Artificial Intelligence (CS 3011)

CHAPTER 4: **Beyond classical search**

Sourav Kumar Giri
Asst. Professor

School of Computer Engineering
KIIT Deemed to be University

Chapter Outline

- Local search algorithms and optimization problem
- Hill Climbing
- Simulated Annealing
- Local Beam Search and
- Genetic Algorithm

Local Search Algorithms and Optimization Problems

- **Systematic Search problems:** The search algorithms that we have seen so far are designed to explore search spaces systematically. This is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path. When a goal is found, the path to that goal also constitutes a solution to the problem (a sequence of actions from initial state to goal state).
- **Local Search problems:** These algorithms perform purely local search in the state space, evaluating and modifying one or more current states rather than systematically exploring paths from an initial state. These algorithms are suitable for problems in which all that matters is the solution state, not the path cost to reach it. The path to the goal is irrelevant.
- ***For example,*** in the 8-queens problem what matters is the final configuration of queens, not the order in which they are added.

Local Search Algorithms and Optimization Problems

- The same general property holds for many important applications as indicated below:
  - Integrated-circuit design
  - Factory-floor layout
  - Job-shop scheduling
  - Automatic programming
  - Telecommunications network optimization
  - Vehicle routing
  - Portfolio management
- The family of local search algorithms includes: **Hill Climbing**, **Simulated Annealing** (*inspired by statistical physics*), **Local Beam search** and **Genetic Algorithms** (*inspired by evolutionary biology*).

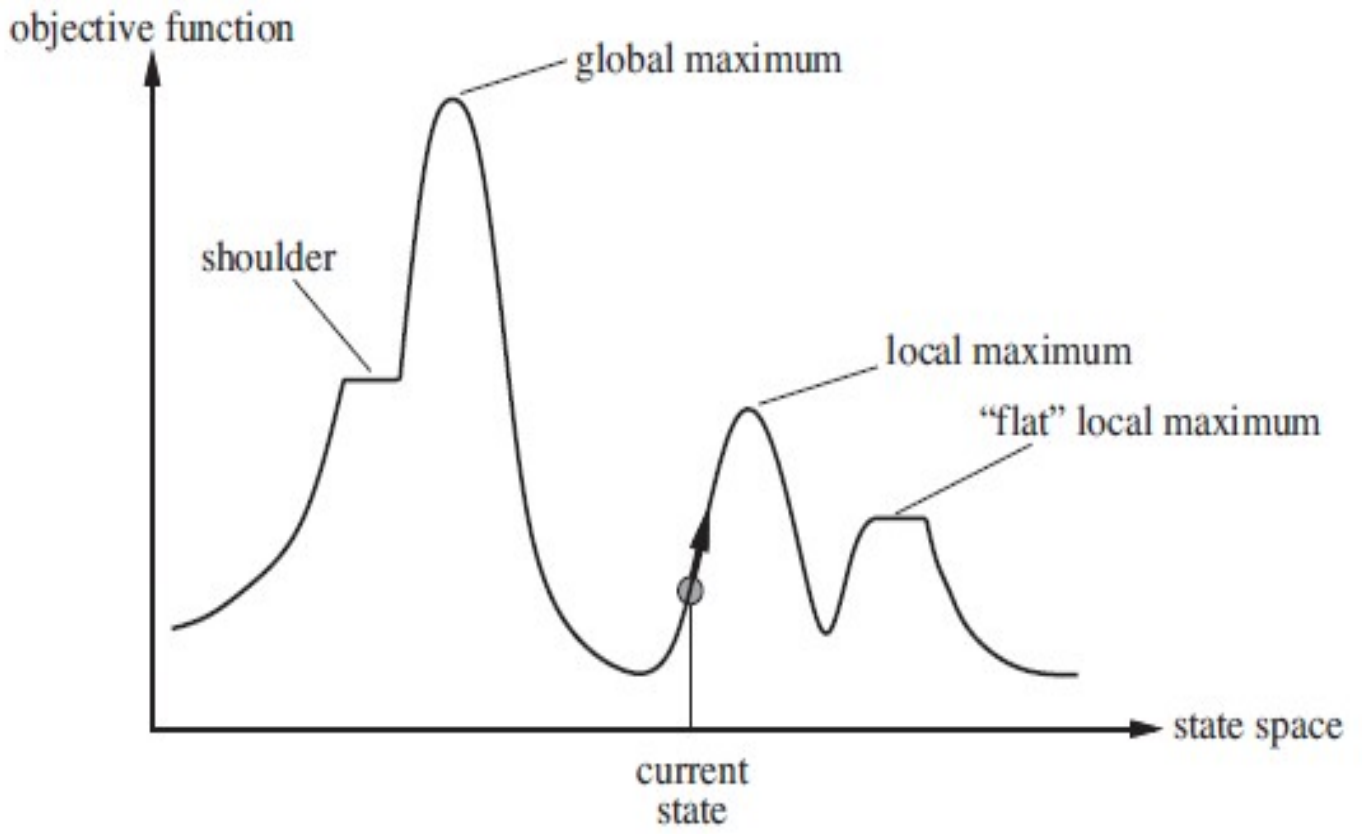
Local Search Algorithms and Optimization Problems

