Local Search

COURSE: CS60045

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Lecture Objectives

- Learning the concept of problem solving using local improvements
- Learning the concept of getting stuck in local optima
- Learning the ways to get out of local optima
- Local search algorithms

REFERENCE

Artificial Intelligence – A Modern Approach, Stuart J Russell and Peter Norvig , Pearson Education India

Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
  - Local search: widely used for *very big* problems
  - Returns good but *not optimal* solutions in general

- The state space consists of "complete" configurations
  - For example, every permutation of the set of cities is a configuration for the traveling salesperson problem
- The goal is to find a “close to optimal” configuration satisfying constraints
  - Examples: n-Queens, VLSI layout, exam time table

- Local search algorithms
  - Keep a single "current" state, or small set of states
  - Iteratively try to improve it / them
  - Very memory efficient since only a few states are stored

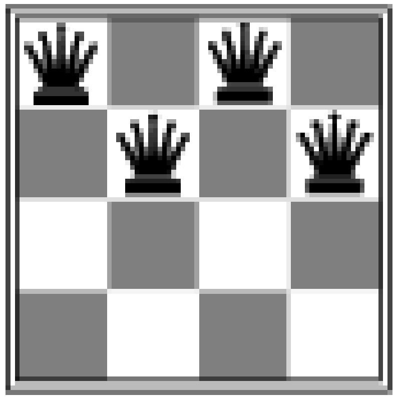
Example: *4*-queens

Goal: Put *4* queens on an *4* × *4* board with no two queens on the same row, column, or diagonal

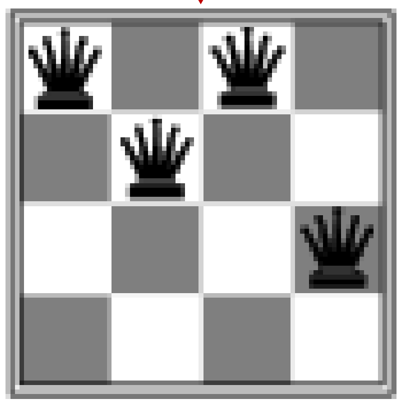
State space: All configurations with the queens in distinct columns

State transition: Move a queen from its present place to some other square in the same column

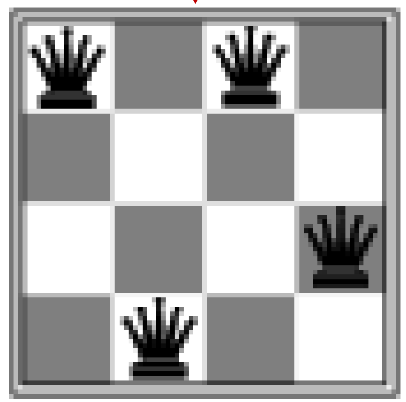
Local Search: Start with a configuration and repeatedly use the moves to reach the goal



Move queen in Column 4



Move queen in Column 2



The last configuration has fewer conflicts than the first, but is still not a solution

Hill-climbing: *A greedy approach*

THE IDEA: *Make a move only if the neighboring configuration is better than the present one*

```
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
```

The dual of Hill Climbing is Gradient Descent. Hill climbing is for maximizing, Gradient Descent is for minimizing

