1

8/14/2025

Lectures 5B-7

Heuristics and Informed Search

Heuristics, Local, and Metaheuristics Searches

Textbook: Artificial Intelligence, A Modern Approach, IV edition
Authors: Russell and Norvig
Reading material: Chapter 3, sections 3.5 to 3.6, pp. 84-104
Chapter 4, section 4.1, pp. 110-119

1

2

8/14/2025

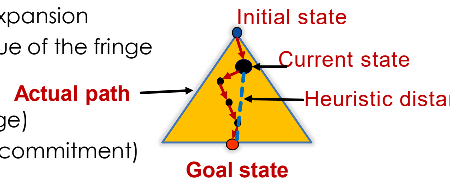
Outline

- A search strategy is used to expand a node
  - focuses the movement towards the goal by using a heuristic function
  - prunes other part of search space
- Types of search
  - best path search towards a goal state
  - finding the best goal state satisfying a final condition
- Best-path search
  - Greedy best-first search; best-first search with limited fringe; A* search
- Local search: hill-climbing search;
- Metaheuristics
  - simulated annealing search
  - local beam search; genetic algorithms

2

Best-first Search

- Use an optimistic heuristic function **h(current, goal_state)** to estimate the distance from the current node to goal node
  - expand child node with the shortest distance estimate from the current node to goal node, including actual distance from current node to its child node(s)
- Implementation
  - order the nodes in fringe in decreasing order of desirability
  - select the best child node for expansion
  - use a limited size of priority queue of the fringe
- Special cases
  - greedy best-first search (no fringe)
  - A* search (total cost, fringe, no commitment)



Best-first Search


```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s′ ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s′)
    yield NODE(STATE=s′, PARENT=node, ACTION=action, PATH-COST=cost)
```

Fringe → (frontier line)

% Frontier is implemented as a priority queue

% Cost of previously visiting the same node

Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

Roadmap of Romania

8/14/2025

Graph edges show actual distance.
dashed line shows h(n)

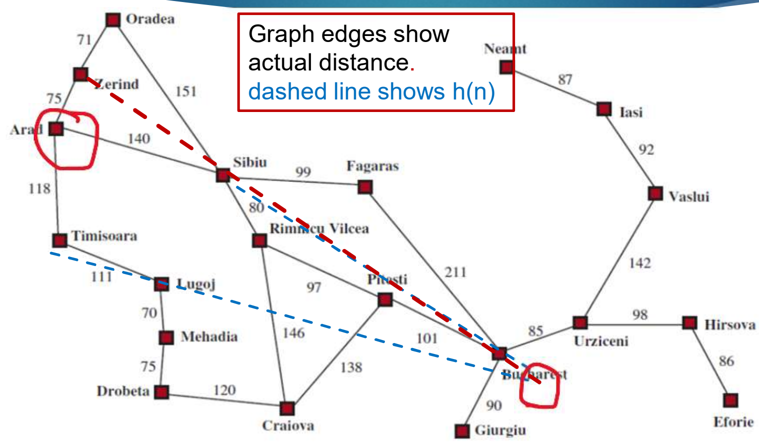


Figure from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

5

Best First Node Expansion

8/14/2025

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Heuristic distance estimate

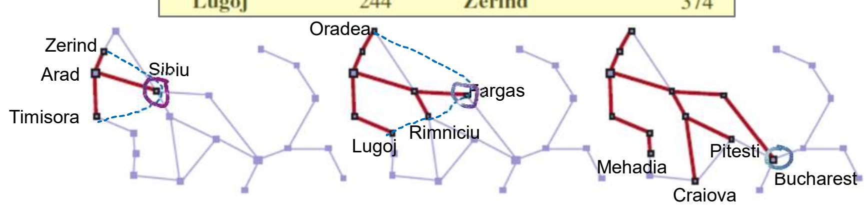


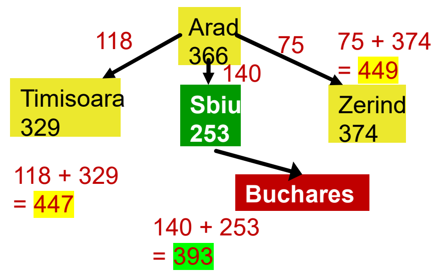
Figure from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

6

Greedy Best-first Search

7

- Heuristic function h(n)
  - estimate of cost from the current node n to the goal-node
  - example: h(n) = straight-line distance from n to Bucharest
- Greedy best-first search
  - **commits**, moves to, and expands the child-node with smallest estimated distance from the current node to goal node
  - **no nodes are retained in the memory after the commitment**
  - **Advantage**: memory saving, less computation
  - **Disadvantage:** not optimal and not complete



