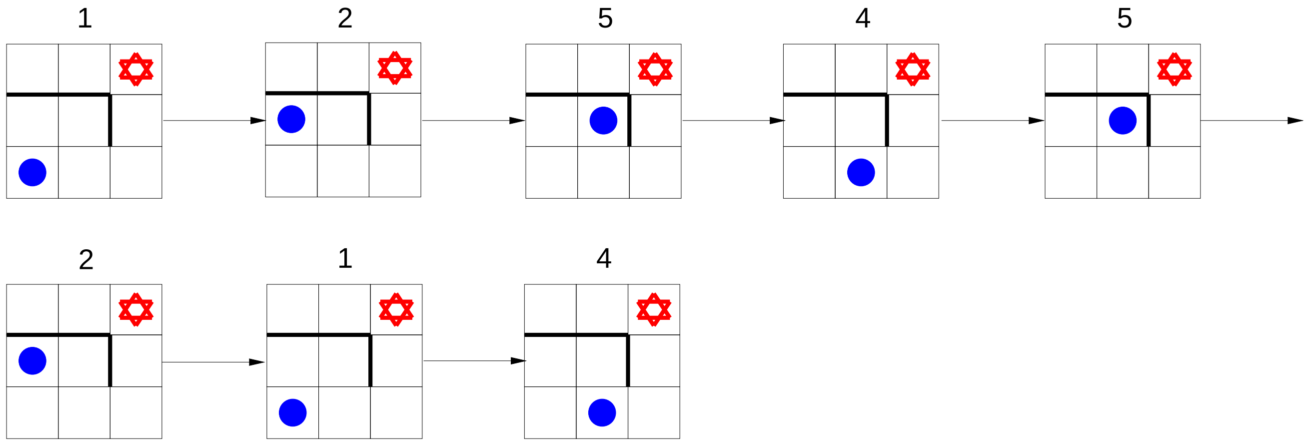
5	4	
4	3	

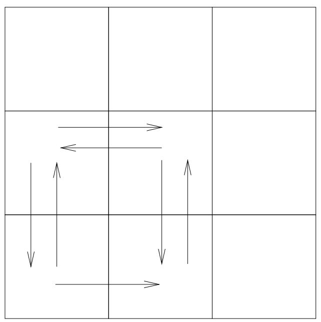
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example




result function

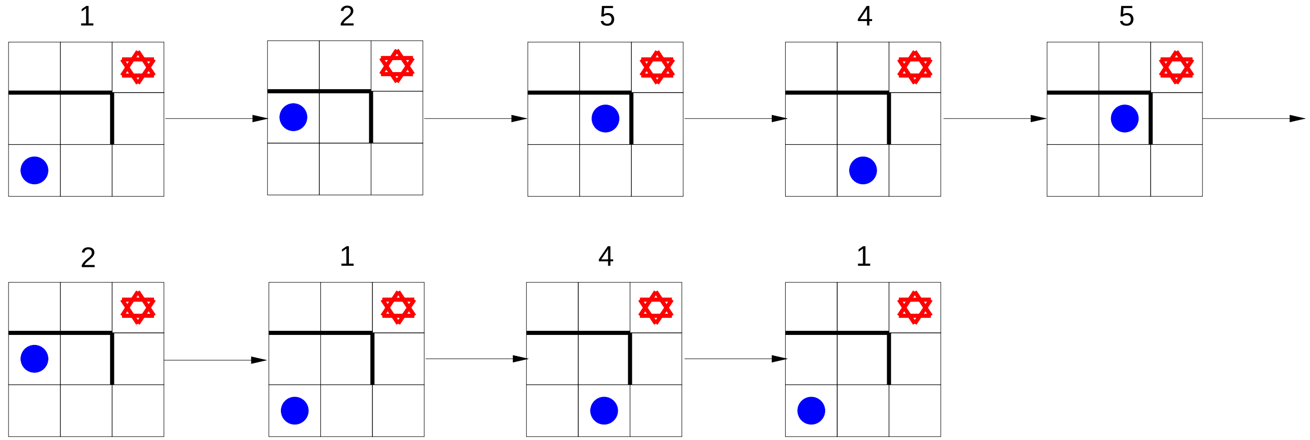
5	4	
4	3	

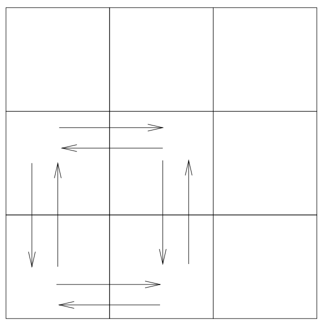
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