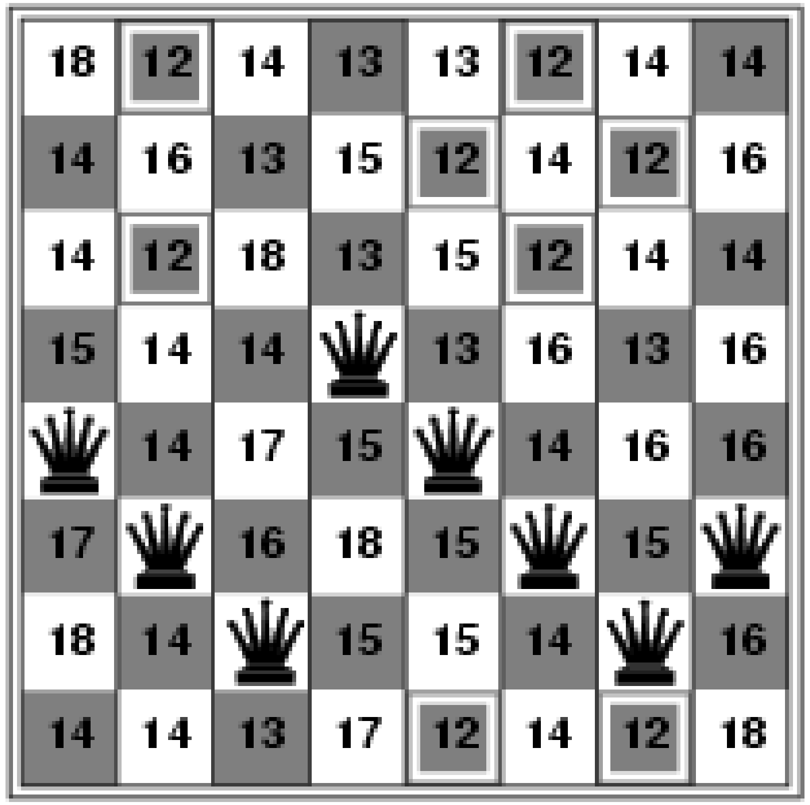
Source: Artificial Intelligence – A Modern Approach, Peter Norvig and Stuart Russell, Prentice Hall

Gradient Descent in 8-queens

Value[state] = The numbers pairs of queens that are attacking each other, either directly or indirectly.

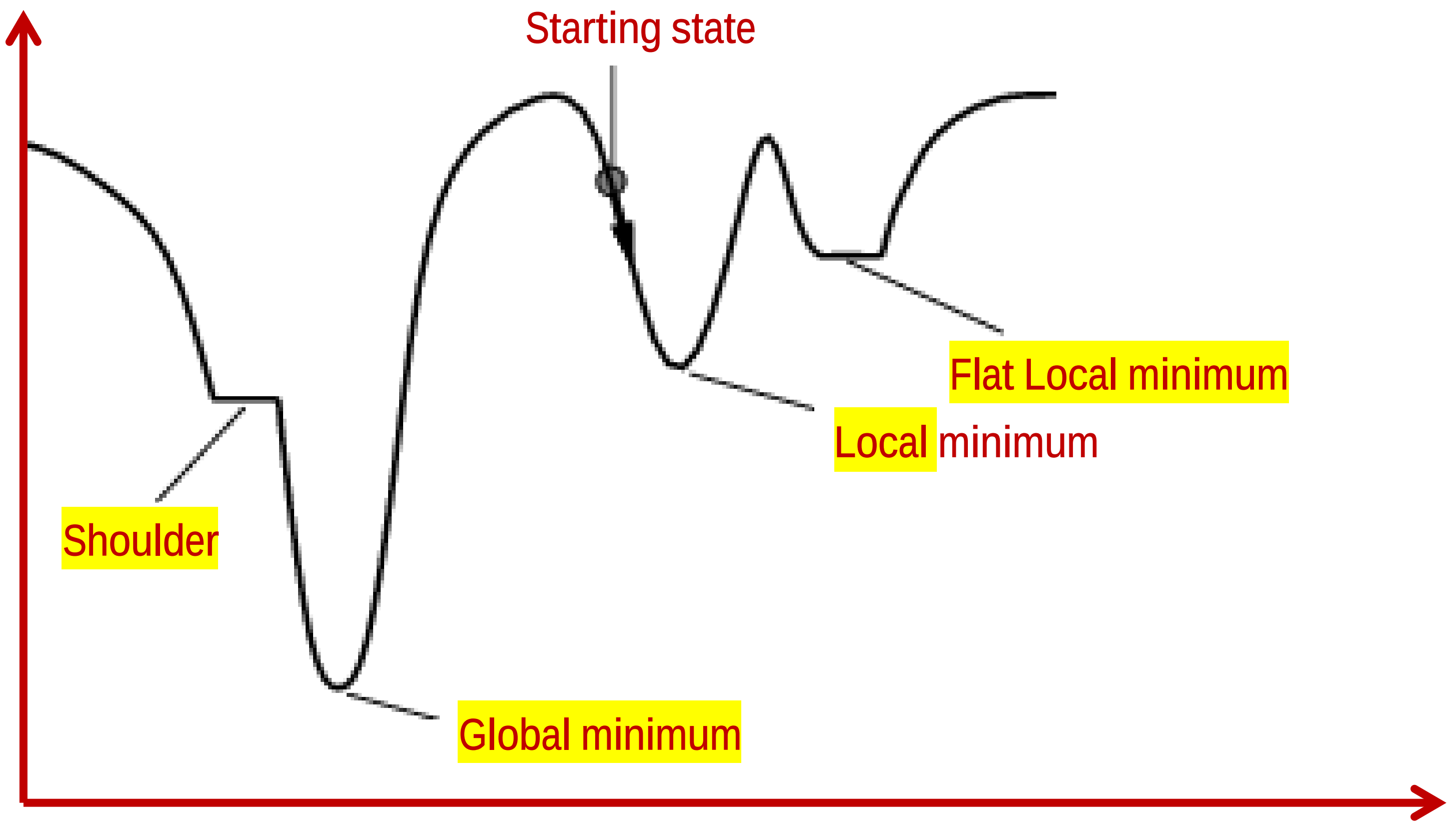
Value[state] = 17 for the state shown in the Fig.