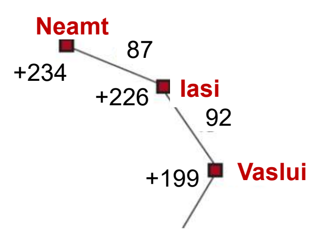
Commits to travel to Sibiu

7

Properties of Best-first

8

- Problem
  - **incomplete and not optimal**
  - can meet dead ends in the absence of good heuristic function
  - **may never terminate in the presence of a cycle**, e.g. Iasi → Neamt → Iasi → Neamt



- **Solution**
  - needs membership test to break the cycle, or
  - go to another slightly further node in the fringe and try again
- Time complexity: $O(b^m)$
  - b is the branching-factor and m is the depth of the tree
- Space complexity: O(bm) for general best-first search

  O(b) for greedy best first

8

A* Search

9

- **Uses total cost** = Actual Cost of past traversal + estimated cost using heuristic function
- Evaluation function **f(n) = g(n) + h(n)**
  - g(n) = actual cost to reach from initial node to current node n,
  - h(n) = estimated cost from the node n to the goal node
- Minimize f(n) to get the cheapest path
- Avoids expanding more expensive paths
- Can go back and forth after actual distance is discovered
  - stores traversed nodes and fringe in the memory
  - sorts and compares f(n) for every node in the fringe

  expensive
- A* search is optimal if h(n) never overestimates (optimistic)

9

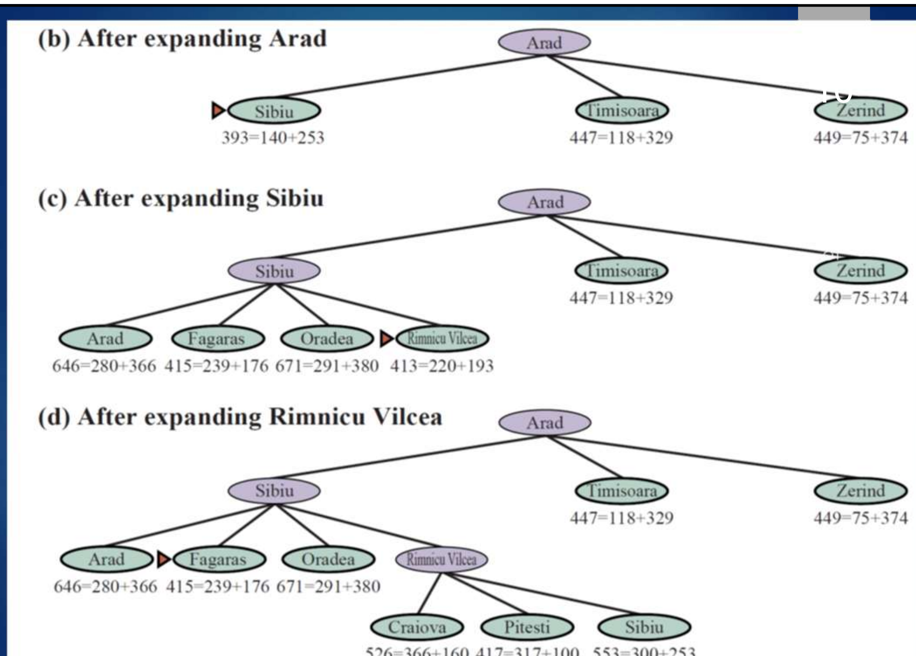


Figure from the book “Artificial Intelligence – A modern Approach by Russell and Norwig

