Problem Solving & Search

Informatics Engineering Study Program
School of Electrical Engineering and Informatics

Institute of Technology Bandung

Contents

- Review
- Problem Solving
- Example of Problem
- Formal Definition
- Search → UnInformed and Informed
- Introduction to Constraint Satisfaction Problem (CSP)

Review

- What is AI → 4 approaches
  - we use 4th approach
  - Rationality ≠ omniscience ≠ success
  - Perfect rationality → limited rationality
- Intelligent Agent
  - Specifying task environment: PEAS
  - Properties/classes of task environment

Classes of Environments

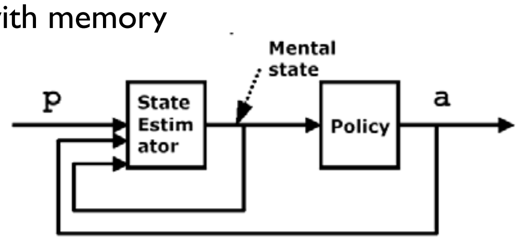
- Accessible (vs. Inaccessible) / Fully (vs partially) observable
  - Observable: sensors → relevant information
  - Can you see the state of the world directly?
- Deterministic (vs. Non-Deterministic)
  - Does an action map one state into a single other state?
  - No uncertainty
- Static (vs. Dynamic)
  - Can the world change while you are thinking?
  - Formulating and solving problem without paying attention to any changes that might be occuring in the environment
- Discrete (vs. Continuous)
  - Are the percepts and actions discrete (like integers) or continuous (like reals)?
- Episodic vs sequential, single vs multi agent

Example

- Vacuum Cleaner: partially observable, deterministic, static/semidynamic, discrete, episodic, single agent
- Taxi driving: partially observable, stochastic, sequential, dynamic, continuous, multi-agent
- News classification: fully observable, deterministic, episodic, semidynamic, continuous, single agent

Structures of Agent

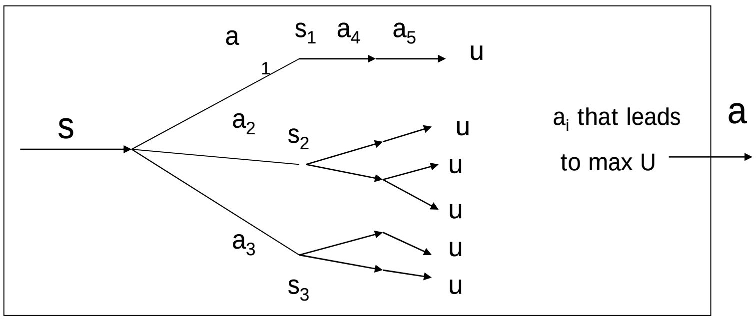
- Agent with memory



- State estimator/Memory
  - What we've chosen to remember from the history of percepts
  - Maps what you knew before, what you just perceived and what you just did, into what you know now.
- Problem of behavior: Given my mental state, what action should I take?

Planning Agent Policy

- Planning is explicitly considering future consequences of actions in order to choose the best one.
- So, planning is the process of generating possible sequences of actions, simulating their consequences, picking which is the best and committing to one of these actions.



Problem Solving Agent

- Agent design:
  - formulate problem → search solution → execute
- Problem: satisfy goal (goal state)
  - Agent task: find out which sequence of actions will get it to a goal state
  - 4 components of a problem: initial state, operator/successor function, goal test, path cost
- Searching: process of looking for sequence of action
- Solution: sequence of action to goal state

Problem Solving

- Agent knows world dynamics
  - World states, actions
  - [when agent doesn't know → learning]
- World state is finite, small enough to enumerate
  - [when state is infinite →logic]
- World is deterministic
  - [when non-deterministic →uncertainty]
- Agent knows current state
  - [when agent doesn't know → logic, uncertainty]
- Utility for a sequence of states is a sum over path

Few real problems are like this, but this may be a useful abstraction of a real problem → searching

Problem: Formal Definition

Problem components:

- Set of states: S
  - State space forms graph (node: state, arc: action)
  - Initial state, goal state
- Operators (actions): S → S [deterministic]
- Goal test: S → { t,f }
- Path cost: (S,O)* → real
  - Sum of costs: Σ c(S,O)

- Solution: graph path
- Criteria for algorithms:
  - Computation time/space
  - Solution quality

Example: Route Planning in a Map

A map is a graph where nodes are cities and links are roads. This is an abstraction of the real world.