- The number in each square is the value of state if we move the queen in the same column to that square.
- Therefore the best greedy move is to move a queen to a square labeled with 12.
    - There are many such moves. We choose one of them at random.



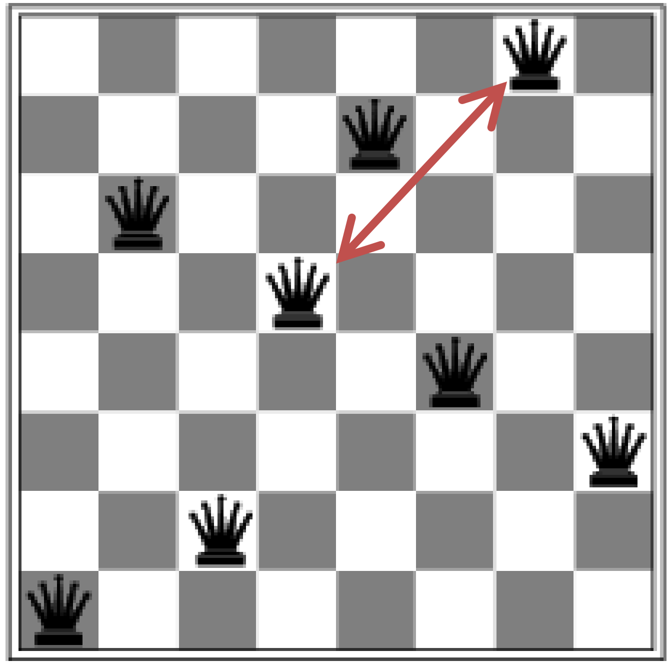
Gradient descent can get stuck in local minima

- Each neighbor of a minimum is inferior with respect to the minimum
- No move in a minimum takes us to a better state than the present state



Local minimum in 8-queens

- A local minimum with only one conflict
- All one-step neighbors have more than one conflict