10

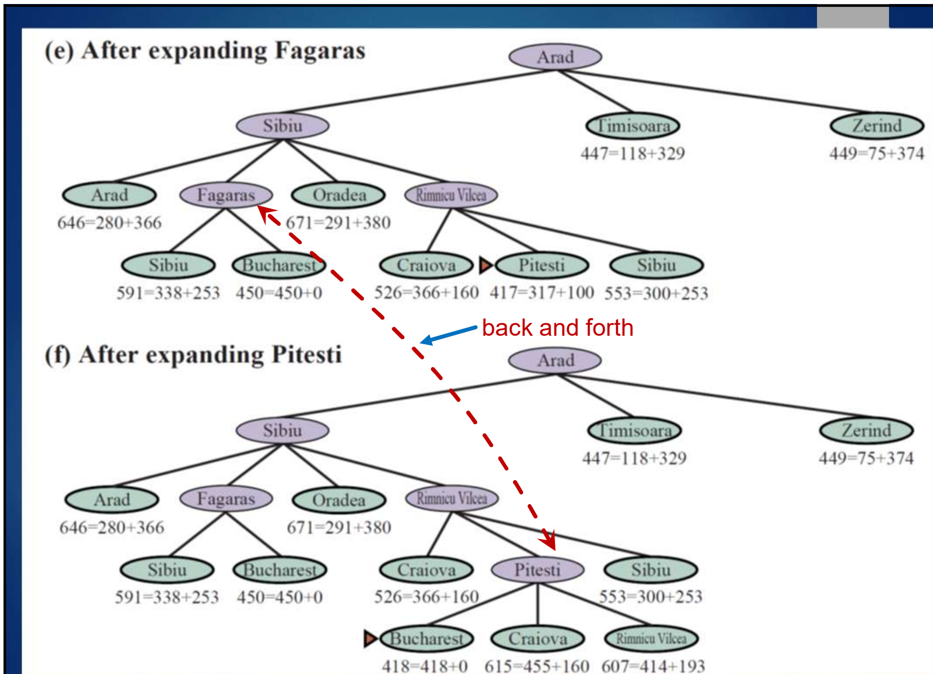


Figure from the book “Artificial Intelligence – A modern Approach by Russell and Norwig

11

Admissible Heuristics

12

- A heuristic h(n) is **admissible if for every node n, h(n) ≤ c(n),** where c(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
- **Theorem:** If h(n) is admissible, A* is optimal

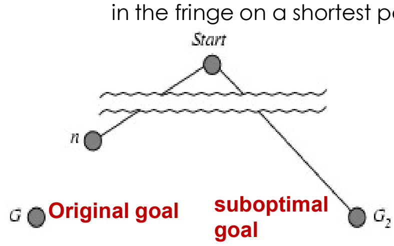
12

Optimality of A*(proof)

13

8/14/2025

- Assumptions
  - Let us assume there is a suboptimal goal G_2
  - Let n be an unexpanded node in the fringe on a shortest path



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$ (goal)
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ suboptimal eqn II
- $h(n) \le h^*(n)$ since h is admissible

  h^* is suboptimal
- $g(n) + h(n) \le g(n) + h^*(n)$
- $f(n) \le f(G_2)$
- Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

13

Consistent Heuristics (triangular inequality)

14

8/14/20

- A heuristic is **consistent** if for every node n, every successor n' of n generated by any action a, following triangular inequality is satisfied
- **Theorem:** If h(n) is consistent, A* is optimal
- $h(n) \le c(n, a, n') + h(n')$
- **Proof**

  If h is consistent, we have $f(n') = g(n') + h(n')$

  $= g(n) + c(n, a, n') + h(n')$

  $\ge g(n) + h(n) = f(n)$ - from consistency
- **f(n) is non-decreasing along any path.**

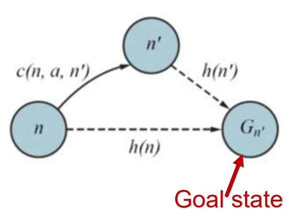


Figure from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

14

Problem of A* Search

15

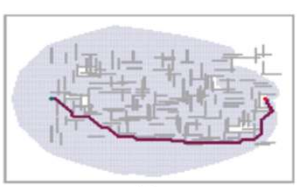
- Time complexity is exponential unless
  - The error function $|h(n) - c(n)| < O(\log c(n))$
- Space complexity is large
  - keeps all generated fringe nodes in the memory
  - exponential growth and **unsuitable for large problems**
- Good for small problems
- Improving A* algorithm
  - **reduce the difference between actual and estimated cost** by increasing weight to > 1 to multiply with h(n)
  - **use memory bounded heuristic search** schemes such as reference counts, beam search, IDA* or recursive best-first search

15

Weighted A* Search

16

- Node expansion is reduced by reducing difference $|h(n) - c(n)|$
- Achievable by multiplying h(n) with a weight > 1
  - A* search: g(n) + 1* h(n) (weight = 1)
  - Uniform cost search: g(n) + 0* h(n) (weight = 0)
  - Greedy best-first search: h(n) (no actual cost)
  - Weighted A*: g(n) + weight * h(n) (weight > 1)



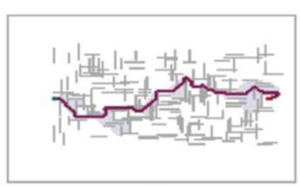


Figure taken from the book “Artificial Intelligence – A modern Approach by Russell and Norwig

