

Artificial Intelligence- Informed Search and Exploration

Amir Aavani
Amir@Aavani.net
<http://Amir.Aavani.net/CS/AI-86>

5. November 2007

Outline

- Heuristic functions
- Local search alg. and Optimization problems
- Online search agents

Heuristic functions

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- Avg. solution depth is about 22.
- Branching factor is about 3(2,3 or 4).
- $3^{22} = 3 * 10^{10}$.
- The state space has about $\frac{9!}{2}$.

Heuristic functions, 8-puzzle, Cont

- For 15-puzzle, there are about 10^{13} different states.
- A good heuristic function can help to find solution.
- Two standard heuristic functions:

Heuristic functions, 8-puzzle, Cont

- For 15-puzzle, there are about 10^{13} different states.
- A good heuristic function can help to find solution.
- Two standard heuristic functions:
 - h_1 = number of misplaced tiles.

Heuristic functions, 8-puzzle, Cont

- For 15-puzzle, there are about 10^{13} different states.
- A good heuristic function can help to find solution.
- Two standard heuristic functions:
 - h_1 = number of misplaced tiles.
 - h_2 = Manhattan distance.

Heuristic quality and Dominance

- Effective branching factor (b^*):
$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$
- The less the b^* , the better the heuristic function.
- Let h_1 and h_2 are two admissible heuristic function.
- If $h_2(n) \geq h_1(n)$ for all n , then h_2 **dominates** h_1 .
- Using h_2 is more efficient h_1 .

Inventing admissible heuristic function

- Relaxed problem: problem with fewer restriction on actions.
- Cost of optimal solution of relaxed prob. is an admissible heuristic func.
- Any solution for main prob. is a solution of relaxed prob.
- Cost of optimal solution of relaxed prob is also **consistent**.

Inventing admissible heuristic function, Cont

- If problem can be stated in a formal Lang., it is possible to construct relaxed prob. automatically.
- For example:
 - A tile can move from A to B if A is horizontally or vertically adjacent to B and B is blank.
 - Can be relaxed to
 - A tile can move from A to B if A is adjacent to B.
 - A tile can move from A to B if B is blank.
 - A tile can move from A to B.

Inventing admissible heuristic function, Cont

*	2	4
*		*
*	3	1

Start State

	1	2
3	4	*
*	*	*

Goal State

- Solution cost of a subproblem of given problem can be admissible func.
- Pattern databases store the exact solution to for every possible subproblem instance.

Inventing admissible heuristic function, Cont

- There is a trade-off between better heuristic function and calculation.
- Consider having some admissible functions, h_1, h_2, \dots, h_k :
- How to build a good (admissible) function from them?

Inventing admissible heuristic function, Cont

- There is a trade-off between better heuristic function and calculation.
- Consider having some admissible functions, h_1, h_2, \dots, h_k :
- How to build a good (admissible) function from them?
- $h(n) = \max(h_1(n), h_2(n), \dots, h_k(n))$

Local Search and Optimization

- Previously: systematic exploration of search space for finding a path to goal.
- In some problems, the path to goal is irrelevant, e.g., n-queens, optimization problems.
- Search alg. can not be effective in these problems.
- Local search algorithms are used for this purpose.

Local Search and Optimization

- Previously: systematic exploration of search space for finding a path to goal.
- In some problems, the path to goal is irrelevant, e.g., n-queens, optimization problems.
- Search alg. can not be effective in these problems.
- Local search algorithms are used for this purpose.

Local Search and Optimization

- Previously: systematic exploration of search space for finding a path to goal.
- In some problems, the path to goal is irrelevant, e.g., n-queens, optimization problems.
- Search alg. can not be effective in these problems.
- Local search algorithms are used for this purpose.

Local Search and Optimization

- Previously: systematic exploration of search space for finding a path to goal.
- In some problems, the path to goal is irrelevant, e.g., n-queens, optimization problems.
- Search alg. can not be effective in these problems.
- Local search algorithms are used for this purpose.

Local Search Algorithms

- Local search algs uses single current state and move to neighbor states.
- Need to limited memory.
- Can find reasonable solutions (even in infinite space).
- Are usefull for solving the optimization problem.

Local Search Algorithms

- Local search algs uses single current state and move to neighbor states.
- Need to limited memory.
- Can find reasonable solutions (even in infinite space).
- Are usefull for solving the optimization problem.

Local Search Algorithms

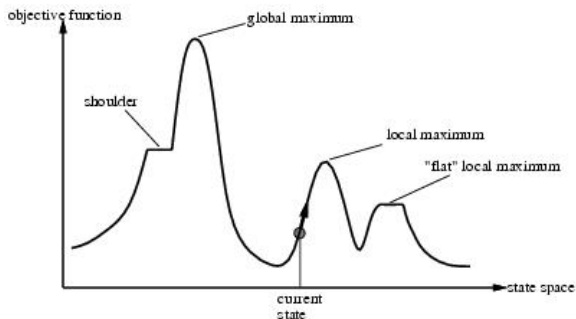
- Local search algs uses single current state and move to neighbor states.
- Need to limited memory.
- Can find reasonable solutions (even in infinite space).
- Are usefull for solving the optimization problem.

Local Search Algorithms

- Local search algs uses single current state and move to neighbor states.
- Need to limited memory.
- Can find reasonable solutions (even in infinite space).
- Are usefull for solving the optimization problem.