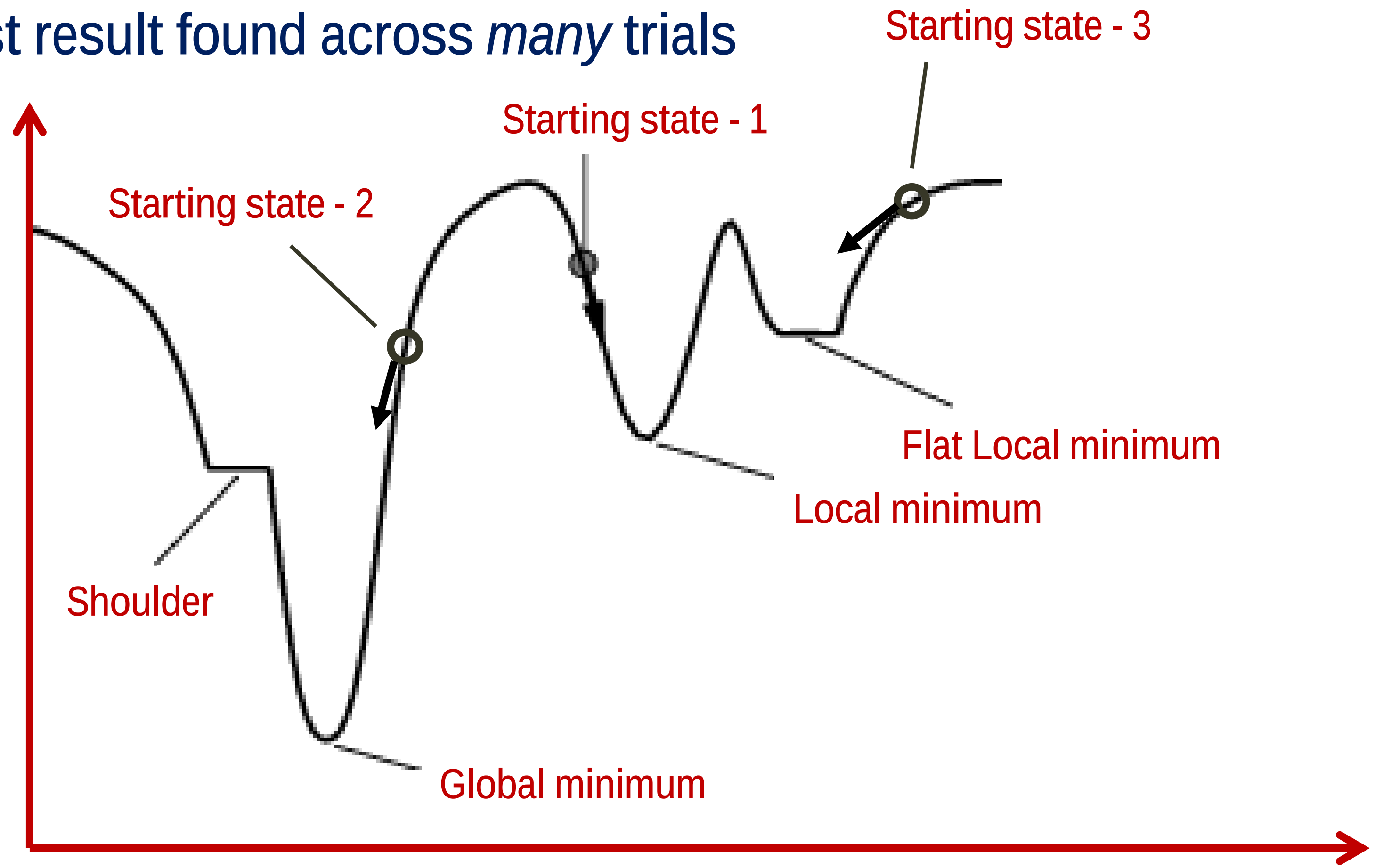
How to get out of local minima?

Idea-1: Gradient Descent with Random Restart

Using many random restarts improves our chances

Restart a random initial state, *many times*

- Report the best result found across *many* trials



Idea-2: Allow moves to inferior neighbors

To get out of a local minimum, we must allow moves to inferior neighbors

However, we must ensure that we do not oscillate among a set of states

IDEAS

Simulated Annealing: Allow moves to inferior neighbors with a probability that is regulated over time. We discuss this in more details later

Tabu Search: Add recently visited states to a tabu-list.