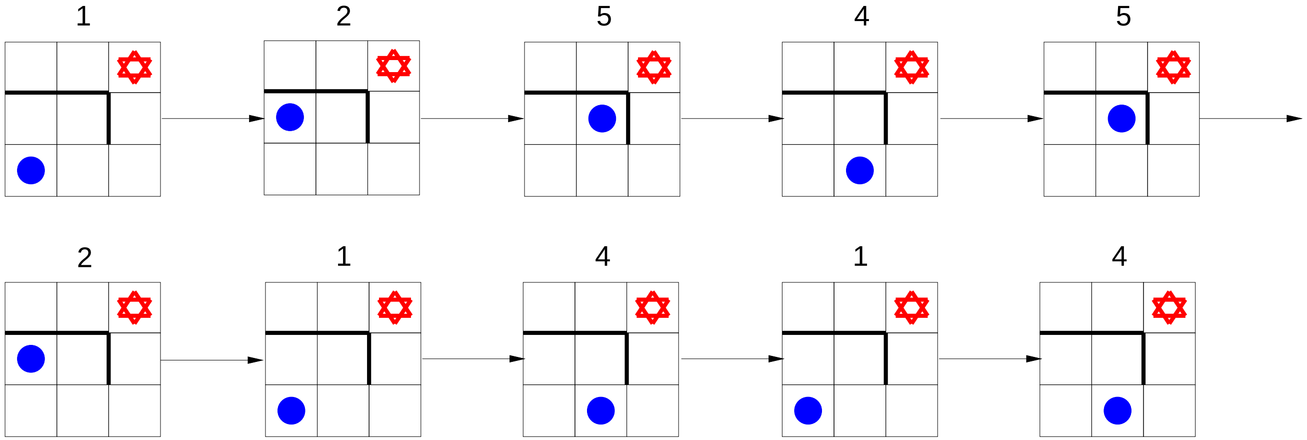
5	4	
4	3	

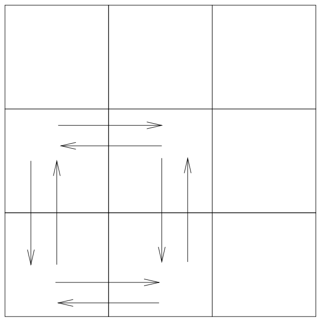
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