16

Memory Bounded Schemes

17

8/14/2025

- Reference count
  - keep a node in reached node only if reference-count > 1
  - decrement reference count every-time a node is visited
- Beam search: spawn multiple threads and then focus on better threads
- IDA* - Iterative Deepening A*
  - cutoff is the smallest f-value > the cutoff from the previous iteration
  - suffers from multiple visit to the same nodes like iterative deepening
- Recursive best-first search
  - if current-path > f-value of the alternative path, then backtrack
  - more efficient than IDA* yet too many node regeneration
- Simplified Memory Bounded A*
  - delete the oldest worst leaf, and use the released memory

17

Recursive Best-first Search Algorithm

18

8/14/2025

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution or failure
   solution, fvalue ← RBFS(problem, NODE(problem.INITIAL), ∞)
  return solution

function RBFS(problem, node, f_limit) returns a solution or failure, and a new f-cost limit
  if problem.IS-GOAL(node.STATE) then return node
  successors ← LIST(EXPAND(node))
  if successors is empty then return failure, ∞
  for each s in successors do        // update f with value from previous search
     s.f ← max(s.PATH-COST + h(s), node.f))
  while true do
     best ← the node in successors with lowest f-value
     if best.f > f_limit then return failure, best.f
     alternative ← the second-lowest f-value among successors
     result, best.f ← RBFS(problem, best, min(f_limit, alternative))
     if result ≠ failure then return result, best.f
```

Algorithm taken from the book “Artificial Intelligence – A modern Approach by Russell and Norwig

18

Example of Recursive Best-first

19

- Arad to Sibiu
- Sibiu to Rimnicu Vilcea
  - Fagras is the alternative
- Fagras is chosen since
  - Pitesti cost > Fagaras cost
  - Alternative is Pitesti
- Pitesti is chosen since
  - Bucharest cost from Fagras is greater than Bucharest cost from Pitesti

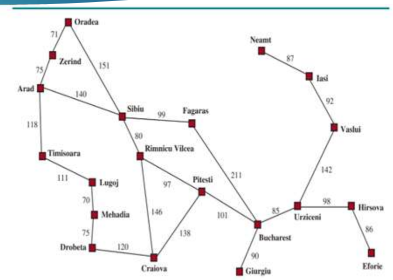


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Figure from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

19

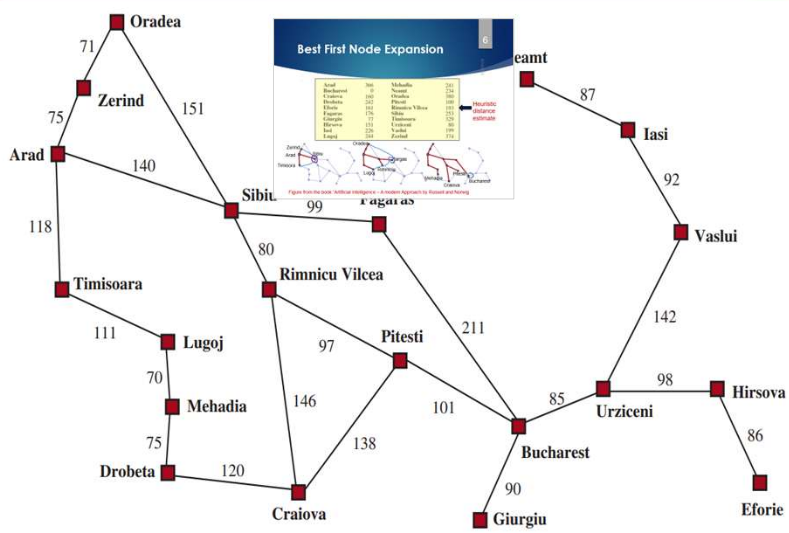


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Figure from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

20

Dominant Heuristic Functions

- Good heuristic functions
  - admissible – optimistic
  - dominant – expand fewer nodes.
  - cost closer to actual cost
- Example: tile problem
  - **h_1:** number of displaced tiles
  - **h_2:** sum of the distance of tiles from the goal state
  - **h_2 is dominant**

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

Tile #

$h_1(n) = 9$

$h_2(n) = 3(1) + 1(2) + 2(3) + 2(4) + 2(5) + 3(6) + 3(7) + 2(8) = 18$

depth **Number of moves**

A*(h_1)	6 (1.42)	12(1.45)	20 (1.50)	28 (1.49)
A*(h_2)	6 (1.34)	12 (1.28)	20 (1.34)	28 (1.36)

Searches (effective branching factor)

Designing Admissible Heuristic Functions

- Relax the problem and identify the cost of the optimal solution
  - follows triangular inequality for admissibility
- Identify heuristics cost function closest to actual cost without overestimating
  - more focused search and less branching
- Use pattern databases
  - identify intermediate nodes from where optimal solution is known
  - develop heuristic function to reach up to the intermediate nodes