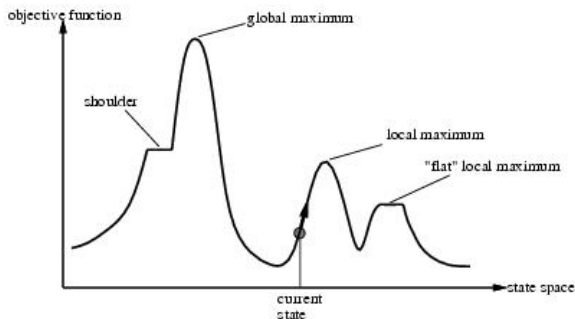
Optimization problems

- Optimization prob: finding the best state according to an **objective function**.
- In these problems, there is no “goal test” and no “path cost”.
- Global minimum/maximum vs local minimum/maximum



Optimization problems

- Optimization prob: finding the best state according to an **objective function**.
- In these problems, there is no “goal test” and no “path cost”.
- Global minimum/maximum vs local minimum/maximum



Local searches

- Hill climbing search
- Simulated annealing
- Local beam search
- Genetic Algorithms

Hill Climbing Search

- In each iteration, it moves in a direction of increasing value.
- Loops until a peak is reached.
- Usually uses complete-state formulation.
- It doesn't look-ahead of immediate neighbors of current state.

Hill Climbing Search

- In each iteration, it moves in a direction of increasing value.
- Loops until a peak is reached.
- Usually uses complete-state formulation.
- It doesn't look-ahead of immediate neighbors of current state.

Hill Climbing Search

- In each iteration, it moves in a direction of increasing value.
- Loops until a peak is reached.
- Usually uses complete-state formulation.
- It doesn't look-ahead of immediate neighbors of current state.

Hill Climbing Search

- In each iteration, it moves in a direction of increasing value.
- Loops until a peak is reached.
- Usually uses complete-state formulation.
- It doesn't look-ahead of immediate neighbors of current state.

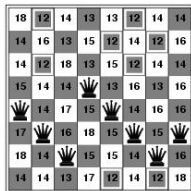
Hill Climbing Search, Cont

```
function HILL-CLIMBING (problem) returns a local maximum
  local variables: current, a node.
                  neighbor, a node.
```

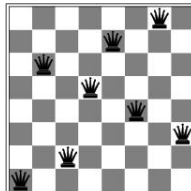
```
  current= MAKE-NODE(INITIAL-STATE[problem])
  loop
    neighbor= a highest valued successor of current
    if VALUE [neighbor] <= VALUE[current] then
      return STATE[current]
    current= neighbor
  do
```

Hill Climbing Search, Example

- 8-queens problem (complete-state formulation).
- Successor function: move a single queen to another square in same column.
- Evaluation function: Number of pairs of queens attacking each other.



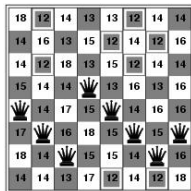
a.



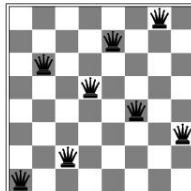
b.

Hill Climbing Search, Example

- 8-queens problem (complete-state formulation).
- Successor function: move a single queen to another square in same column.
- Evaluation function: Number of pairs of queens attacking each other.



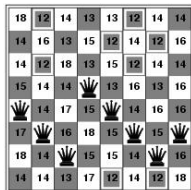
a.



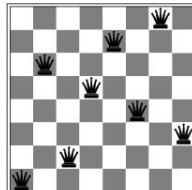
b.

Hill Climbing Search, Example

- 8-queens problem (complete-state formulation).
- Successor function: move a single queen to another square in same column.
- Evaluation function: Number of pairs of queens attacking each other.



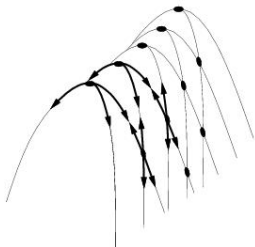
a.



b.

Hill Climbing Search, Drawbacks

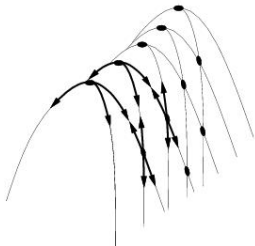
- Local maxima: Hill climbing will be got stuck.
- Ridges: Ridges result in a seq. of local maxima that is very difficult for greedy alg. to navigate.



- Plateaux: An area of state space where the eval. func. is flat. Shoulder or local maxima?

Hill Climbing Search, Drawbacks

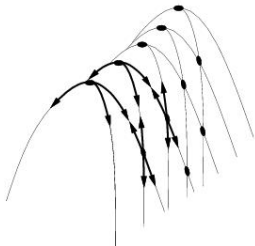
- Local maxima: Hill climbing will be got stuck.
- Ridges: Ridges result in a seq. of local maxima that is very difficult for greedy alg. to navigate.



- Plateaux: An area of state space where the eval. func. is flat. Shoulder or local maxima?

Hill Climbing Search, Drawbacks

- Local maxima: Hill climbing will be got stuck.
- Ridges: Ridges result in a seq. of local maxima that is very difficult for greedy alg. to navigate.



- Plateaux: An area of state space where the eval. func. is flat. Shoulder or local maxima?

Hill-climbing variations

- Stochastic hill-climbing: chooses at random from the uphill moves (by probability).
- Random-restart hill-climbing: Runs many hill-climbing search. It can overcome local maxima.
- Random sideways moves: Keep going in flat area in the hope that the area is a shoulder.

Hill-climbing variations

- Stochastic hill-climbing: chooses at random from the uphill moves (by probability).
- Random-restart hill-climbing: Runs many hill-climbing search. It can overcome local maxima.
- Random sideways moves: Keep going in flat area in the hope that the area is a shoulder.

Hill-climbing variations

- Stochastic hill-climbing: chooses at random from the uphill moves (by probability).
- Random-restart hill-climbing: Runs many hill-climbing search. It can overcome local maxima.
- Random sideways moves: Keep going in flat area in the hope that the area is a shoulder.

Some notes about hill-climbing

- Random-restart hill-climbing can solve three million queens in less than an hour.
- The success of hill-climbing is very dependent to shape of state-space.

Local searches

- Hill climbing search
- **Simulated annealing**
- Local beam search
- Genetic Algorithms

Simulated annealing search

- Hill-climbing alg. may get stuck on a local maximum.
- A purely random walk is complete, but extremely inefficient.
- Combining hill-climbing with a random walk can yields both efficiency and completeness.
- Simulated annealing is such an algorithm.