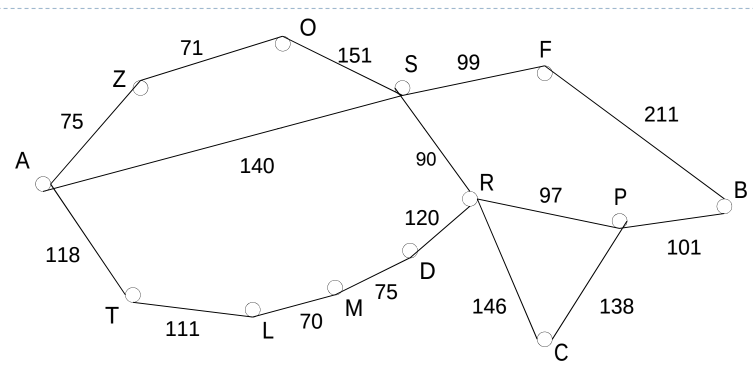
- Map gives world dynamics: starting at city X on the map and taking some road gets you to city Y.

Environment assumptions:

- Static: no change when solving problem
- Discrete: World (set of cities) is finite and enumerable.
- Deterministic: taking a given road from a given city leads to only one possible destination.
- Observable: information is complete
  - We assume current state is known.
- Utility for a sequence of states is usually either total distance traveled on the path or total time for the path.

Search

Source: Russell's book



S: set of cities
i.s: A (Arad)
g.s: B (Bucharest)
Goal test: s = B ?
Path cost: time ~ distance

Search

- UnInformed/Blind Search
  - Look around, don't know where to find the right answer
  - No additional information beyond that provided in problem definitional
  - Example: DFS, BFS, IDS, UCS
- Informed/Heuristic Search
  - Know some information that sometimes helpful
  - Know whether one non-goal state is "more promising" than another
  - Example: Best FS, A*

Search

- It's time to do searching: covering the basic methods really fast.
  - Agenda: a list of states that are waiting to be expand

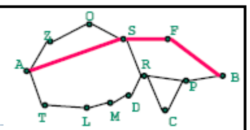
```
{Put start state (initial state) in the agenda}
AddState(Agenda, initial-state)
iterate
  GetState(Agenda, current-state)
stop: isGoal(current-state)
  if not isExpanded(current-state) then
    {put children in agenda}
    ExpandState(current-state, Agenda)
```

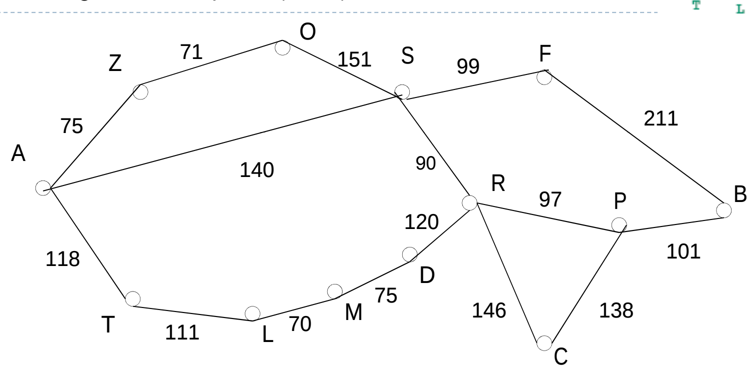
Search

- It's time to do searching: covering the basic methods really fast.
  - Graph search
  - Agenda: a list of states that are waiting to be expand
  - Which state is chosen from the agenda defines the type of search & may have huge impact on effectiveness.

Breadth-First Search (BFS)

Treat agenda as a queue (FIFO)





A → Z_A, S_A, T_A → S_A, T_A, O_{AZ} → $T_A, O_{AZ}, O_{AS}, F_{AS}, R_{AS}$ →
$O_{AZ}, O_{AS}, F_{AS}, R_{AS}, L_{AT}$ → $O_{AS}, F_{AS}, R_{AS}, L_{AT}$ → F_{AS}, R_{AS}, L_{AT} → R_{AS}, L_{AT}, B_{ASF}
→ $L_{AT}, B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}$ → $B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}, M_{ATL}$ →
Stop: B=goal, path: A→ S → F → B, path-cost = 450