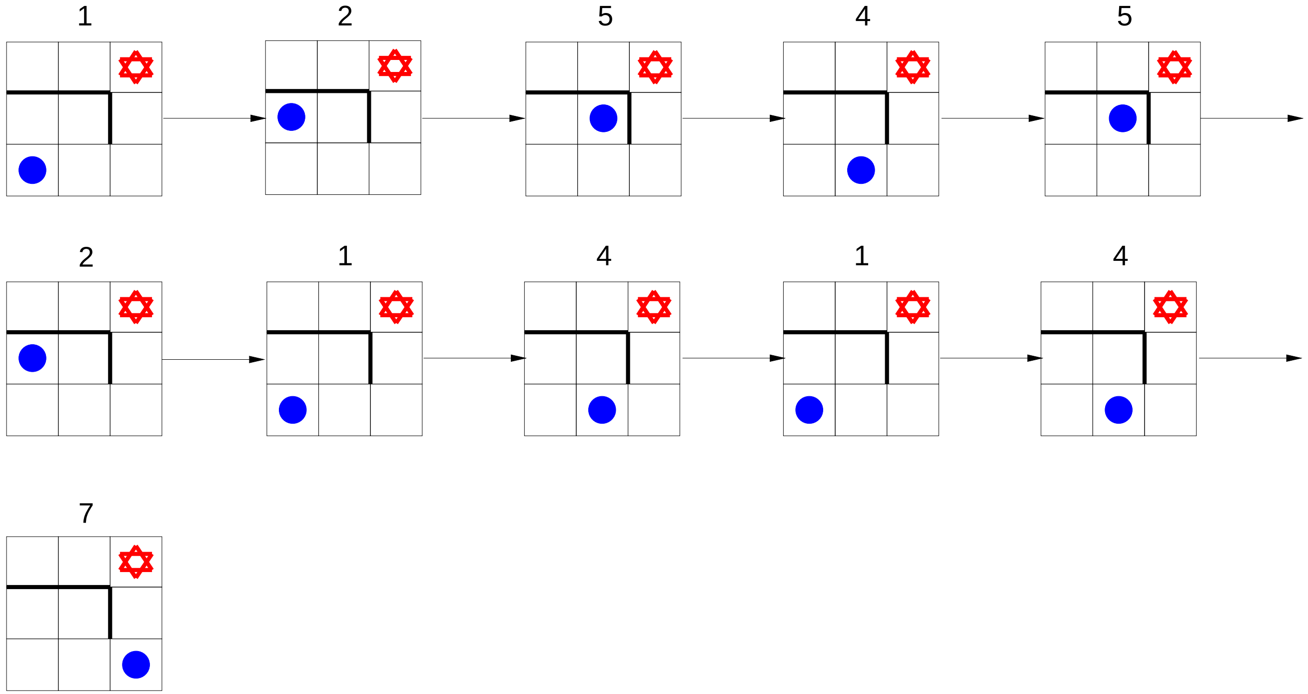
5	4	
4	3	

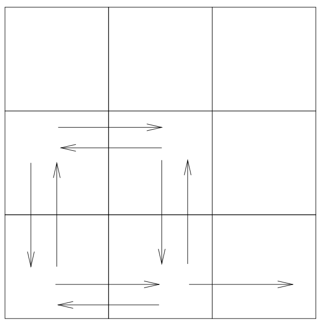
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

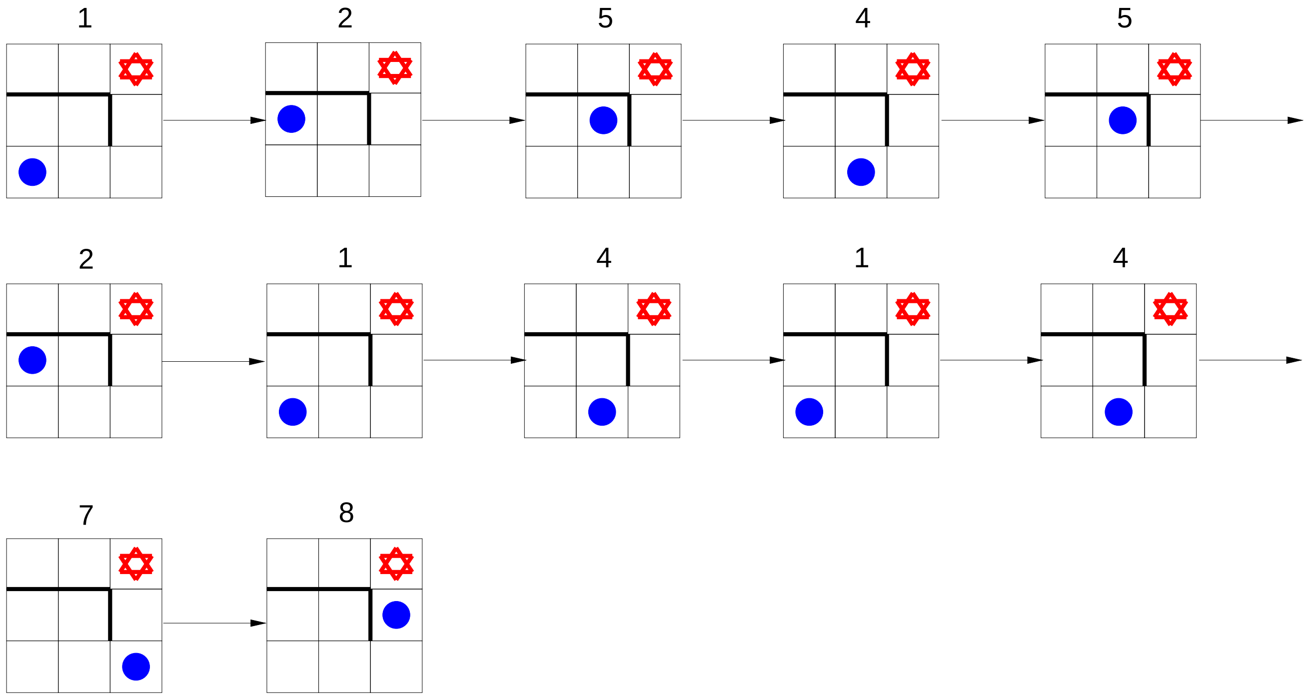
5	4	
4	3	2

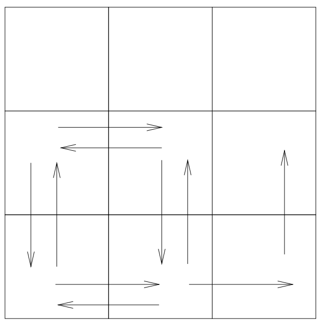
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