Simulated annealing search

- Hill-climbing alg. may get stuck on a local maximum.
- A purely random walk is complete, but extremely inefficient.
- Combining hill-climbing with a random walk can yields both efficiency and completeness.
- **Simulated annealing** is such an algorithm.

Simulated annealing search

- Hill-climbing alg. may get stuck on a local maximum.
- A purely random walk is complete, but extremely inefficient.
- Combining hill-climbing with a random walk can yields both efficiency and completeness.
- Simulated annealing is such an algorithm.

Simulated annealing search

- Hill-climbing alg. may get stuck on a local maximum.
- A purely random walk is complete, but extremely inefficient.
- Combining hill-climbing with a random walk can yields both efficiency and completeness.
- **Simulated annealing** is such an algorithm.

Simulated annealing search, cont

- Getting a ping-pong ball into the deepest crevice:
- Shaking the surface too hard results in random walk.
- Shaking the surface with very low intensity results in hill-climbing.
- Shaking the surface hard and then, gradually reduce intensity of shaking results in simulated annealing.

Simulated annealing search, cont

- Getting a ping-pong ball into the deepest crevice:
- Shaking the surface too hard results in random walk.
- Shaking the surface with very low intensity results in hill-climbing.
- Shaking the surface hard and then, gradually reduce intensity of shaking results in simulated annealing.

Simulated annealing search, cont

- Getting a ping-pong ball into the deepest crevice:
- Shaking the surface too hard results in random walk.
- Shaking the surface with very low intensity results in hill-climbing.
- Shaking the surface hard and then, gradually reduce intensity of shaking results in simulated annealing.

Simulated annealing search, cont

- Getting a ping-pong ball into the deepest crevice:
- Shaking the surface too hard results in random walk.
- Shaking the surface with very low intensity results in hill-climbing.
- Shaking the surface hard and then, gradually reduce intensity of shaking results in simulated annealing.

Simulated annealing search, cont

function SIMULATED-ANNEALING (problem, schedule) returns a
 schedule: a mapping from time to temperature

```
current= MAKE-NODE (INITIAL-STATE [problem]);
for t= 1 to Infity do
    T= schedule [t];
    if T= 0 then
        return current;
    next= a randomly selected successor of current;
    delta= VALUE [next]- VALUE [current];
    if 0< Delta then
        current= next;
    else
        current= next with probability of  $\text{Exp}(\text{delta}/T)$ ;
```


Local searches

- Hill climbing search
- Simulated annealing
- Local beam search
- Genetic Algorithms

Local beam search

- Holds k states instead of one, initially k randomly generated states.
- Find all successors of k states.
- If any of successors is goal \Rightarrow terminate.
- Select k best from successors and repeat.

Local beam search

- Holds k states instead of one, initially k randomly generated states.
- Find all successors of k states.
- If any of successors is goal \Rightarrow terminate.
- Select k best from successors and repeat.

Local beam search

- Holds k states instead of one, initially k randomly generated states.
- Find all successors of k states.
- If any of successors is goal \Rightarrow terminate.
- Select k best from successors and repeat.

Local beam search

- Holds k states instead of one, initially k randomly generated states.
- Find all successors of k states.
- If any of successors is goal \Rightarrow terminate.
- Select k best from successors and repeat.

Local beam search, Cont

- Difference with k hill-climbing:
 - Usefull information is passed among the k parallel search threads.
 - Consider the case in which one of states has better successors than all $k-1$ other states.
 - Can suffer from lack of diversity:
 - Stochastic variant: choose k successors, each with probability related to its value.

Local beam search, Cont

- Difference with k hill-climbing:
- Usefull information is passed among the k parallel search threads.
- Consider the case in which one of states has better successors than all $k-1$ other states.
- Can suffer from lack of diversity:
- Stochastic variant: choose k successors, each with probability related to its value.

Local beam search, Cont

- Difference with k hill-climbing:
- Usefull information is passed among the k parallel search threads.
- Consider the case in which one of states has better successors than all $k-1$ other states.
- Can suffer from lack of diversity:
- Stochastic variant: choose k successors, each with probability related to its value.

Local searches

- Hill climbing search
- Simulated annealing
- Local beam search
- Genetic Algorithms

Genetic Algorithms

- 1 It is a variant of local search.
- 2 Generates successors states:
 - Cross over: by pairs of states (two parents).
 - Mutation: By modifying a single state.
- 3 Generates successors from pairs (two parents) of states.
- 4 Tries to simulates a natural phenomena.

Genetic Algorithms

- 1 It is a variant of local search.
- 2 Generates successors states:
 - Cross over: by pairs of states (two parents).
 - Mutation: By modifying a single state.
- 3 Generates successors from pairs (two parents) of states.
- 4 Tries to simulates a natural phenomena.

Genetic Algorithms

- 1 It is a variant of local search.
- 2 Generates successors states:
 - Cross over: by pairs of states (two parents).
 - Mutation: By modifying a single state.
- 3 Generates successors from pairs (two parents) of states.
- 4 Tries to simulates a natural phenomena.

Genetic Algorithms

- 1 It is a variant of local search.
- 2 Generates successors states:
 - Cross over: by pairs of states (two parents).
 - Mutation: By modifying a single state.
- 3 Generates successors from pairs (two parents) of states.
- 4 Tries to simulates a natural phenomena.

Genetic Algorithms, Cont

- 1 Starts with a set(multi-set) of k randomly generated states.
- 2 Each state(individual) is represented as a chromosome (finite string).
- 3 Each chromosome is made up of some genome (characters).
- 4 Each individual is ranked by fitness function.
- 5 Fitness function should return higher value for better states.

