Beyond Classical Search

Dr. Steven Bethard

Computer and Information Sciences
University of Alabama at Birmingham

26 Jan 2016

Outline

1. Local Search
   - Hill Climbing
   - Random Hill Climbing
   - Local Beam Search

2. Advanced Search
   - Continuous Search Spaces
   - Partial Information
   - Online Search

Outline

1. Local Search
   - Hill Climbing
   - Random Hill Climbing
   - Local Beam Search
2. Advanced Search
   - Continuous Search Spaces
   - Partial Information
   - Online Search

Local Search

Sometimes only the goal matters (not the path)

- 8-Queens
- Bag Generation
- Job Scheduling

Iterative Improvement

Single *current* state, explores neighbors

Memory?	$O(1)$
Optimal?	*Usually not*
Complete?	*Usually not*

Local Search

Sometimes only the goal matters (not the path)

- 8-Queens
- Bag Generation
- Job Scheduling

Iterative Improvement

Single *current* state, explores neighbors

Memory?	$O(1)$
Optimal?	*Usually not*
Complete?	*Usually not*

Local Search

Sometimes only the goal matters (not the path)

- 8-Queens
- Bag Generation
- Job Scheduling

Iterative Improvement

Single *current* state, explores neighbors

Memory?	$O(1)$
Optimal?	*Usually not*
Complete?	*Usually not*

Local Search

Sometimes only the goal matters (not the path)

- 8-Queens
- Bag Generation
- Job Scheduling

Iterative Improvement

Single *current* state, explores neighbors

Memory?	$O(1)$
Optimal?	*Usually not*
Complete?	*Usually not*

Local Search

Sometimes only the goal matters (not the path)

- 8-Queens
- Bag Generation
- Job Scheduling

Iterative Improvement

Single *current* state, explores neighbors

Memory?	$O(1)$
Optimal?	*Usually not*
Complete?	*Usually not*

Local Search

Sometimes only the goal matters (not the path)

- 8-Queens
- Bag Generation
- Job Scheduling

Iterative Improvement

Single *current* state, explores neighbors

Memory?	$O(1)$
Optimal?	*Usually not*
Complete?	*Usually not*

Local Search

Sometimes only the goal matters (not the path)

- 8-Queens
- Bag Generation
- Job Scheduling

Iterative Improvement

Single *current* state, explores neighbors

Memory?	$O(1)$
Optimal?	*Usually not*
Complete?	*Usually not*

Local Search

Sometimes only the goal matters (not the path)