- Local search algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbors of that node. Typically, the paths followed by the search are not retained. Although local search algorithms are not systematic, they have two key advantages:
  - they use very **little memory**—usually a constant amount;
  - they can often find **reasonable solutions** in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

Local Search Algorithms and Optimization Problems

- In addition to finding goals, local search algorithms are useful for solving pure optimization problems, in which the aim is to find the best state according to an objective function.
- Many optimization problems do not fit the "standard" search model (both uninformed and informed).
- For example, nature provides an objective function—reproductive fitness—that Darwinian evolution could be seen as attempting to optimize, but there is no "goal test" and no "path cost" for this problem.
- To understand local search, we find it useful to consider the state-space landscape. A landscape has both "location" (defined by the state) and "elevation" (defined by the value of the heuristic cost function or objective function).

Local Search Algorithms and Optimization Problems



Local Search Algorithms and Optimization Problems

- If elevation corresponds to cost, then the aim is to find the lowest valley—a **global minimum**;
- If elevation corresponds to an objective function, then the aim is to find the highest peak—a **global maximum**.
- We can convert from one to the other just by inserting a minus sign.
- Local search algorithms explore this landscape. A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a **global minimum/maximum**.

Hill climbing

- Hill climbing (basic or steepest-ascent) search algorithm is a local search algorithm.
- It is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.
- This algorithm is simply a loop that continuously moves in the direction of increasing value i.e. *uphill*. It terminates when it reaches a "peak" where *no neighbor has a higher value*.
- The algorithm doesn't maintain a search tree, so the current node data structure only records the state and its objective function value. Hill climbing doesn't look ahead beyond the immediate neighbors of the current state.
- This resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia.

Basic Hill climbing algorithm

1) Evaluate the initial state (IS). If it is the goal state (GS) , then return it and quit. Else consider IS as the current state (CS) and proceed.

2) Loop until a solution is found or there are no new operator (OP) to be applied to the CS.

a) Select an OP that has not yet been applied to the CS and apply it to produce a new state (NS).

b) Evaluate the NS:

- If NS is a GS , then return it and quit.
- If it is not a GS but better than the CS, then consider it as the current state(i,e. CS ←NS) and proceed.
- If NS is not better than CS then continue in the loop by selecting the next appropriate OP for CS.

Steepest ascent hill climbing algorithm

It considers all the moves from the CS and selects the best one as the next state. It is also called gradient search.

Algorithm:

1) Evaluate the initial state (IS). If it is the goal state (GS) , then return it and quit. Else consider IS as the current state (CS) and proceed.
2) Loop until a solution is found or until a complete iteration produces no change to the CS:
   a) Let successor (SUCC) be a state such that any NS that can be generated from CS is better than SUCC. [i.e setting SUCC to a minimum value at the beginning of an iteration or set CS as SUCC]

Steepest ascent hill climbing algorithm

b) For each operator OP that applies to the CS do:

i. Apply OP to CS and generate a NS.

ii. Evaluate the NS. If it is a GS then return it and quit. If not , compare it with SUCC. If NS is better than SUCC, then set SUCC to NS; else leave SUCC unchanged.