Genetic Algorithms, Cont

- 1 Starts with a set(multi-set) of k randomly generated states.
- 2 Each state(individual) is represented as a chromosome (finite string).
- 3 Each chromosome is made up of some genome (characters).
- 4 Each individual is ranked by fitness function.
- 5 Fitness function should return higher value for better states.

Genetic Algorithms, Cont

- 1 Starts with a set(multi-set) of k randomly generated states.
- 2 Each state(individual) is represented as a chromosome (finite string).
- 3 Each chromosome is made up of some genome (characters).
- 4 Each individual is ranked by fitness function.
- 5 Fitness function should return higher value for better states.

Genetic Algorithms, Cont

- 1 Starts with a set(multi-set) of k randomly generated states.
- 2 Each state(individual) is represented as a chromosome (finite string).
- 3 Each chromosome is made up of some genome (characters).
- 4 Each individual is ranked by fitness function.
- 5 Fitness function should return higher value for better states.

Genetic Algorithms, Cont

- 1 Starts with a set(multi-set) of k randomly generated states.
- 2 Each state(individual) is represented as a chromosome (finite string).
- 3 Each chromosome is made up of some genome (characters).
- 4 Each individual is ranked by fitness function.
- 5 Fitness function should return higher value for better states.

Genetic Algorithms, Cont

- 1 Next **generation** is generated using current generation:
- 2 Cross over:
 - 1 Each individual is assigned a prob. based to its rank.
 - 2 Two pairs is selected randomly (according to their prob.)
 - 3 A crossover point is randomly chosen from a position in string.
 - 4 Two individuals are resulted from crossover.
- 3 Mutation:
 - 1 Each location(gnome) in is subject to random mutation with a small prob.
 - 2 A (new) value is assigned to the selected gnome.

Genetic Algorithms, Cont

- ① Next **generation** is generated using current generation:
- ② Cross over:
 - ① Each individual is assigned a prob. based to its rank.
 - ② Two pairs is selected randomly (according to their prob.)
 - ③ A crossover point is randomly chosen from a position in string.
 - ④ Two individuals are resulted from crossover.
- ③ Mutation:
 - ① Each location(gnome) in is subject to random mutation with a small prob.
 - ② A (new) value is assigned to the selected gnome.

Genetic Algorithms, Cont

- ① Next **generation** is generated using current generation:
- ② Cross over:
 - ① Each individual is assigned a prob. based to its rank.
 - ② Two pairs is selected randomly (according to their prob.)
 - ③ A crossover point is randomly chosen from a position in string.
 - ④ Two individuals are resulted from crossover.
- ③ Mutation:
 - ① Each location(gnome) in is subject to random mutation with a small prob.
 - ② A (new) value is assigned to the selected gnome.

Genetic Algorithms, Cont

- ① Next **generation** is generated using current generation:
- ② Cross over:
 - ① Each individual is assigned a prob. based to its rank.
 - ② Two pairs is selected randomly (according to their prob.)
 - ③ A crossover point is randomly chosen from a position in string.
 - ④ Two individuals are resulted from crossover.
- ③ Mutation:
 - ① Each location(gnome) in is subject to random mutation with a small prob.
 - ② A (new) value is assigned to the selected gnome.

Genetic Algorithms, Cont

- ① Next **generation** is generated using current generation:
- ② Cross over:
 - ① Each individual is assigned a prob. based to its rank.
 - ② Two pairs is selected randomly (according to their prob.)
 - ③ A crossover point is randomly chosen from a position in string.
 - ④ Two individuals are resulted from crossover.
- ③ Mutation:
 - ① Each location(gnome) in is subject to random mutation with a small prob.
 - ② A (new) value is assigned to the selected gnome.

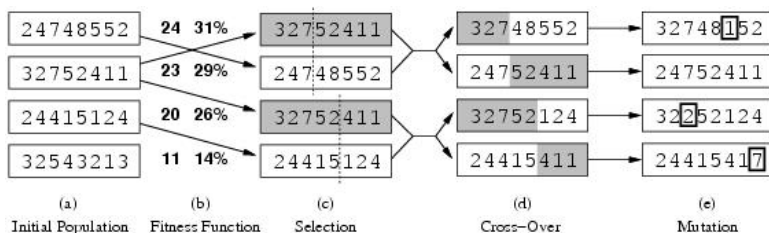
Genetic Algorithms, Cont

- ① Next **generation** is generated using current generation:
- ② Cross over:
 - ① Each individual is assigned a prob. based to its rank.
 - ② Two pairs is selected randomly (according to their prob.)
 - ③ A crossover point is randomly chosen from a position in string.
 - ④ Two individuals are resulted from crossover.
- ③ Mutation:
 - ① Each location(gnome) in is subject to random mutation with a small prob.
 - ② A (new) value is assigned to the selected gnome.

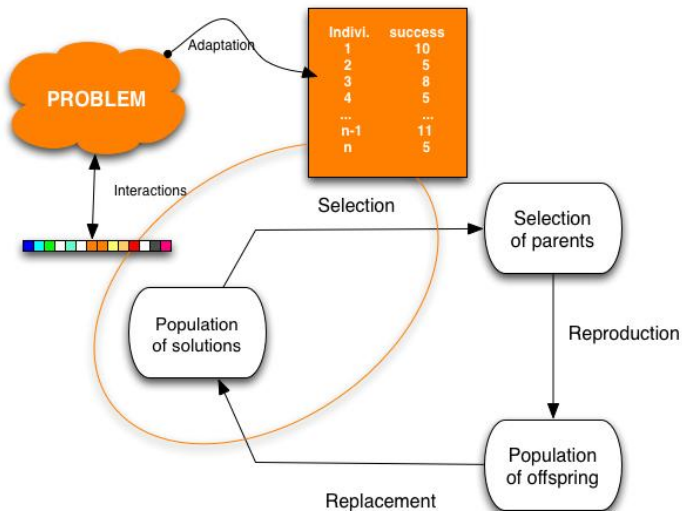
Genetic Algorithms, Cont

- ① Next **generation** is generated using current generation:
- ② Cross over:
 - ① Each individual is assigned a prob. based to its rank.
 - ② Two pairs is selected randomly (according to their prob.)
 - ③ A crossover point is randomly chosen from a position in string.
 - ④ Two individuals are resulted from crossover.
- ③ Mutation:
 - ① Each location(gnome) in is subject to random mutation with a small prob.
 - ② A (new) value is assigned to the selected gnome.