Breadth-First Search (BFS)

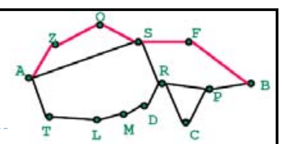
- Treat agenda as a queue (FIFO)
- Let's see what would happen if we did BFS on the graph G_1:
  - Start with initial state: A
  - Get A, expand it, add Z, S, T $\Rightarrow Z_A, S_A, T_A$
  - Get Z, expand it, add O $\Rightarrow S_A, T_A, O_{AZ}$
  - Get S, expand it, add O, F, R $\Rightarrow T_A, O_{AZ}, O_{AS}, F_{AS}, R_{AS}$
  - Get T, expand it, add L $\Rightarrow O_{AZ}, O_{AS}, F_{AS}, R_{AS}, L_{AT}$
  - Get O, expand it, nothing to add (already expanded) done twice!
  - Get F, expand it, add B $\Rightarrow R_{AS}, L_{AT}, B_{ASF}$
  - Get R, expand it, add D, C, P $\Rightarrow L_{AT}, B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}$
  - Get L, expand it, add M $\Rightarrow B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}, M_{ATL}$
  - Pop B, it is the goal state, and terminate.

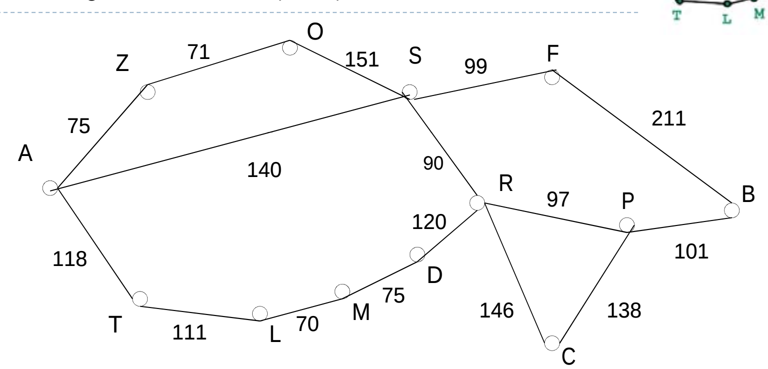
$\Rightarrow$ The RESULT is B_{ASF} with path: A, S, F, B

$\Rightarrow$ Path cost: 450

Depth-First Search (DFS)

Treat agenda as a stack (LIFO)





A → Z_A, S_A, T_A → O_{AZ}, S_A, T_A → S_{AZO}, S_A, T_A → $F_{AZOS}, R_{AZOS}, S_A, T_A$ → $B_{AZOSF}, R_{AZOS}, S_A, T_A$ →

Stop: B=goal, path: A→ Z → O → S → F → B, path-cost = 607

Depth-First Search (DFS)

- Agenda: stack (LIFO, top stack, push, pop)
  - Start with initial state: A {children of A: Z, S, T}
  - Pop A, expand it, and then push T,S,Z $\Rightarrow Z_A, S_A, T_A$
  - Pop Z, expand it, push O (A already expanded) $\Rightarrow O_{ZA}, S_A, T_A$
  - Pop O, expand it, push S $\Rightarrow S_{AZO}, S_A, T_A$
  - Pop S, expand it, push F, R $\Rightarrow F_{AZOS}, R_{AZOS}, S_A, T_A$
  - Pop F, expand it, push B $\Rightarrow B_{AZOSF}, R_{AZOS}, S_A, T_A$
  - Pop B, it is the goal state, and terminate. Ok!

$\Rightarrow$ The RESULT is B_{AZOSF} with path: A, Z, O, S, F, B

$\Rightarrow$ Path cost: 607

Depth-Limited Search

- BFS finds min-step path but requires exponential space
- DFS is efficient in space, but has no path-length guarantee
  - DFS: can make a wrong choice and get stuck going down a very long (or even infinite) path when a different choice would lead to a solution near root of the search tree
- Solution: DFS-limited search
  - DFS with a predetermined depth limit l
  - Nodes at depth l are treated as if they have no successors.
  - Problem: the shallowest goal is beyond the depth limit
  - Depth limit can be based on knowledge of the problem