Learning Heuristics

23

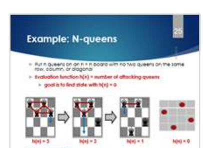
- Analyze multiple solutions to identify common patterns and rules
- Trace back from goal state and catalog the actual solutions
  - intermediate states and memorize the optimal solutions from the intermediate nodes
  - use statistics on solution costs on the intermediate nodes
  - analyze the solution paths to identify common patterns and features
- **$H(n) = c_1h_1(n) + \ldots + c_m h_m(n)$** where c_i is a constant and h_i are intermediate heuristics functions
- Combine multiple heuristics with different weights
  - **$H(n) = w_1 * h_1(n) + w_2 * h_2(n)$**

23

Local Search Algorithms

24

- Local Search
  - the distance to the goal-state cannot be estimated
  - use an evaluation function to evaluate the current state
  - move to the neighboring node with best evaluation function.
- Technique
  - start from any state.
  - iteratively move to a neighbor with best evaluation function
  - stop when you reach the final state
- Advantages
  - small memory requirement since nodes are not saved
- Issues
  - getting stuck in local maxima; ridges; shoulders

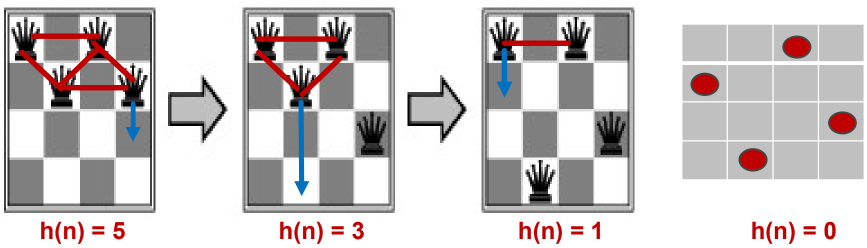


24

Example: N-queens

25

- Put n queens on an n × n board with no two queens on the same row, column, or diagonal
- **Evaluation function h(n) = number of attacking queens**
  - **goal is to find state with h(n) = 0**



25

State Space Landscape

26

- Model
  - needs an objective evaluation function to as the current state
  - global maximum is desira
- Hill-climbing (picking best neighbor) gives local max
- Shoulders, plateaus, and ridges are problematic

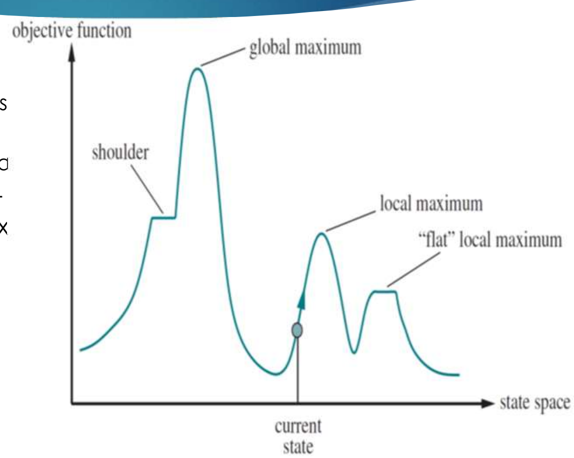


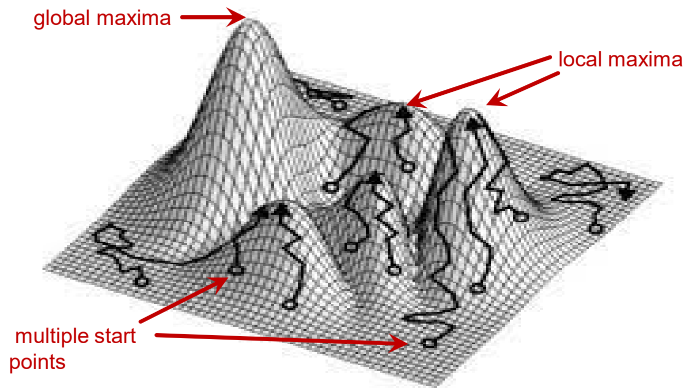
Figure from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

26

3-D Search Terrain

27

8/14/2025



Picture taken from local search lecture at CMU (credit: Prof Gianni De Caro) from website https://www.cs.cmu.edu/~arielpro/15381f16/c_slides/781f16-2a.pdf for teaching purpose only

27

Naïve Hill Climbing

28

8/14/20

- Moves to neighboring state with best value of evaluation function.
- Stops when final conditions are met, **or** number of iterations are over.
- Finds local maxima with respect to the initial state.

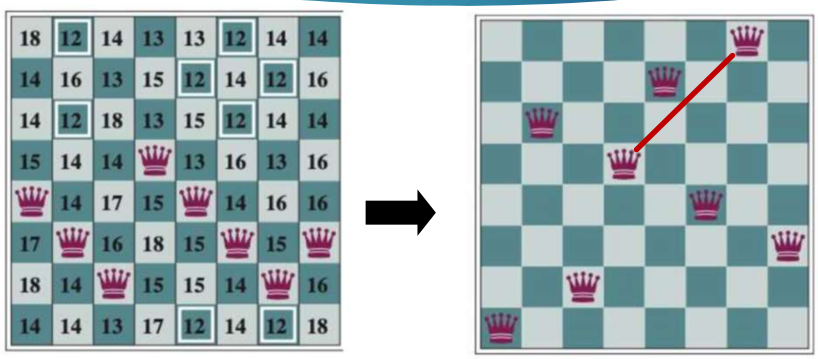
function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
    current $\leftarrow$ *problem*.INITIAL
    while *true* **do**
        neighbor $\leftarrow$ a highest-valued successor state of *current*
        if VALUE(*neighbor*) $\leq$ VALUE(*current*) **then return** *current*
        current $\leftarrow$ *neighbor*

Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

28

Hill-climbing Search - 8-queens Problem

29



h is pairs of attacking queens
current state h = 17

local maxima h = 1

Figure from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

29

Avoiding Local Maxima

30

- Use sideways movement with larger look-ahead step when two nodes with equal cost function identified
- Use random selection of initial states
- Stochastic hill climbing - use probability to pick up the next state from the fringe; not always picking up the steepest ascent
- Keep K best steepest ascent in fringe. choose next steepest ascent when first best hits local maxima