- 8-Queens
- Bag Generation
- Job Scheduling

Iterative Improvement

Single *current* state, explores neighbors

Memory?	$O(1)$
Optimal?	*Usually not*
Complete?	*Usually not*

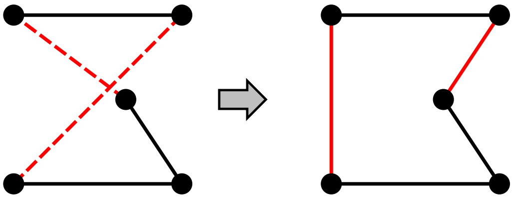
Example: Traveling Salesperson Problem

Problem

Visit all cities exactly once, minimum distance

Start A random complete tour

Move Swap a pair to reduce total distance



Achieves 1% of optimal with thousands of cities

Example: n-queens

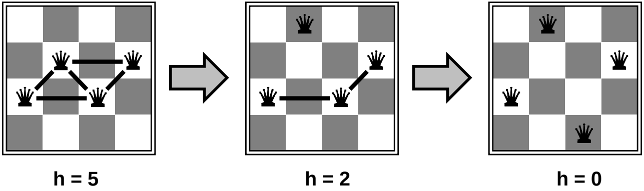
Problem

Put n queens on an $n \times n$ board

No two queens on the same row, column, or diagonal

Start All queens placed randomly

Move Move a queen to reduce conflicts

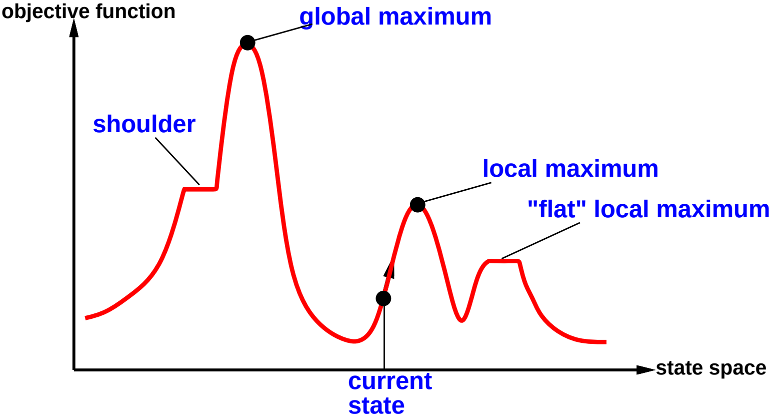


Can solve n-queens problems for, e.g. $n = 1$ million

Hill Climbing

```
def hill_climb(problem):

    # start at the problem's initial state
    current = Node(problem.get_random_complete_state())
    while True:

        # select the neighboring state with the best score
        state_scores = []
        for state, score in problem.get_neighbors(current.state):
            state_scores.append((score, state))
        best_score, best_state = max(state_scores)

        # if no neighbors are better, return the current
        if best_score <= current.score:
            return current.state

        # otherwise, move current to the best state
        current = Node(best_state, best_score)
```

Hill Climbing



Un-Sticking Hill Climbing

Sideways Moves

Allow moving to neighbors as good as the current

Stochastic Hill Climbing

Choose randomly from *all* neighbors that improve score

Random Restart Hill Climbing

Generate a new initial state and try again Complete!

Simulated Annealing

Some bad moves; gradually decrease size and frequency

Complete and Optimal if "gradual" enough

Un-Sticking Hill Climbing

Sideways Moves

Allow moving to neighbors as good as the current

Stochastic Hill Climbing

Choose randomly from *all* neighbors that improve score

Random Restart Hill Climbing

Generate a new initial state and try again

Simulated Annealing

Some bad moves; gradually decrease size and frequency

Complete and Optimal if gradual enough

Un-Sticking Hill Climbing

Sideways Moves

Allow moving to neighbors as good as the current

Stochastic Hill Climbing

Choose randomly from *all* neighbors that improve score

Random Restart Hill Climbing

Generate a new initial state and try again Complete!

Simulated Annealing

Some bad moves; gradually decrease size and frequency

Complete and Optimal if "gradual" enough

Un-Sticking Hill Climbing

Sideways Moves

Allow moving to neighbors as good as the current

Stochastic Hill Climbing

Choose randomly from *all* neighbors that improve score

Random Restart Hill Climbing

Generate a new initial state and try again Complete!

Simulated Annealing

Some bad moves; gradually decrease size and frequency

Complete and Optimal if "gradual" enough

Un-Sticking Hill Climbing

Sideways Moves

Allow moving to neighbors as good as the current

Stochastic Hill Climbing

Choose randomly from *all* neighbors that improve score

Random Restart Hill Climbing

Generate a new initial state and try again Complete!

Simulated Annealing

Some bad moves; gradually decrease size and frequency

Complete and Optimal if "gradual" enough

Un-Sticking Hill Climbing

Sideways Moves

Allow moving to neighbors as good as the current

Stochastic Hill Climbing

Choose randomly from *all* neighbors that improve score

Random Restart Hill Climbing

Generate a new initial state and try again Complete!

Simulated Annealing

Some bad moves; gradually decrease size and frequency

Complete and Optimal if “gradual” enough

Simulated Annealing

```python
def simulated_annealing(problem, get_temperature):
    current = Node(problem.get_random_complete_state())
    for time in itertools.count():

        # stop when the temperature reaches zero
        temperature = get_temperature(time)
        if temperature == 0:
            return current

        # select a random neighbor
        neighbors = problem.get_neighbors(current.state)
        state, score = random.choice(neighbors)

        # always move to the neighbor if it's better,
        # and sometimes if it's worse
        change = score - current.score
        prob = math.exp(change / temperature)
        if change > 0 or random.random() < prob:
            current = Node(state, score)
```