DLS Algorithm

```
Function DLS (problem, limit)
→ rec_DLS(make_node(init_state),problem,limit)

Function Rec_DLS (node,problem, limit)
  if isGoal(node) then → solution(node)
  else if depth(node)=limit then → cutoff
  else
      for each successor in Expand(node,problem) do
        result ←rec_DLS(successor,problem,limit)
        if result=cutoff then cutoff_occured← true
        else if result≠failure then → result
  if cutoff_occured then → cutoff
  else → failure
```


DLS (2)

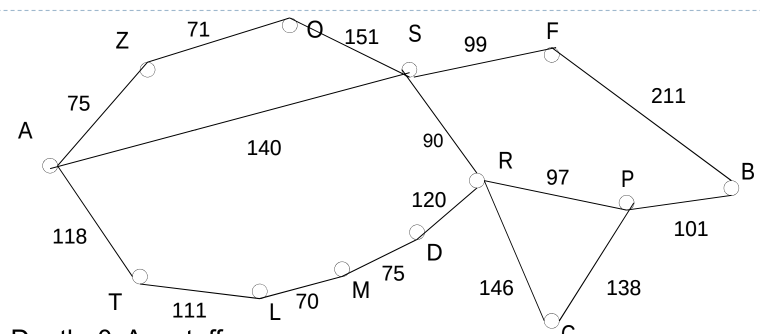
DFS cutoff depth	Space	Time
1	O(b)	O(b)
2	O(2b)	$O(b^2)$
3	O(3b)	$O(b^3)$
4	O(4b)	$O(b^4)$
…		
D	O(db)	$O(b^d)$
Total	Max = O(db)	Sum = $O(b^{d+1})$
BFS	$O(b^{d+1})$	$O(b^{d+1})$
DFS	O(bm)	$O(b^m)$

Iterative Deepening Search (IDS)

- IDS: perform a sequence of DFS searches with increasing depth-cutoff until goal is found
- Assumption: most of the nodes are in the bottom level so it does not matter much that upper levels are generated multiple times.

```
Depth ← 0
Iterate
  result← DLS(problem,depth)
stop: result ≠ cutoff
  depth← depth+1
→ result
```


IDS



Depth=0: A: cutoff

Depth=1: A → Z_A,S_A,T_A → Z_A: cutoff, S_A: cutoff, T_A: cutoff

Depth=2: A → Z_A,S_A,T_A → O_{AZ}, S_A,T_A → O_{AZ} : cutoff → F_{AS}, R_{AS},T_A → F_{AS} : cutoff → R_{AS} : cutoff → L_{AT} → L_{AT} : cutoff

Depth=3: A → Z_A,S_A,T_A → O_{AZ}, S_A,T_A → S_{AZO},S_A,T_A → S_{AZO}: cutoff → F_{AS}, R_{AS},T_A → B_{ASF}, R_{AS},T_A → B_{ASF}

Stop: B=goal, path: A→ S → F → B, path-cost = 450

Uniform Cost Search (UCS)

- BFS & IDS find path with fewest steps
- If steps ≠ cost, this is not relevant (to optimal solution)
- How can we find the shortest path (measured by sum of distances along path)?
- UCS:
  - Nodes in agenda keep track of total path length from start to that node
  - Agenda kept in priority queue ordered by path length
  - Get shortest path in queue
- Explores paths in contours of total path length; finds optimal path

Uniform Cost Search (UCS)

- Let's see what would happen if we did UCS on the graph G_1:
  - Start with start state: A
  - Remove A, add Z with cost 75, add T with cost 118, add S with cost 140 $\Rightarrow Z_{A-75}, T_{A-118}, S_{A-140}$
  - Remove Z (the shortest path), add its children: $O_{146} \Rightarrow T_{A-118}, S_{A-140}, O_{AZ-146}$
  - Remove T, add $L_{229} \Rightarrow S_{A-140}, O_{AZ-146}, L_{AT-229}$
  - Remove S, add $O_{291}, F_{239}, R_{230} \Rightarrow O_{AZ-146}, L_{AT-229}, R_{AS-230}, F_{AS-239}, O_{AS-291}$
  - Remove O, add nothing (already expanded)
  - Remove L, add $M_{299} \Rightarrow R_{AS-230}, F_{AS-239}, O_{AS-291}, M_{ATL-299}$
  - etc ...
- It seems clear that in the process of removing nodes from the agenda, we're enumerating all the paths in the graph in order of their length from the start state.