c) If the SUCC is better than CS, then set CS to SUCC [i.e move to the next best state]

```
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

Both basic and steepest-ascent hill climbing may fail to find a solution. Either algorithm may terminate not by finding a goal state but by getting to a state from which no better states can be generated.

This will happen if the program has reached either a **local maximum** or a **plateau** or a **ridge**.

- A **local maximum** is a state that is better than all its neighbors but is not better than some other states farther away.

  Solution of local maxima:-

  - Move in some arbitrary direction
  - Back track to an ancestor and try some other alternatives.

Hill climbing search

- A **ridge** is an area in the search space which is higher than its surroundings but itself has slopes. It is not possible to traverse a ridge by a single move i. e. no such operator is available.

  Solution of ridges:-

  - Apply several operators before doing the evaluation

- A **plateau** is a flat area in the search space in which all the neighboring states have the same heuristic function value. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau.

  Solution of plateau

  - Expand few generation ahead to move to a different section of the search space.

More variants of hill climbing search

- **Stochastic hill climbing:** It chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move. This usually converges more slowly than steepest ascent, but in some state landscapes, it finds better solutions.
- **First-choice hill climbing**: It implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state. This is a good strategy when a state has many (e.g., thousands) of successors.

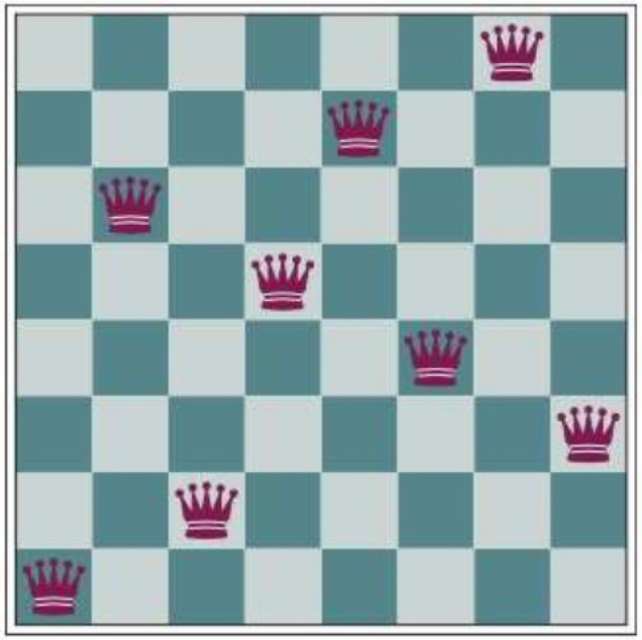
(The hill-climbing algorithms described so far are incomplete—they often fail to find a goal when one exists because they can get stuck on local maxima.)

More variants of hill climbing search

- **Random-restart hill climbing** adopts the well-known adage, "*If at first you don't succeed, try, try again.*" It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

The success of hill climbing depends very much on the shape of the state-space landscape: if there are few local maxima and plateaus, random-restart hill climbing will find a good solution very quickly.

8-queens problem



(a)

(b)

Simulated annealing search

- A hill-climbing algorithm that never makes “downhill” moves toward states with lower value (or higher cost) is guaranteed to be incomplete, because it can get stuck on a local maximum.
- In contrast, a purely random walk—that is, moving to a successor chosen uniformly at random from the set of successors—is complete but extremely inefficient.
- Therefore, it seems reasonable to try to combine hill climbing with a random walk in some way that yields both efficiency and completeness. Simulated annealing is such an algorithm.
- Simulated annealing was first used extensively to solve VLSI layout problems in the early 1980s. It has been applied widely to factory scheduling and other large-scale optimization tasks.

Simulated annealing search

- In metallurgy, annealing is the process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low energy crystalline state.
- To explain simulated annealing, we switch our point of view from hill climbing to gradient descent (i.e., minimizing cost) and imagine the task of getting a ping-pong ball into the deepest crevice in a bumpy surface.