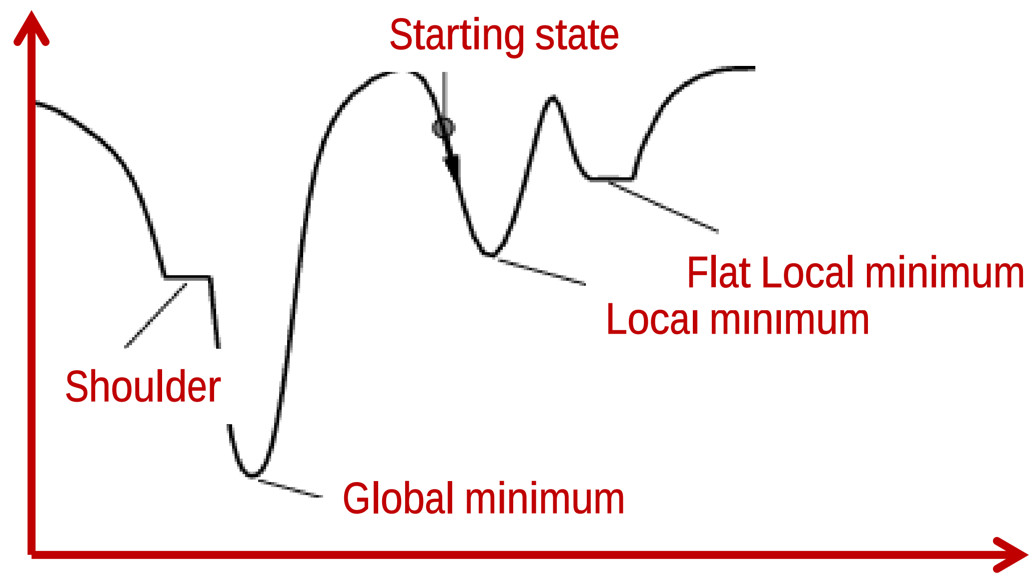
- These states are temporarily excluded from being visited again
- Forces solver away from explored regions
- Avoid getting stuck in local minima (in principle)

Simulated annealing search

IDEA: Escape local maxima by allowing some "bad" moves but gradually decrease their probability

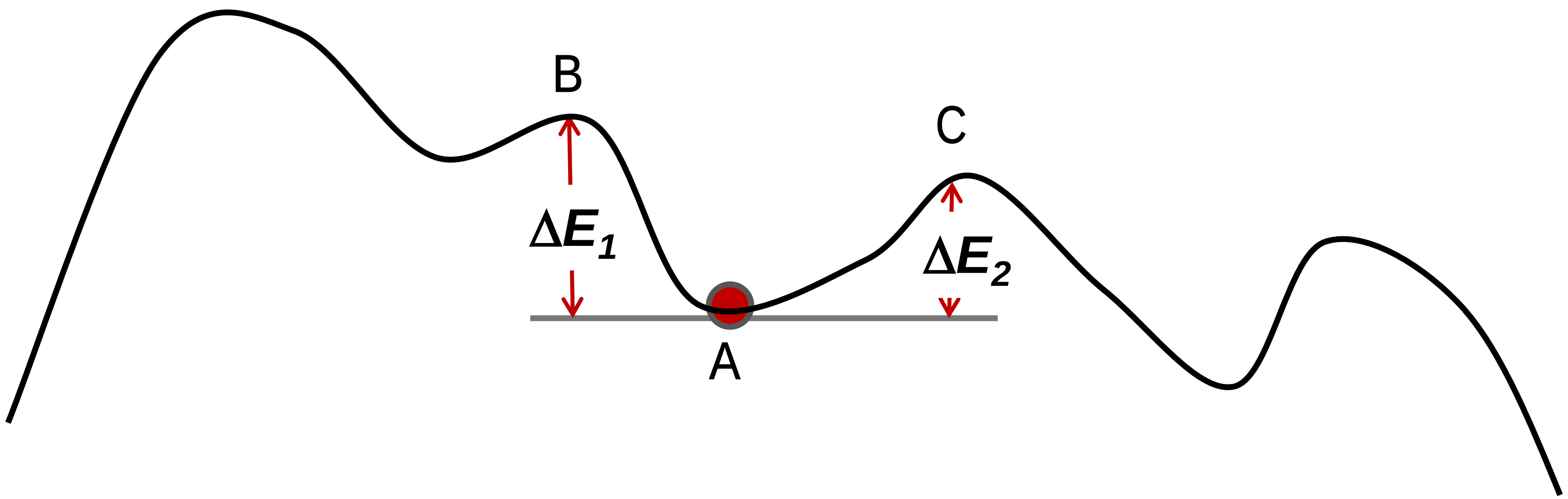
- The probability is controlled by a parameter called *temperature*
- *Higher temperatures allow more bad moves than lower temperatures*
- Annealing: Lowering the temperature gradually   Quenching: Lowering the temperature rapidly