Genetic Algorithms, Cont



Genetic Algorithms, Cont



Genetic Algorithms, Cont

```
function GENETIC_ALGORITHM (population, FN)
repeat
  new_population= empty set
  for i= 1 to SIZE(population) do
    x= RANDOM_SELECTION (population, FITNESS_FN)
    y= RANDOM_SELECTION (population, FITNESS_FN)
    child= REPRODUCE(x,y)
    if (small random probability) then
      child= MUTATE(child)
    add child to new_population
  population= new_population
until some condition
return the best individual
```

Genetic Algorithms, 8 puzzle

- 1 Chromosome: Position of 8 queens one in separate col.
- 2 Genome: Index of the row.
- 3 Fitness function: No. of non-attacking queens.
- 4 Crossover:

Genetic Algorithms, 8 puzzle

- 1 Chromosome: Position of 8 queens one in separate col.
- 2 Genome: Index of the row.
- 3 Fitness function: No. of non-attacking queens.
- 4 Crossover:

Genetic Algorithms, 8 puzzle

- 1 Chromosome: Position of 8 queens one in separate col.
- 2 Genome: Index of the row.
- 3 Fitness function: No. of non-attacking queens.
- 4 Crossover:

Genetic Algorithms, 8 puzzle

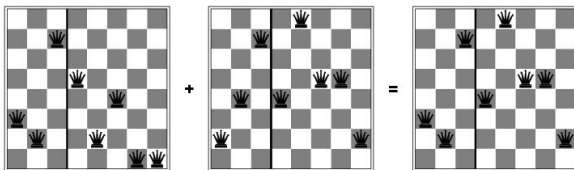
- 1 Chromosome: Position of 8 queens one in separate col.
- 2 Genome: Index of the row.
- 3 Fitness function: No. of non-attacking queens.
- 4 Crossover:

Genetic Algorithms, 8 puzzle

- 1 Chromosome: Position of 8 queens one in separate col.
- 2 Genome: Index of the row.
- 3 Fitness function: No. of non-attacking queens.
- 4 Crossover:

Genetic Algorithms, 8 puzzle

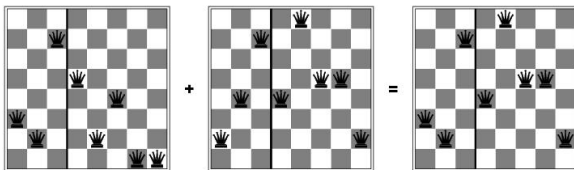
- 1 Chromosome: Position of 8 queens one in separate col.
- 2 Genome: Index of the row.
- 3 Fitness function: No. of non-attacking queens.
- 4 Crossover:



- 5 Mutation: Change a queen's place

Genetic Algorithms, 8 puzzle

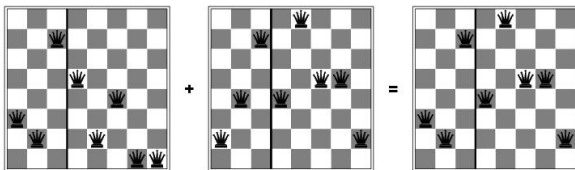
- 1 Chromosome: Position of 8 queens one in separate col.
- 2 Genome: Index of the row.
- 3 Fitness function: No. of non-attacking queens.
- 4 Crossover:



- 5 Mutation: Change a queen's place

Genetic Algorithms, 8 puzzle

- 1 Chromosome: Position of 8 queens one in separate col.
- 2 Genome: Index of the row.
- 3 Fitness function: No. of non-attacking queens.
- 4 Crossover:



- 5 Mutation: Change a queen's place

Genetic Algorithms, Minimum cost matching

- 1 Chromosome: ?
- 2 Genome: ?
- 3 Crossover: ?
- 4 Mutation: ?
- 5 Fitness function: ?

Genetic Algorithms, Minimum cost matching

- ① Chromosome: ?
- ② Genome: ?
- ③ Crossover: ?
- ④ Mutation: ?
- ⑤ Fitness function: ?

Genetic Algorithms, Minimum cost matching

- 1 Chromosome: ?
- 2 Genome: ?
- 3 Crossover: ?
- 4 Mutation: ?
- 5 Fitness function: ?

Genetic Algorithms, Minimum cost matching

- 1 Chromosome: ?
- 2 Genome: ?
- 3 Crossover: ?
- 4 Mutation: ?
- 5 Fitness function: ?

Genetic Algorithms, Minimum cost matching

- 1 Chromosome: ?
- 2 Genome: ?
- 3 Crossover: ?
- 4 Mutation: ?
- 5 Fitness function: ?

Genetic Algorithms, TSP

- 1 Exercise 6: A weighted graph is given, find a Hamiltonian cycle.
- 2 Chromosome: ?
- 3 Genome: ?
- 4 Crossover: ?
- 5 Mutation: ?
- 6 Fitness function: ?

Genetic Algorithms, Summary

- 1 Having enough time, it can be proved that it finds the global minimum.
- 2 Chromosome encoding is important.
- 3 Mutation is a guaranty that the whole space is being searched.
- 4 Can be used to solve many problems.

Genetic Algorithms, Summary

- 1 Having enough time, it can be proved that it finds the global minimum.
- 2 Chromosome encoding is important.
- 3 Mutation is a guaranty that the whole space is being searched.
- 4 Can be used to solve many problems.

Genetic Algorithms, Summary

- 1 Having enough time, it can be proved that it finds the global minimum.
- 2 Chromosome encoding is important.
- 3 Mutation is a guaranty that the whole space is being searched.
- 4 Can be used to solve many problems.