- Use random jump away when a local maxima is found
- Search more than one local space and converge after some time to investigate best local spaces

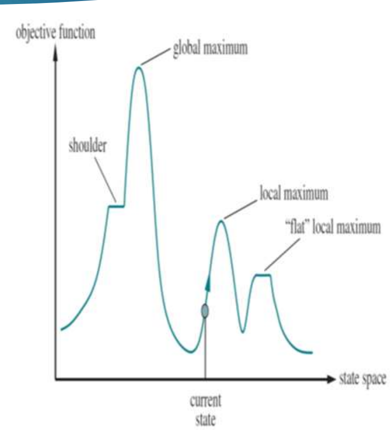


Figure from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

30

Metaheuristics Algorithms

31

8/14/2025

- Problems of classical algorithms
  - gradient-based search can not handle discontinuity; suffer from local maxima
  - stochastic algorithms use too much randomness increasing execution-time
- Metaheuristics algorithms
  - integrate both stochastic and gradient-based search
  - utilize randomness (diversification) to avoids local maxima
  - utilize gradient-based search to remain focused (intensification)
  - amplify the terrain near the goal state for better final state
- Types of metaheuristics algorithms
  - trajectory based: **simulated annealing**
  - **evolutionary algorithms**
  - **nature inspired**: ant colony, bee, fire-fly, bat, cuckoo, particle swarm etc.

31

Simulated Annealing

32

8/14/2025

- **Assumption:** energy (evaluation function value) decreases towards the goal state
- Start randomly and end with gradient-based search.
- initial temperature is high; initial probability of accepting any move is 1.
  - temperature decreases linearly or with geometric progression
- Except best child with $\Delta E < 0$.
- Accept undesirable next move with probability p = $e^{-\Delta E/KT}$
  - ΔE is the change in energy (value of the evaluation function), T is the temperature for controlling the annealing process, and K is constant analogous to Boltzmann's constant. Generally, K = 1
  - move is accepted if probability **>** a random threshold r (randomness)
- **Advantages**
  - avoids being trapped in local maxima due to the probabilistic random jump to nodes with worse evaluation function
  - high probability of convergence to global maxima due to initial randomness.

32

Simulated Annealing Parameters

33

8/14/2025

- Temperature T
  - For high T, probability → 1. All changes accepted. Random exploration is supported; moves are not trapped in local maxima.
  - For low T, probability → 0. Only transitions **with ΔE < 0 are accepted** making the search gradient-based.
- Annealing mechanisms
  - linear annealing: **$T = T_0 - \beta t$** where t is time and β is the cooling rate
  - geometric annealing: **$T(t) = T_0\alpha^t$** where α is the cooling factor around 0.7 to 0.9 and t is the time. Initially, temperature drops faster but slowly later resulting into more evaluations and better stability towards the end.

33

Simulated Annealing Algorithm

34

8/14/20

Finds minima

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for t = 1 **to** maxIteration **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 ΔE < 0 **then** current ← next
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

Bonus: handle shoulders and local maxima in this algorithm

Note: instead of s evaluating all successors a successor is picked randomly

Algorithm taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

34

Local Beam Search

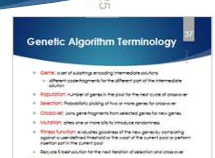
35

- Technique
  - spawn k threads starting from different initial states
  - at each iteration, successors of all k states are generated
  - if any one is a goal state, stop; else select the k best successors for all K-threads and repeat.