Local Beam Search

Idea

Keep k states instead of just one

vs. k Random Restarts

One state has good neighbors, others have bad neighbors

Local Beam Search: All searches share good neighbors

k Random Restarts: Other searches use bad neighbors

Variants

- Keep k best states
- Keep k random states, probabilities based on scores

Local Beam Search

Idea

Keep k states instead of just one

vs. k Random Restarts

One state has good neigbors, others have bad neighbors

Local Beam Search All searches share good neighbors

k Random Restarts Other searches use bad neighbors

Variants

- Keep k best states
- Keep k random states, probabilities based on scores

Local Beam Search

Idea

Keep k states instead of just one

vs. k Random Restarts

One state has good neigbors, others have bad neighbors

Local Beam Search All searches share good neighbors

k Random Restarts Other searches use bad neighbors

Variants

- Keep k best states
- Keep k random states: probabilities based on scores

Local Beam Search

Idea

Keep k states instead of just one

vs. k Random Restarts

One state has good neigbors, others have bad neighbors

Local Beam Search All searches share good neighbors

k Random Restarts Other searches use bad neighbors

Variants

- Keep k best states
- Keep k random states, probabilities based on scores

Local Beam Search

Idea

Keep k states instead of just one

vs. k Random Restarts

One state has good neigbors, others have bad neighbors

Local Beam Search All searches share good neighbors

k Random Restarts Other searches use bad neighbors

Variants

- Keep k best states
- Keep k random states, probabilities based on scores

Local Beam Search

Idea

Keep k states instead of just one

vs. k Random Restarts

One state has good neigbors, others have bad neighbors

Local Beam Search All searches share good neighbors

k Random Restarts Other searches use bad neighbors

Variants

- Keep k best states
- Keep k random states, probabilities based on scores

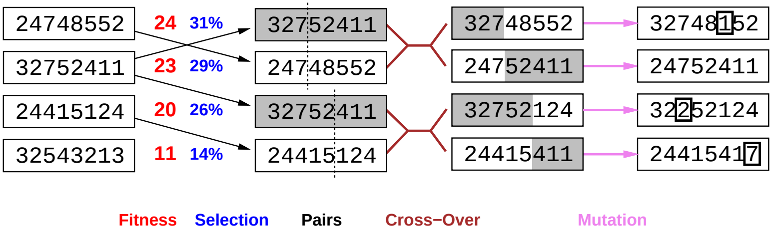
Local Beam Search

```
def local_beam_search(problem, k):
    current = [Node(problem.get_random_complete_state())]
    while True:

        # get all neighbors of the current states
        state_scores = []
        for node in current:
            for state, score in problem.get_neighbors(node.state):

                # return the first goal state generated
                if problem.is_goal(state):
                    return state
                state_scores.append((score, state))

        # select the k best states to consider next time
        current = []
        for score, state in heapq.nlargest(k, state_scores):
            current.append(Node(state, score))
```

Genetic Algorithms

Idea

- Stochastic local beam search
- Successors generated from **pairs** of states

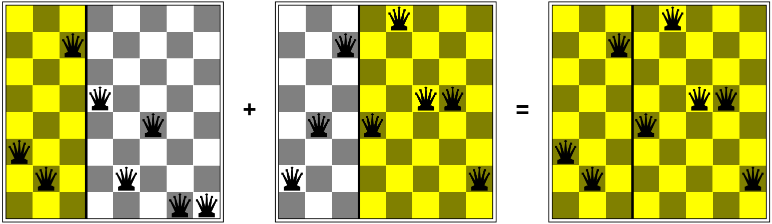


Genetic Algorithms

Requirements

- States must be encoded as strings
- Substrings must be meaningful components

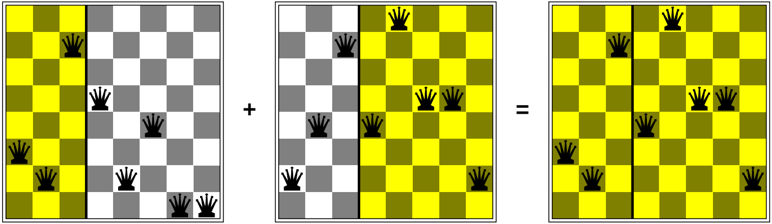
  or crossover is pointless!