Informed Search

Best-first search

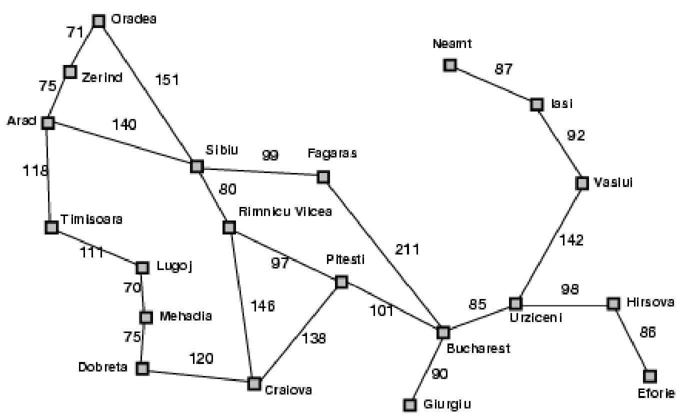
- Idea: use an evaluation function $f(n)$ for each node
  - estimate of "desirability"

  → Expand most desirable unexpanded node

- Implementation:
  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A^* search

Romania with step costs in km



Straight–line distance to Bucharest

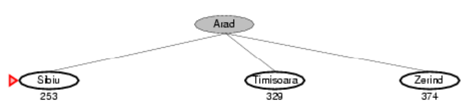
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

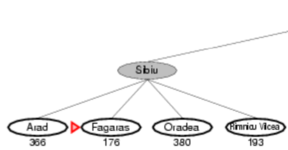
- Evaluation function $f(n) = h(n)$ (heuristic) = estimate of cost from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

Greedy best-first search example

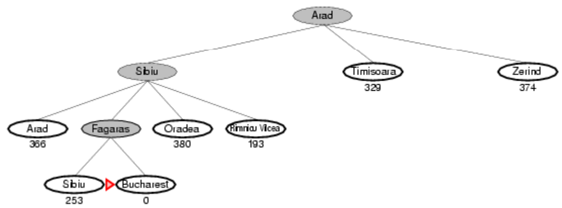
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example



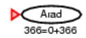
Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

A* search

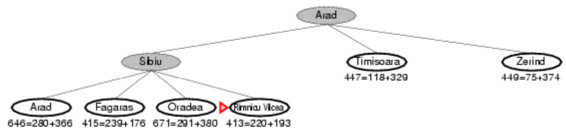
- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n)$ = estimated total cost of path through n to goal

A* search example

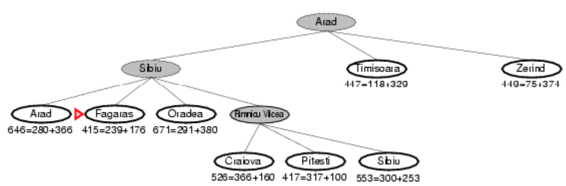


A* search example

A* search example

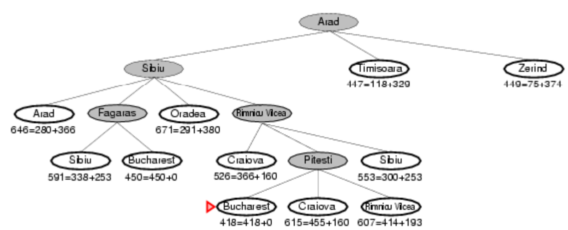


A* search example



A* search example

A* search example

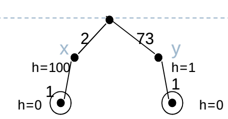


Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal

Admissibility

- What must be true about h for A* to find optimal path?
- A* finds optimal path if h is admissable; h is admissible when it never overestimates.
- In this example, h is not admissible.
- In route finding problems, straight-line distance to goal is admissible heuristic.



g(X)+h(X)=2+100=102

G(Y)+h(Y)=73+1=74

Optimal path is not found!

Because we choose Y, rather than X which is in the optimal path.

