

# LOCAL SEARCH ALGORITHMS

## CHAPTER 4, SECTIONS 3–4

## Iterative improvement algorithms

In many optimization problems, **path** is irrelevant;  
the goal state itself is the solution

Then state space = set of “complete” configurations;  
find **optimal** configuration, e.g., TSP  
or, find configuration satisfying constraints, e.g., timetable

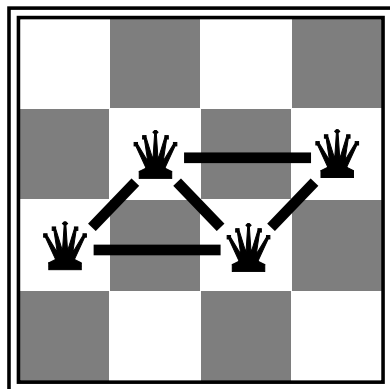
In such cases, can use **iterative improvement** algorithms;  
keep a single “current” state, try to improve it

Constant space, suitable for online as well as offline search

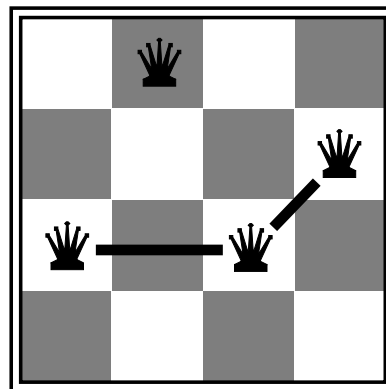
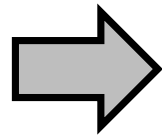
## Example: $n$ -queens

Put  $n$  queens on an  $n \times n$  board with no two queens on the same row, column, or diagonal

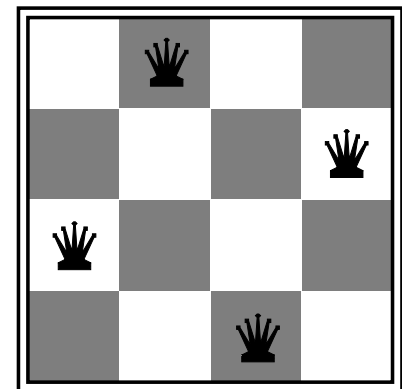
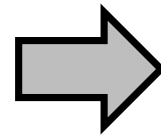
Move a queen to reduce number of conflicts



$h = 5$



$h = 2$



$h = 0$

Almost always solves  $n$ -queens problems almost instantaneously for very large  $n$ , e.g.,  $n = 1\text{million}$

## Hill-climbing (or gradient ascent/descent)

“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
  end
```

# Simulated annealing

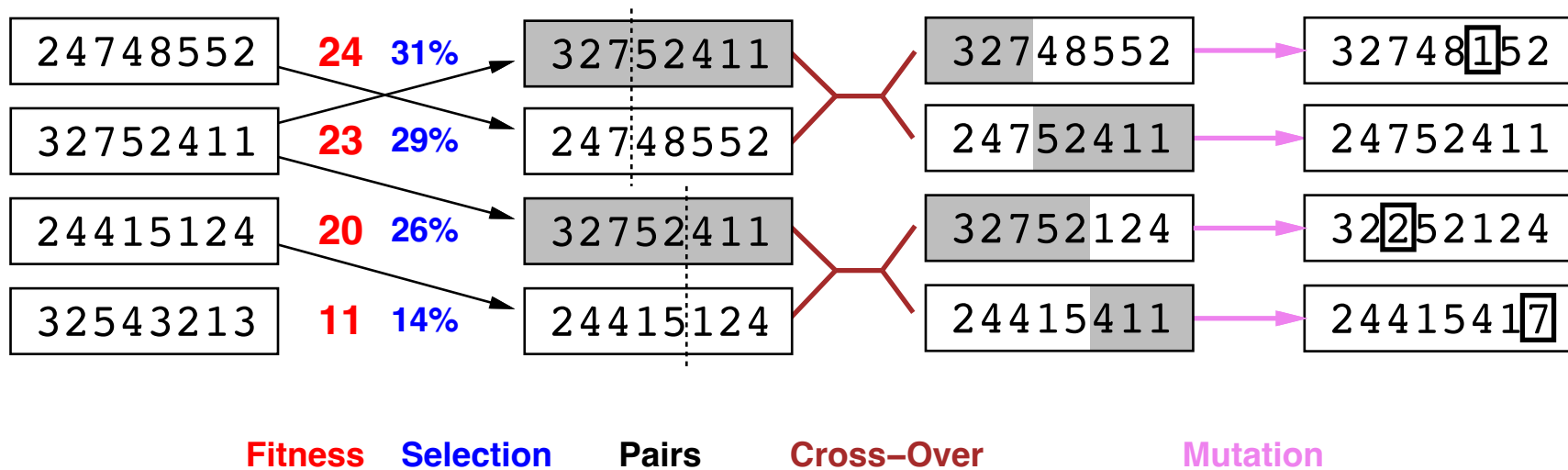
Idea: escape local maxima by allowing some “bad” moves  
but gradually decrease their size and frequency

```
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 ← 1 to  $\infty$  do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
     $\Delta E$  ← VALUE[next] − VALUE[current]
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{\Delta E/T}$ 
```

# Genetic algorithms

= stochastic local beam search + generate successors from **pairs** of states



# Genetic algorithms

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights  $\leftarrow$  WEIGHTED-BY(population, fitness)
    population2  $\leftarrow$  empty list
    for  $i = 1$  to SIZE(population) do
      parent1, parent2  $\leftarrow$  WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child  $\leftarrow$  REPRODUCE(parent1, parent2)
      if (small random probability) then child  $\leftarrow$  MUTATE(child)
      add child to population2
    population  $\leftarrow$  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
   $n \leftarrow$  LENGTH(parent1)
   $c \leftarrow$  random number from 1 to  $n$ 
  return APPEND(SUBSTRING(parent1, 1,  $c$ ), SUBSTRING(parent2,  $c + 1$ ,  $n$ ))
```