  - After many moves, all the threads start converging towards few better performing threads.
- **Issues**
  - An over-optimistic thread might lead to late failure
- **Solution**
  - stochastic beam search: randomly pick K successors
  - probability of picking up a successor is higher for neighbors with better values

35

Genetic Algorithms

36



- A local search algorithm based upon evolution
- Integrates randomness and focused search
- Search strategy is modeled like a gene
  - each gene is divided into multiple fragments
  - different fragments model solution of different problem subsets

- Overall approach
  - maintain a fixed pool of genes
  - probabilistically select two or more genes, cross their fragments, mutate newly formed fragments and apply fitness function. The best fragments are retained in the pool for the next cycle.

- Operations: selection, cross-over, mutation, fitness-checking

36

Genetic Algorithm Terminology

37

- Gene: a set of substrings encoding intermediate solutions
  - different code-fragments for the different part of the intermediate solution
- Population: number of genes in the pool for the next cycle of cross-over
- Selection: Probabilistic picking of two or more genes for cross-over
- Crossover: Joins gene-fragments from selected genes for new genes.
- Mutation: alters one or more bits to introduce randomness
- Fitness function: evaluates goodness of the new genes by comparing against a user-defined threshold or the worst of the current pool or perform insertion sort in the current pool
- Recycle K-best solution for the next iteration of selection and cross-over

37

Selection

38

- Expected value EV_i = *fitness-value* / sum of all *fitness-values*
- Selection strategies
  - roulette wheel (probability proportional to the EV_i)
  - ranking – descending sort by EV_i
  - tournament
    - randomly select k candidates in each tournament
    - find best (based upon EV_i) for cross-over
    - run as many tournaments as pool size
- Sampling algorithm
  - use one of the selection strategies.
  - pick some genes with lower EV_i with a lower probability

38

Crossover Example

39

- first board:(327 | 52411); second board: (247 | 48552)
- cross over of (**327** | 52411) and (247 | **48552**) → (**327** | **48552**)

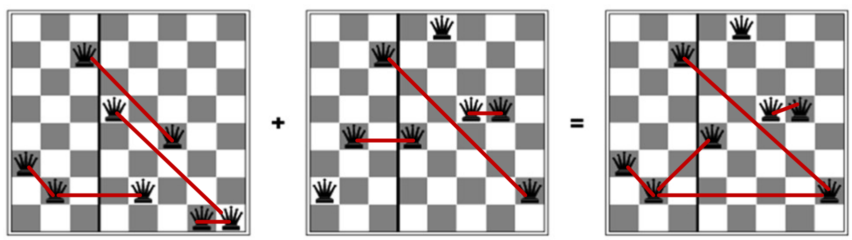


Figure taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

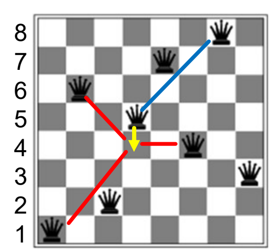
39

Mutation

40

- Alter the basic unit randomly
  - alter one digit
  - causes additional randomness
- **Example**
  - queen position by row number in each column
  - left most digit represents leftmost column
  - before mutation: 162 | **5** 7483
  - after mutation: 162 | **4** 7483

mutated digit



Before mutation:
162|**5**7483

- mutation worsens the board position from attacking pairs h
  - **initial h = 1; after mutation h = 3**

Figure taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

40

N-queens Example

- Fitness value calculated using number of attacking queens
- Probability = individual fitness value * 100 / sum of all fitness-values

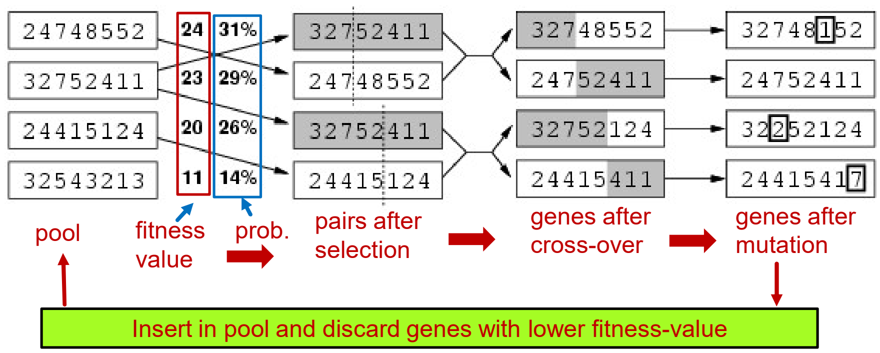


Figure taken from the book "Artificial Intelligence – A modern Approach by Russell and Norwig

41

Genetic Evolution Algorithm