Local Search Algorithm

Hill-climbing search

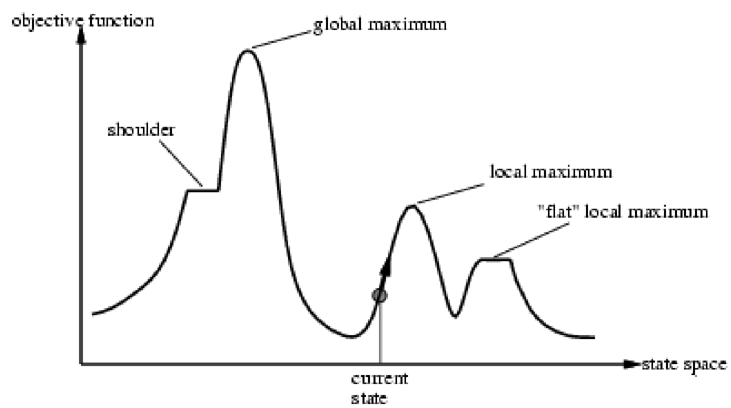
- "Like climbing Everest in thick fog with amnesia"

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  inputs: *problem*, a problem
  local variables: *current*, a node
    neighbor, a node

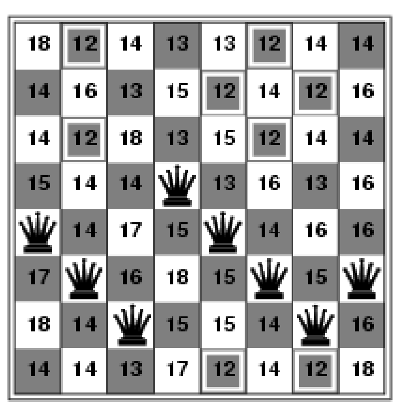
  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    neighbor ← a highest-valued successor of *current*
    if VALUE[neighbor] ≤ VALUE[current] **then return** STATE[*current*]
    current ← *neighbor*

Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima

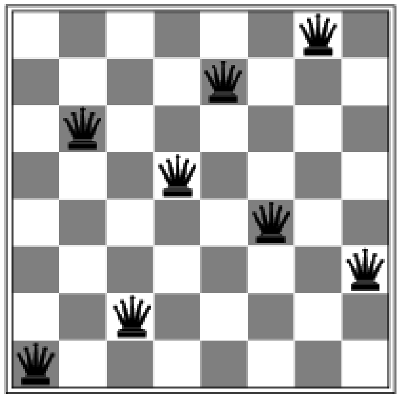


Hill-climbing search: 8-queens problem



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h* = *17* for the above state

Hill-climbing search: 8-queens problem



- A local minimum with *h = 1*

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[*t*]
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

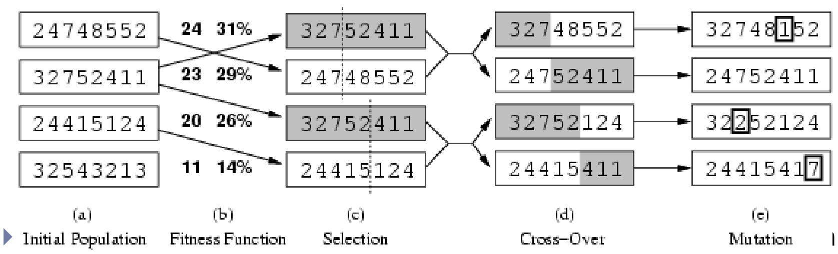
Properties of simulated annealing search

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Widely used in VLSI layout, airline scheduling, etc

Genetic algorithms

- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation

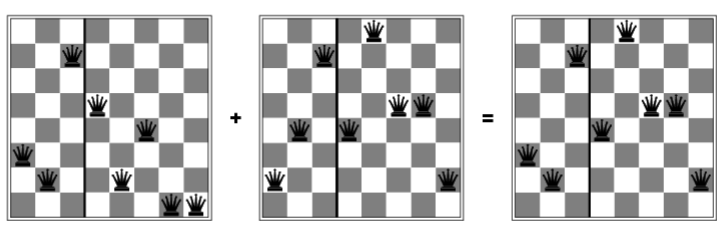
Genetic algorithms



= 0, max = $8 \times 7/2$ = 28)

- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms



Review

- Intelligent Agent → Solving simple problem (finite state, knows world dynamics, deterministic, knows current state, utility = sum over path)
- Searching
  - Uninformed: DFS, BFS, IDS, UCS
  - Informed: Greedy, A*, [Hill Climbing, Simulated Annealing, Genetic Algorithm → Local Search]
  - Heuristic function, must be admissible (path is the solution)
- Searching tools: http://www.aispace.org/downloads.shtml

THANK YOU