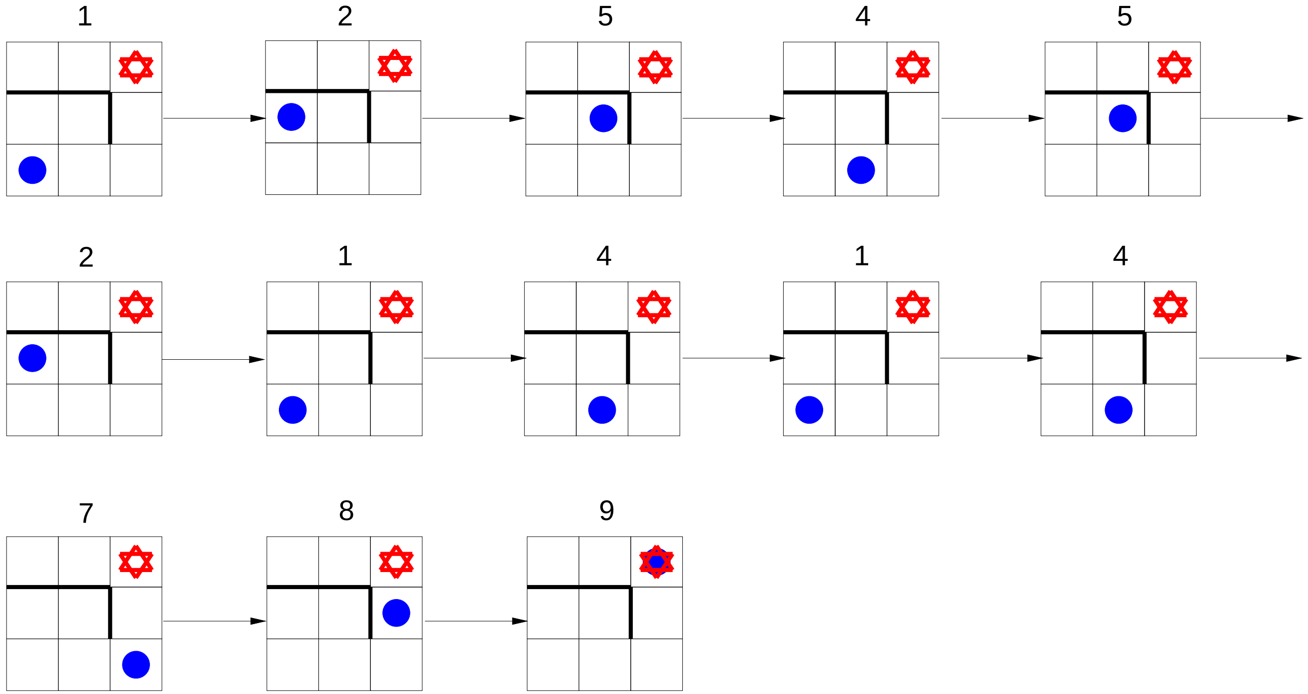
5	4	1
4	3	2

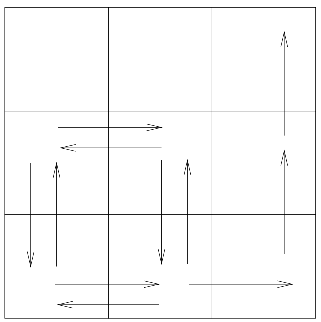
cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Example





result function

		0
5	4	1
4	3	2

cost estimation H

2	1	0
3	2	1
4	3	2

heuristic h

Properties of LRTA*

Complete?? Yes if the state space is finite (and there are no dead-ends)

Time?? $O(|S|^3)$

Space?? $O(|S| \times |A|)$ to store Result

Optimal?? no, but if h is admissible, converges to the optimal over repeated trials

"actually pretty good"

Summary

Exploration problems arise when the agent does not know the states or physics of its environment.

On-line search can build a map and find a goal state if there are no dead-ends.

Hill-climbing can get stuck in local minima.

LRTA* updates heuristic information to escape local minima and find the goal in much fewer steps than random walks.