Genetic Algorithms

Requirements

- States must be encoded as strings
- Substrings must be meaningful components
  or crossover is pointless!



Outline

Continuous Search Spaces

Problem

Place 3 airports in Romania, minimizing

$$\sum_{a \in airports} \sum_{c \in cities} (x_c - x_a)^2 + (y_c - y_a)^2$$

As a Search Problem?

But there are ∞ actions from each state!

Solutions

Discretize each action moves $\pm\delta$ in x or y direction

Gradient each action moves $\alpha \nabla f(x)$

Continuous Search Spaces

Problem

Place 3 airports in Romania, minimizing

$$\sum_{a \in airports} \sum_{c \in cities} (x_c - x_a)^2 + (y_c - y_a)^2$$

As a Search Problem?

But there are ∞ actions from each state!

Solutions

Discretize each action moves $\pm\delta$ in x or y direction

Gradient each action moves $\alpha \nabla f(x)$

Continuous Search Spaces

Problem

Place 3 airports in Romania, minimizing

$$\sum_{a \in airports} \sum_{c \in cities} (x_c - x_a)^2 + (y_c - y_a)^2$$

As a Search Problem?

But there are ∞ actions from each state!

Solutions

Discretize each action moves $\pm\delta$ in x or y direction

Gradient each action moves $\alpha \nabla f(x)$

Continuous Search Spaces

Problem

Place 3 airports in Romania, minimizing

$$\sum_{a \in airports} \sum_{c \in cities} (x_c - x_a)^2 + (y_c - y_a)^2$$

As a Search Problem?

But there are ∞ actions from each state!

Solutions

Discretize each action moves $\pm\delta$ in x or y direction

Gradient each action moves $\alpha \nabla f(x)$

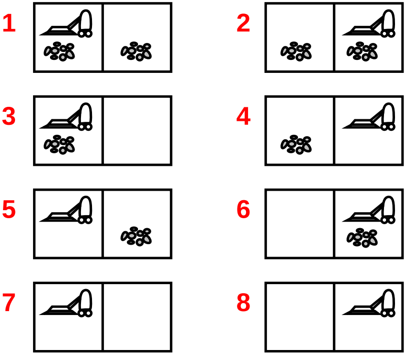
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
RIGHT	{2, 4, 6, 8}
SUCK	{4, 8}
LEFT	{3, 7}
SUCK	{7}



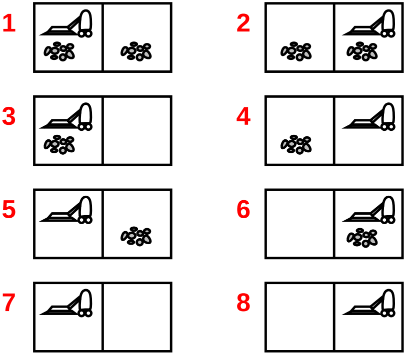
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
RIGHT	{2, 4, 6, 8}
SUCK	{4, 8}
LEFT	{3, 7}
SUCK	{7}



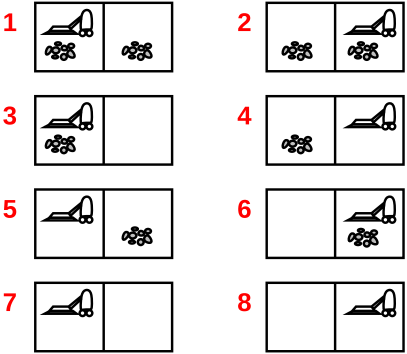
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
RIGHT	{2, 4, 6, 8}
SUCK	{4, 8}
LEFT	{3, 7}
SUCK	{7}



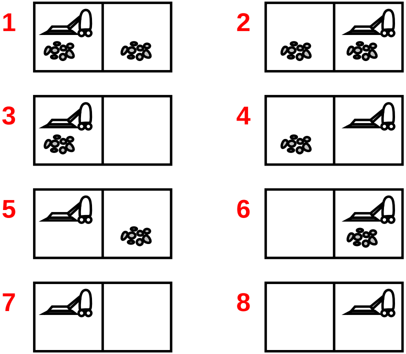
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
Right	{2, 4, 6, 8}
Suck	{4, 8}
Left	{3, 7}
Suck	{7}