Algorithm genetic evolution
Input: 1. A pool of genes P_0; 2. Fitness threshold F;
Output: Best gene g after genetic evolution;
{new-population ← empty set; i = 0; t = 0;
  while (i++ =< size(PopSize) **or** t < T^{max}) % tmax is the maximum number of iteration
  { x ← probabilistically-select(P_t); y ← probabilistically-select(P_t);
    child ← crossover(x, y);
    if goal-state(child) return child;
    else { mutated-child = mutate(child);
      fValue ← fitness-function(mutated-child);
      if (fValue > F) { P_{t+1} ← P_t ∪ {child} ; P_{t+1} = remove-worst(P_{t+1})}
      else $P_{t+1} = P_t$;
      t++; }
  }
  g = best(P_t); **return** g;
}

- Slow due to excessive cross-overs and mutations introducing randomness
- **Solution:** mix with other strategies to reduce randomness with time

42

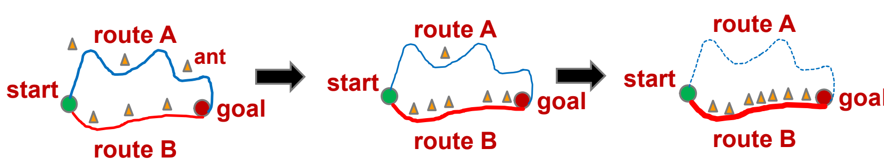
Applications

- General applications
  - final goal-state is ill defined
  - a problem can be decomposed into independent sub-problems
- Examples
  - optimal resource allocation: routing; layout planning; design
  - scheduling problems
  - self-healing systems
  - marketing, credit and insurance modeling problems, stock prediction, credit scoring, risk assessment etc.
  - automated adjustment of weights in neural networks

43

Natural Ant Mechanism

- Route B is shorter. More ants will traverse route B and lay more pheromone over it.
- The pheromone concentration on trail B will increase at a higher rate than on A.
- Pheromone on trail A will evaporate.
- Only the shortest route B will remain.



44

Ant Colony Metaheuristics

45

8/14/2025

- Population based metaheuristics
  - ants deposit pheromones when traversing after collecting food
  - pheromone dissipates slowly on shorter paths. Higher intensity means more ants and shorter path
- Probabilities are adjusted according to information on solution quality gained from previous solutions
- Has capability to dynamically search new goals due to forgetfulness caused by pheromone dissipation
- Application to a broad range of problems such as routing, traffic management, optimum postal delivery

45

Other Popular Nature-inspired Metaheuristics

46

8/14/2025

- Bee colony optimization
- Firefly optimization
- Particle swarm optimization
- Cuckoo search optimization
- Bat search optimization
- Golden Eagle Optimization

46

Ant Colony Optimization

- Probability of ants to go from node i to node j is given by
  - P_{ij} is the network routing probability
  - α and β are influence parameters
  - $\emptyset_{ij}$ is the pheromone concentration
  - δ_{ij} is the desirability of a path is inversely proportional to the length

$$P_{ij} = \frac{\phi_{ij}^{\alpha}\ \delta_{ij}^{\beta}}{\sum_{i,\,j\,=\,1}^{i,\,j\,=\,n} \phi_{ij}^{\alpha}\ \delta_{ij}^{\beta}}$$

- Pheromone evaporation /deposition
  - $\emptyset(t) = \emptyset_0 e^{-\gamma t}$ **where γ is the evaporation rate of the pheromone**
  - if the evaporation rate is small the equation reduces to $\emptyset_0(1 - \gamma t)$ by using Binomial series expansion and ignoring high power terms
  - $\emptyset_{ij}(t+1) = (1 - \gamma)\emptyset_{ij}(t) + \delta\ \emptyset_{ij}(t)$ **where δ is the deposition rate**
  - system avoids being trapped in local maxima as the objective function (pheromone value) is dynamic.

47