- If we just let the ball roll, it will come to rest at a local minimum. If we shake the surface, we can bounce the ball out of the local minimum. The trick is to shake just hard enough to bounce the ball out of local minima but not hard enough to dislodge it from the global minimum.
- The simulated-annealing solution is to start by shaking hard (i.e., at a high temperature) and then gradually reduce the intensity of the shaking (i.e., lower the temperature).

Simulated annealing search algorithm

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The schedule input determines the value of the temperature T as a function of time.

Local beam search

- Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations.
- The local beam search algorithm, which is a path-based algorithm keeps track of k states rather than just one. It begins with k randomly generated states.
- At each step, all the successors of all k states are generated. If any one is a goal, the algorithm halts. Otherwise, it selects the k best successors from the complete list and repeats.
- At first sight, a local beam search with k states might seem to be nothing more than running k random restarts in parallel instead of in sequence. In fact, the two algorithms are quite different.
- In a random-restart search, each search process runs independently of the others.

Local beam search

- In a local beam search, useful information is passed among the parallel search threads. In effect, the states that generate the best successors say to the others, “Come over here, the grass is greener!” The algorithm quickly abandons unfruitful searches and moves its resources to where the most progress is being made.
- In its simplest form, local beam search can suffer from a lack of diversity among the k states—they can quickly become concentrated in a small region of the state space, making the search little more than an expensive version of hill climbing.
- A variant called stochastic beam search, analogous to stochastic hill climbing, helps alleviate this problem. Instead of choosing the best k from the pool of candidate successors, stochastic beam search chooses k successors at random, with the probability of choosing a given successor being an increasing function of its value.

Local beam search

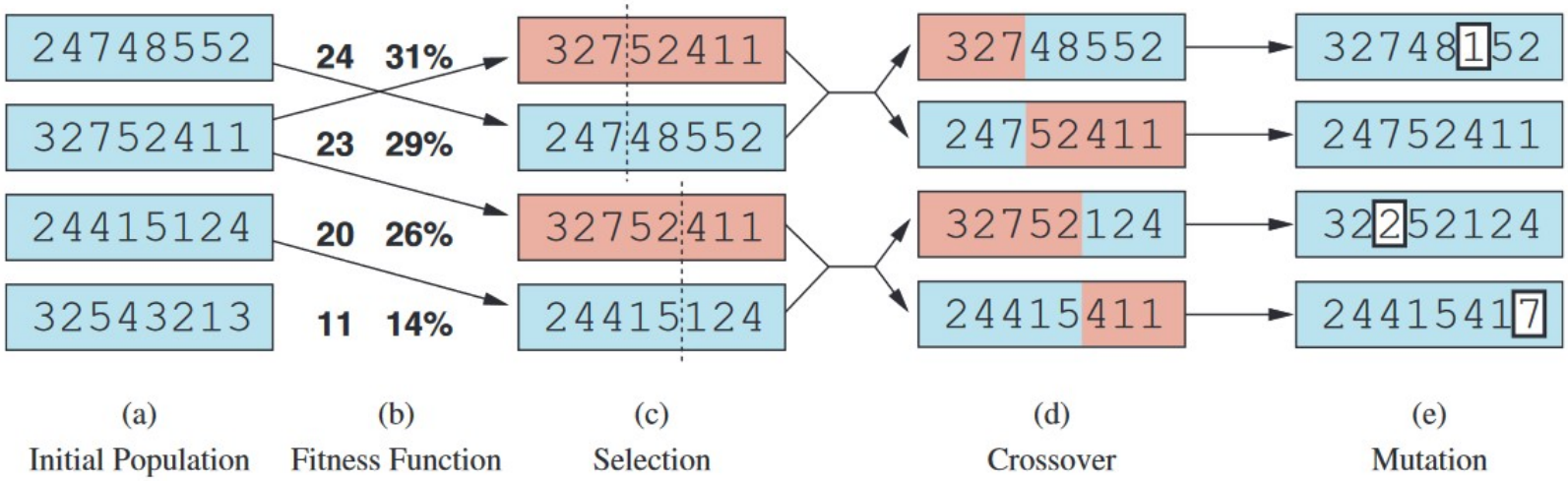
- Stochastic beam search bears some resemblance to the process of natural selection, whereby the “successors” (offspring) of a “state” (organism) populate the next generation according to its “value” (fitness).

Genetic algorithm

- A genetic algorithm (or GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state.
- The analogy to natural selection is the same as in stochastic beam search, except that now we are dealing with mimicking the natural real life reproduction rather than asexual reproduction as in stochastic beam search. (Later normally does not happen in real life).
- Like beam searches, GAs begin with a set of k randomly generated states, called the population. Each state, or individual, is represented as a string over a finite alphabet—most commonly, a string of 0s and 1s.

Genetic algorithm

- For example, an 8-queens state must specify the positions of 8 queens, each in a column of 8 squares, and so requires 8 × log28 bits. Alternatively, the state could be represented as 8 digits, each in the range from 1 to 8. These two encodings behave differently.



Genetic algorithm

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```