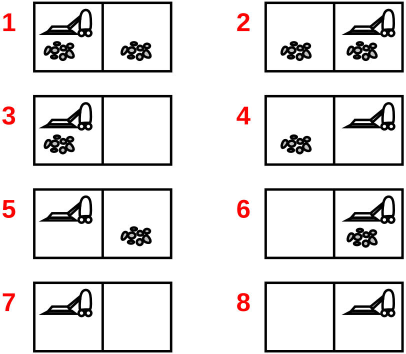
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
RIGHT	{2, 4, 6, 8}
SUCK	{4, 8}
LEFT	{3, 7}
SUCK	{7}



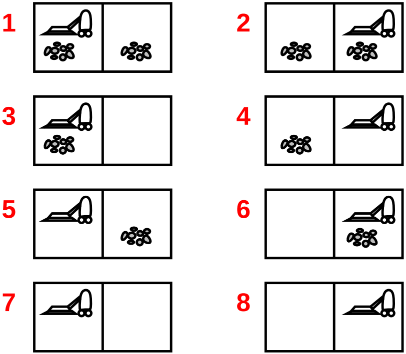
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
RIGHT	{2, 4, 6, 8}
SUCK	{4, 8}
LEFT	{3, 7}
SUCK	{7}



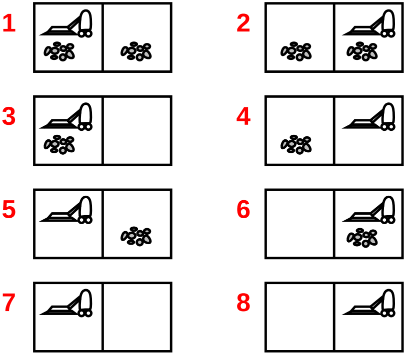
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
RIGHT	{2, 4, 6, 8}
SUCK	{4, 8}
LEFT	{3, 7}
SUCK	{7}



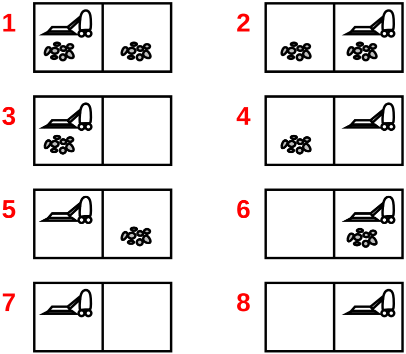
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
Right	{2, 4, 6, 8}
Suck	{4, 8}
Left	{3, 7}
Suck	{7}



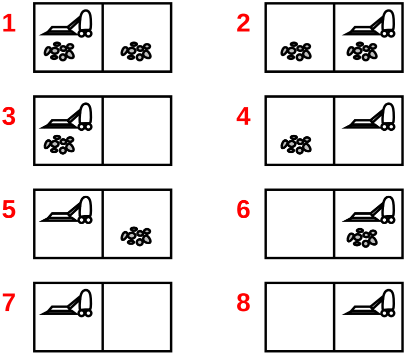
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
Right	{2, 4, 6, 8}
Suck	{4, 8}
Left	{3, 7}
Suck	{7}



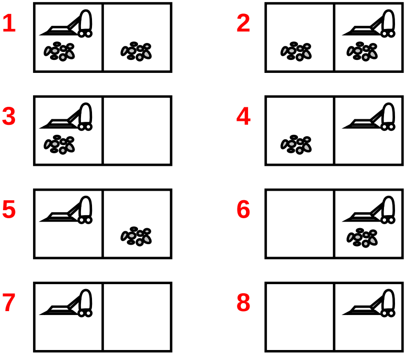
Searching with Partial Information

Problem

Start in any state

Solution?