Genetic Algorithms, Summary

- ① Having enough time, it can be proved that it finds the global minimum.
- ② Chromosome encoding is important.
- ③ Mutation is a guaranty that the whole space is being searched.
- ④ Can be used to solve many problems.

Online search

- 1 Offline: solution is determined before executing it.
- 2 Online: interleaving computation and action.
- 3 Dynamic and semi-dynamic environments needs online search.
- 4 Online search can be used for exploration problems.

Online search

- 1 Offline: solution is determined before executing it.
- 2 Online: interleaving computation and action.
- 3 Dynamic and semi-dynamic environments needs online search.
- 4 Online search can be used for exploration problems.

Online search

- 1 Offline: solution is determined before executing it.
- 2 Online: interleaving computation and action.
- 3 Dynamic and semi-dynamic environments needs online search.
- 4 Online search can be used for exploration problems.

Online search problems

- 1 In these problems, agent knows:
 - 1 ACTIONS (s): lists all allowed action in state s .
 - 2 $C(s, a, s')$: step cost function.
 - 3 GOAL-TEST (s).
- 2 Actions are deterministic.
- 3 Agent can recognize previously visited states.
- 4 May have access to a heuristic function.

Online search problems

- 1 In these problems, agent knows:
 - 1 ACTIONS (s): lists all allowed action in state s .
 - 2 $C(s, a, s')$: step cost function.
 - 3 GOAL-TEST (s).
- 2 Actions are deterministic.
- 3 Agent can recognize previously visited states.
- 4 May have access to a heuristic function.

Online search problems

- 1 In these problems, agent knows:
 - 1 ACTIONS (s): lists all allowed action in state s .
 - 2 $C(s, a, s')$: step cost function.
 - 3 GOAL-TEST (s).
- 2 Actions are deterministic.
- 3 Agent can recognize previously visited states.
- 4 May have access to a heuristic function.

Online search problems

- 1 In these problems, agent knows:
 - 1 ACTIONS (s): lists all allowed action in state s .
 - 2 $C(s, a, s')$: step cost function.
 - 3 GOAL-TEST (s).
- 2 Actions are deterministic.
- 3 Agent can recognize previously visited states.
- 4 May have access to a heuristic function.

Online search problems

- 1 In these problems, agent knows:
 - 1 ACTIONS (s): lists all allowed action in state s .
 - 2 $C(s, a, s')$: step cost function.
 - 3 GOAL-TEST (s).
- 2 Actions are deterministic.
- 3 Agent can recognize previously visited states.
- 4 May have access to a heuristic function.

Online search problems

- 1 In these problems, agent knows:
 - 1 ACTIONS (s): lists all allowed action in state s .
 - 2 $C(s, a, s')$: step cost function.
 - 3 GOAL-TEST (s).
- 2 Actions are deterministic.
- 3 Agent can recognize previously visited states.
- 4 May have access to a heuristic function.

Online search problems

- ① In these problems, agent knows:
 - ① ACTIONS (s): lists all allowed action in state s .
 - ② $C(s, a, s')$: step cost function.
 - ③ GOAL-TEST (s).
- ② Actions are deterministic.
- ③ Agent can recognize previously visited states.
- ④ May have access to a heuristic function.

Online search problems, Cont

- 1 Agent wants to reach to goal while minimizing the path cost.
- 2 Path cost: total cost of travelled path.
- 3 Competitive ratio: comparison of cost with cost of the solution path if search space is known.
- 4 Can be infinite if the agent reaches dead ends.
- 5 There is no online search that can avoid deal ends.

Online search problems, Cont

- 1 Agent wants to reach to goal while minimizing the path cost.
- 2 Path cost: total cost of travelled path.
- 3 Competitive ratio: comparison of cost with cost of the solution path if search space is known.
- 4 Can be infinite if the agent reaches dead ends.
- 5 There is no online search that can avoid deal ends.

Online search problems, Cont

- 1 Agent wants to reach to goal while minimizing the path cost.
- 2 Path cost: total cost of travelled path.
- 3 Competitive ratio: comparison of cost with cost of the solution path if search space is known.
- 4 Can be infinite if the agent reaches dead ends.
- 5 There is no online search that can avoid deal ends.

Online search problems, Cont

- 1 Agent wants to reach to goal while minimizing the path cost.
- 2 Path cost: total cost of travelled path.
- 3 Competitive ratio: comparison of cost with cost of the solution path if search space is known.
- 4 Can be infinite if the agent reaches dead ends.
- 5 There is no online search that can avoid deal ends.

Online search problems, Cont

- 1 Agent wants to reach to goal while minimizing the path cost.
- 2 Path cost: total cost of travelled path.
- 3 Competitive ratio: comparison of cost with cost of the solution path if search space is known.
- 4 Can be infinite if the agent reaches dead ends.
- 5 There is no online search that can avoid deal ends.

Online search agent

- 1 After each action, it receives a percept telling it what state has reached.
- 2 Agent can build/upadte its map from the environment.
- 3 Expansion of the nodes are different for online and offline agents(search).
- 4 Travelling from one node to another node is time-consuming.
- 5 An online agent can only expand the node it is physically in (local order).

Online search agent

- ➊ After each action, it receives a percept telling it what state has reached.
- ➋ Agent can build/upadte its map from the environment.
- ➌ Expansion of the nodes are different for online and offline agents(search).
- ➍ Travelling from one node to another node is time-consuming.
- ➎ An online agent can only expand the node it is physically in (local order).

Online search agent

- ➊ After each action, it receives a percept telling it what state has reached.
- ➋ Agent can build/upadte its map from the environment.
- ➌ Expansion of the nodes are different for online and offline agents(search).
- ➍ Travelling from one node to another node is time-consuming.
- ➎ An online agent can only expand the node it is physically in (local order).