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



How simulated annealing works

Probability of making a bad move = $\boldsymbol{e}^{-\Delta E/T} = \dfrac{\mathbf{1}}{\boldsymbol{e}^{\Delta E/T}}$



Since $\Delta E_1 > \Delta E_2$ moving from A to C is exponentially more probable than moving from A to B

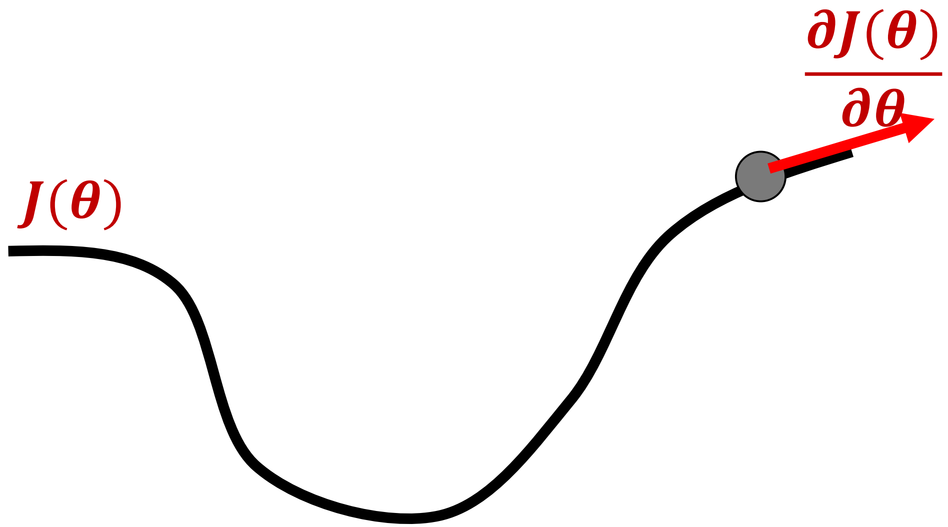
Properties of Simulated Annealing

- It can be proven that:
  - If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
  - Since this can take a long time, we typically use a temperature schedule which fits our time budget and settle for the sub-optimal solution

- Simulated annealing works very well in practice
- Widely used in VLSI layout, airline scheduling, etc.

Hill Climbing in Continuous Multi-variate State Spaces

Denote "state" as **μ**; cost as J(**μ**)



- Choose a direction in which J(μ) is decreasing
- Derivative: $\frac{\partial J(\theta)}{\partial \theta}$
  - Positive derivative means increasing
  - Negative derivative means decreasing
- Move: A short uphill step in the chosen direction

Local Search with Multiple Present States

Instead of working on only one configuration at any time, we could work on multiple promising configurations concurrently

LOCAL BEAM SEARCH

Maintain *k* states rather than just one. Begin with *k* randomly generated states

In each iteration, generate all the successors of all *k* states

Stop if a goal state is found; otherwise Select the *k* best successors from the complete list and repeat

GENETIC ALGORITHMS

States are strings over a finite alphabet (genes). Begin with *k* randomly generated states (population).