Initial	{1, 2, 3, 4, 5, 6, 7, 8}
RIGHT	{2, 4, 6, 8}
SUCK	{4, 8}
LEFT	{3, 7}
SUCK	{7}



Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory

Complete in finite, safely explorable environments

Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory

$h(x) = 2$

Complete in finite, safely explorable environments

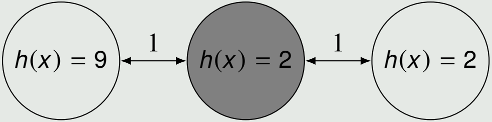
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

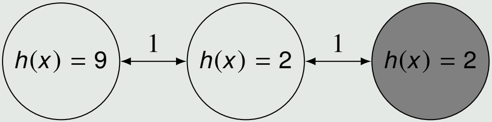
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

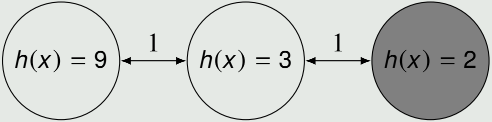
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

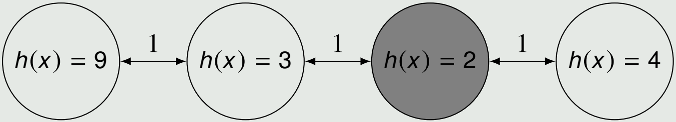
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

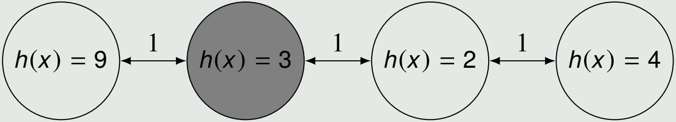
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

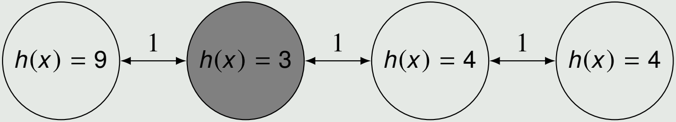
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

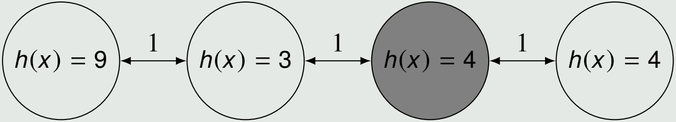
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

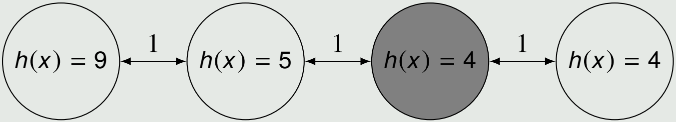
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

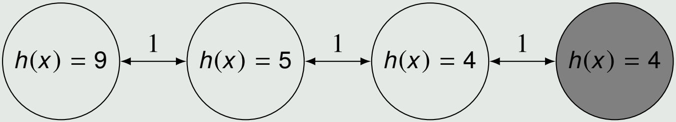
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

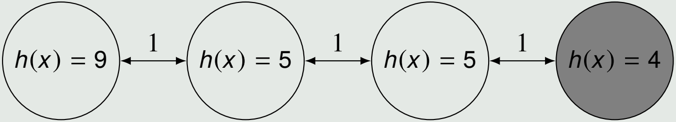
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

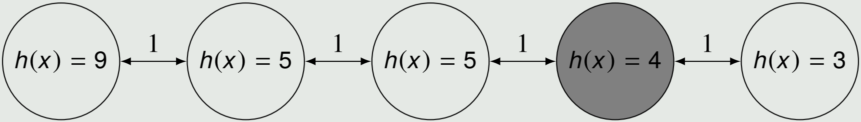
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

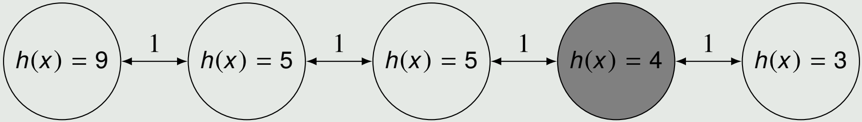
Online Search

Problem

Successor function is not available until a state is visited, e.g. robot exploration, maze problems

Solution: Learning Real-Time A*

Augment hill climbing with memory



Complete in finite, safely explorable environments

Key Points

Search Algorithms

- Hill Climbing
- Simulated Annealing
- Local Beam Search
- Discretized Search
- Multiple-Belief State Search
- LRTA* Search