Online search agent

- ➊ After each action, it receives a percept telling it what state has reached.
- ➋ Agent can build/upadte its map from the environment.
- ➌ Expansion of the nodes are different for online and offline agents(search).
- ➍ Travelling from one node to another node is time-consuming.
- ➎ An online agent can only expand the node it is physically in (local order).

Online search agent

- ➊ After each action, it receives a percept telling it what state has reached.
- ➋ Agent can build/upadte its map from the environment.
- ➌ Expansion of the nodes are different for online and offline agents(search).
- ➍ Travelling from one node to another node is time-consuming.
- ➎ An online agent can only expand the node it is physically in (local order).

Online search agent, Cont

- 1 DFS is a good choice for online agents, because of locality.
- 2 Online-DFS works in environments (state spaces) where actions are reversible.

Online search agent, Cont

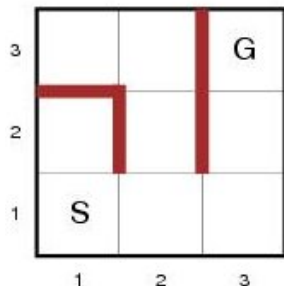
- 1 DFS is a good choice for online agents, because of locality.
- 2 Online-DFS works in environments (state spaces) where actions are reversible.

Online-DFS

```
function ONLINE_DFS-AGENT(s') return an action

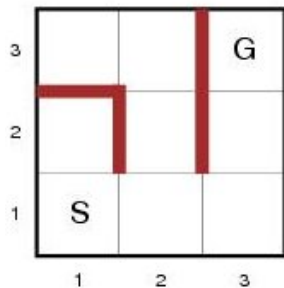
  if GOAL-TEST(s') then return stop
  if s' is a new state then unexplored[s'] = ACTIONS(s')
  if s is not null then
    result[a,s] = s'
    add s to the front of unbacktracked[s']
  if unexplored[s'] is empty then
    if unbacktracked[s'] is empty then return stop
  else
    a = b such that result[b, s'] = POP(unbacktracked[s'])
  else a = POP(unexplored[s'])
s = s'
return a
```

Online-DFS, Example



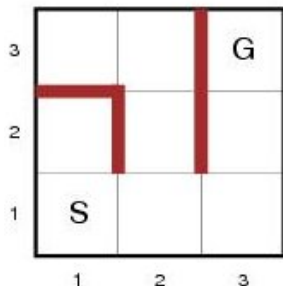
- 1 Assume maze problem on 3x3 grid.
- 2 $s' = (1,1)$ is initial state.
- 3 Result, unexplored (UX), unbacktracked (UB) are empty.
- 4 s, a are also empty.

Online-DFS, Example



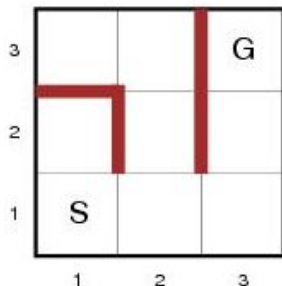
- 1 Assume maze problem on 3x3 grid.
- 2 $s' = (1,1)$ is initial state.
- 3 Result, unexplored (UX), unbacktracked (UB) are empty.
- 4 s, a are also empty.

Online-DFS, Example



- 1 Assume maze problem on 3x3 grid.
- 2 $s' = (1,1)$ is initial state.
- 3 Result, unexplored (UX), unbacktracked (UB) are empty.
- 4 s, a are also empty.

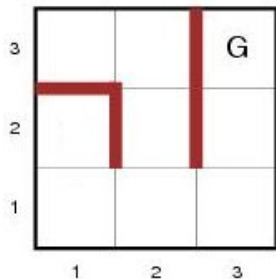
Online-DFS, Example



- 1 Assume maze problem on 3x3 grid.
- 2 $s' = (1,1)$ is initial state.
- 3 Result, unexplored (UX), unbacktracked (UB) are empty.
- 4 s, a are also empty.

Online-DFS, Example

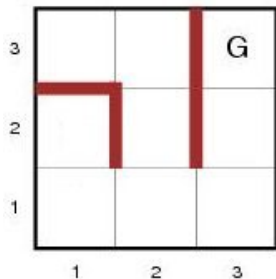
$$S' = (1, 1)$$



- ❶ GOAL-TEST($((1,1))$) fails.
- ❷ $(1,1)$ a new state.
- ❸ $UX[(1,1)] = \textit{Right, Up}$.
- ❹ $UX[(1,1)]$ is not empty.
- ❺ $a = \text{POP}(UX[(1,1)]) = \textit{UP}$.
- ❻ $s = (1,1)$.
- ❼ Return a .

Online-DFS, Example

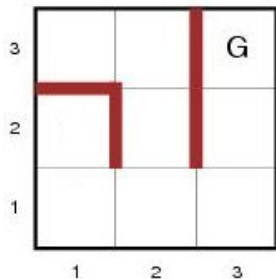
$$S' = (2, 1)$$



- ❶ GOAL-TEST($((2,1))$) fails.
- ❷ $(2,1)$ is a new state.
- ❸ $UB[(2,1)]_i = (1,1)$.
- ❹ $a = \text{POP}(\text{UX}[(2,1)]) = \text{Down}$.
- ❺ $s = (2,1)$.