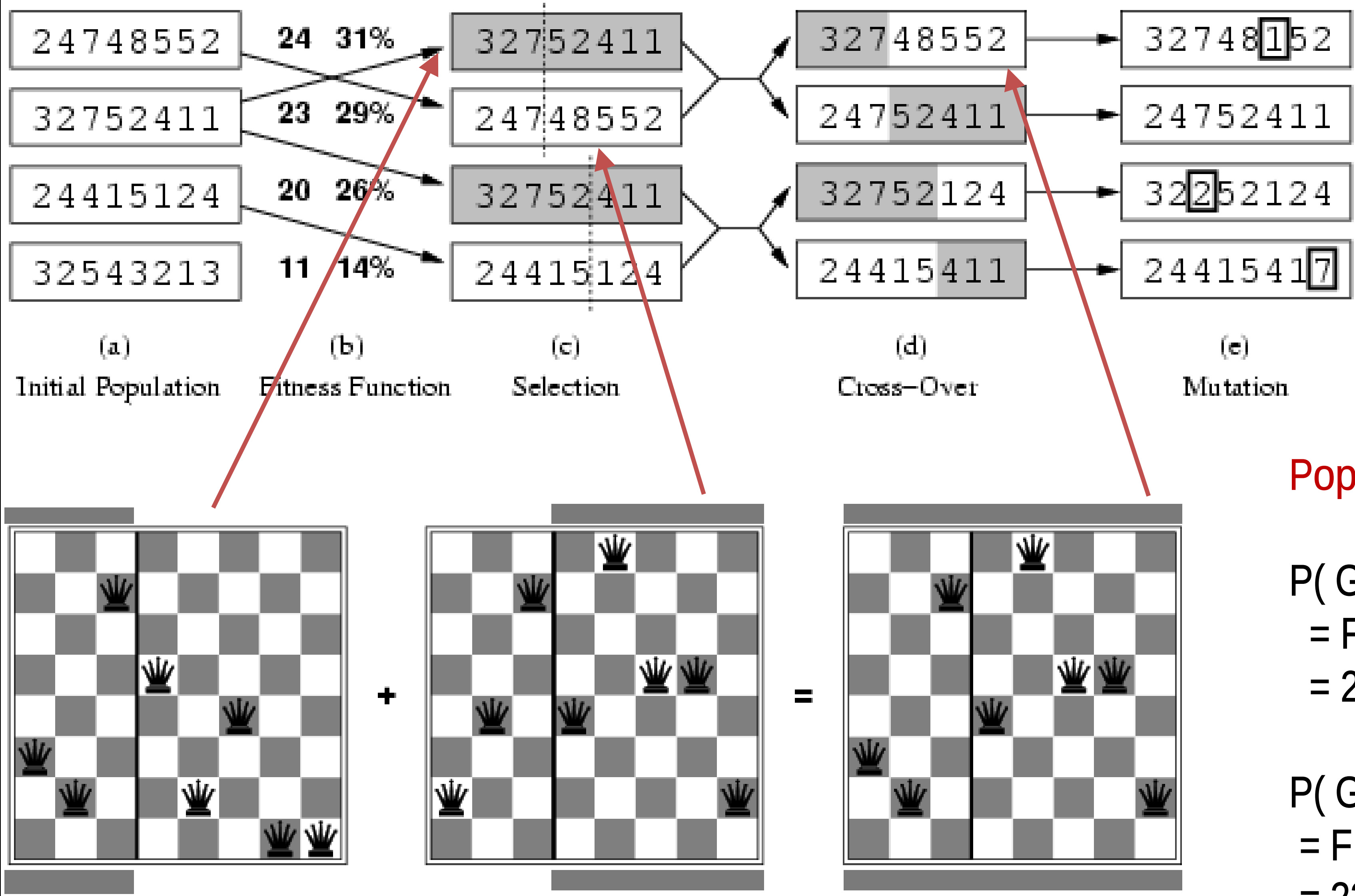
Select individuals for next generation based on a fitness function.

Two types of operators for creating the next states:

- Crossover: Fit parents to yield next generation (offspring)
- Mutation: Mutate a parent to create an offspring randomly with some low probability

Genetic Algorithm for 8 Queens

Fitness function: # non-attacking pairs
(min = 0, max = 8×7 / 2 = 28)



Population fitness = 24+23+20+11 = 78

P(Gene-1 is chosen)
= Fitness of Gene-1 / Population fitness
= 24 / 78 = 31%

P(Gene-2 is chosen)
= Fitness of Gene-2 / Population fitness
= 23 / 78 = 29%

Concluding Remarks

- Memory usage is one of the determining factors for choosing a search algorithm
  - For large state spaces, local search is an attractive practical option
- For local search:
  - It is important to understand the tradeoff between time and solution quality
  - It is important to understand the shape of the state space to decide things like temperature schedule in simulated annealing, durations of locking of moves in tabu search, and number of random restarts in gradient descent
- Local search is not a push-button solution