Online-DFS, Example

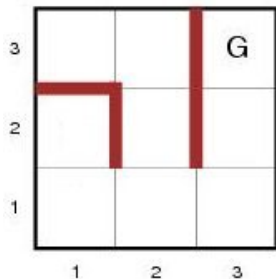
$$S' = (1, 1)$$



- ❶ GOAL-TEST($((1,1))$) fails.
- ❷ $(1,1)$ is not a new state.
- ❸ $UB[(1,1)] = (2, 1)$.
- ❹ $a = \text{POP}(\text{UX}[(1,1)]) = \text{Right}$
- ❺ $s = (1,1)$

Online-DFS, Example

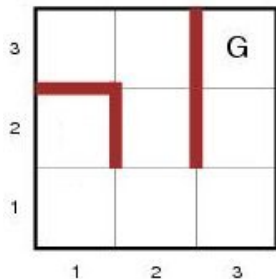
$$S' = (1, 2)$$



- ❶ GOAL-TEST($((1,2))$) fails
- ❷ $(1,2)$ is a new state.
- ❸ $UX[(1,1)] = \text{Right, Up, Left}$.
- ❹ $UB[(1,2)] = (1, 1)$.
- ❺ $a = \text{POP}(UX[(1,2)]) = \text{Left}$
- ❻ $s = (1,2)$

Online-DFS, Example

$$S' = (1, 1)$$



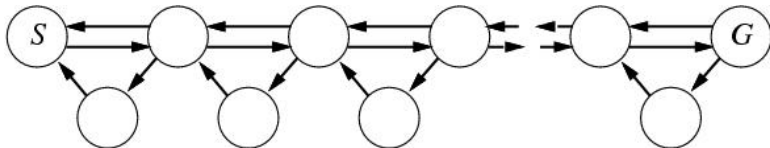
- ① GOAL-TEST($((1,1))$) fails.
- ② $(1,1)$ is not a new state.
- ③ $UX[(1,1)] = \cdot$
- ④ $UB[(1,1)] = (2, 1), (1, 2)$.
- ⑤ $b = \text{Right}$ and $UB[(1,1)] = (2, 1)$.
- ⑥ $s = (1,1)$

Online-DFS, Cont

- 1 In worst case, each node is visited twice.
- 2 DFS works where the actions are reversible.
- 3 There are some more complex alg. that works in general state spaces.
- 4 There is no alg whose competitive ratio is not bounded (in general state spaces).

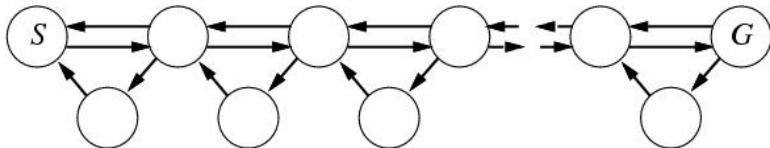
Online search agent, Hill-climbing

- 1 Hill-climbing stores just one node.
- 2 Bad performance due to local maxima.
- 3 Random restart strategy is impossible.
- 4 Solution: Random walk introduces exploration (can produce exponentially many steps)



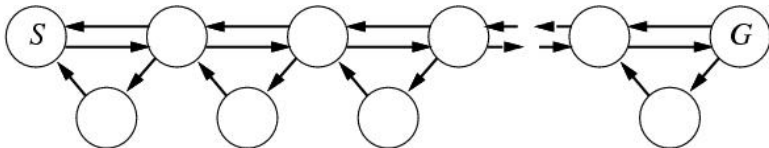
Online search agent, Hill-climbing

- 1 Hill-climbing stores just one node.
- 2 Bad performance due to local maxima.
- 3 Random restart strategy is impossible.
- 4 Solution: Random walk introduces exploration (can produce exponentially many steps)



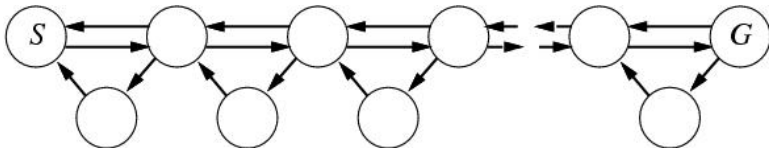
Online search agent, Hill-climbing

- 1 Hill-climbing stores just one node.
- 2 Bad performance due to local maxima.
- 3 Random restart strategy is impossible.
- 4 Solution: Random walk introduces exploration (can produce exponentially many steps)



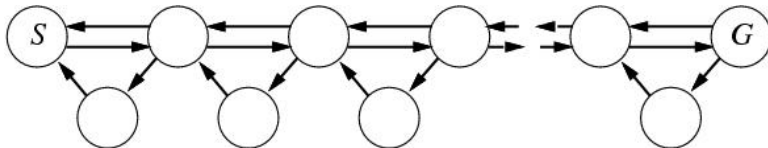
Online search agent, Hill-climbing

- 1 Hill-climbing stores just one node.
- 2 Bad performance due to local maxima.
- 3 Random restart strategy is impossible.
- 4 Solution: Random walk introduces exploration (can produce exponentially many steps)



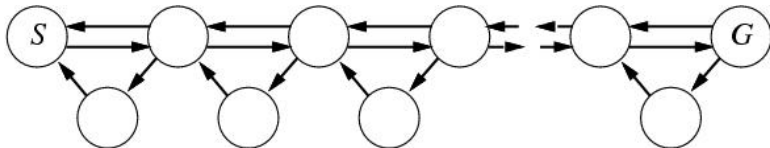
Online search agent, Hill-climbing

- 1 Hill-climbing stores just one node.
- 2 Bad performance due to local maxima.
- 3 Random restart strategy is impossible.
- 4 Solution: Random walk introduces exploration (can produce exponentially many steps)



Online search agent, Hill-climbing

- 1 Hill-climbing stores just one node.
- 2 Bad performance due to local maxima.
- 3 Random restart strategy is impossible.
- 4 Solution: Random walk introduces exploration (can produce exponentially many steps)



Online search agent, Learning real-time A^*

```
function LRTA*-COST(s,a,s,H) return an cost estimate
  if s is undefined then return h(s)
  else return c(s,a,s) + H[s]
```

```
function LRTA*-AGENT(s) return an action
  if GOAL-TEST(s) then return stop
  if s is a new state (not in H) then H[s]= h(s)
  unless s is null
    result[a,s]= s
    H[s]= MIN LRTA*-COST(s,b,result[b,s],H)
  a= an action b in ACTIONS(s) that minimizes LRTA*-COST(s,
  s= s
return a
```

Online search agent, $LRTA^*$

