

COMP338: ARTIFICIAL INTELLIGENCE

Solving Problems by Searching –
Local Search

Dr. Radi Jarrar



LOCAL SEARCH

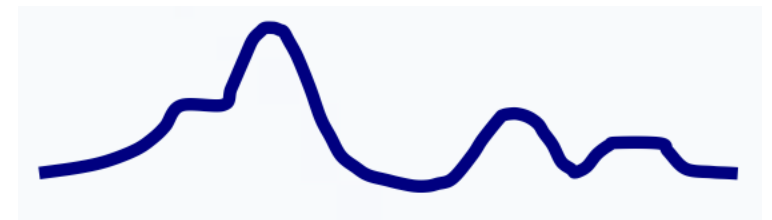
Local Search



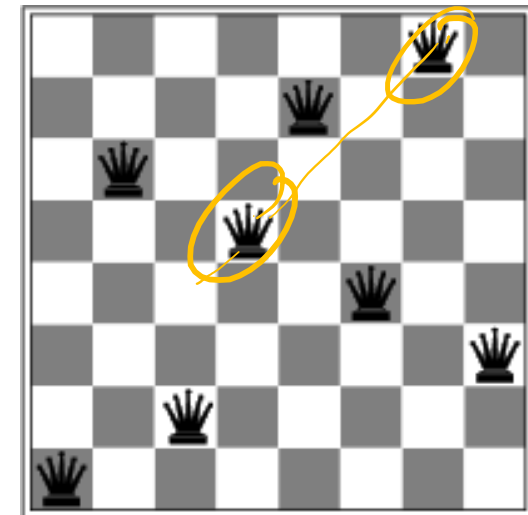
- If the path to the goal does not matter, another class of algorithms that do not worry about the path is considered min max
- Here, in many optimization problems, the path to the goal is irrelevant; *the goal state itself is the solution.*
- Local Search algorithms operate using a single current node (rather than multiple paths) and move only to neighbours of that node
- The paths followed by the search are not retained

Local Search

- Examples:
- Minimisation or Maximisation problems
 - Minimise the cost, as in cost functions
- Reducing conflicts, as in n-queens



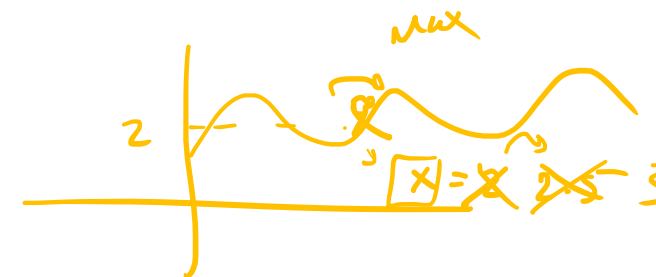
Vert: \rightarrow
Horizontal
Diagonal \leftarrow



Local Search: Gradient Descent ^{start} X ^{Goal} X

- In most problems, once a solution is found, the path to the goal constitutes a solution to the problem
- In many problems, the path to the goal is irrelevant
 - For example, in the 8-queens problem: what matters is the final configurations of queens not the order in which they are added
- This also holds in many applications such as integrated-circuit design, factory-floor layout, telecommunication network optimisation, vehicle routing, ...

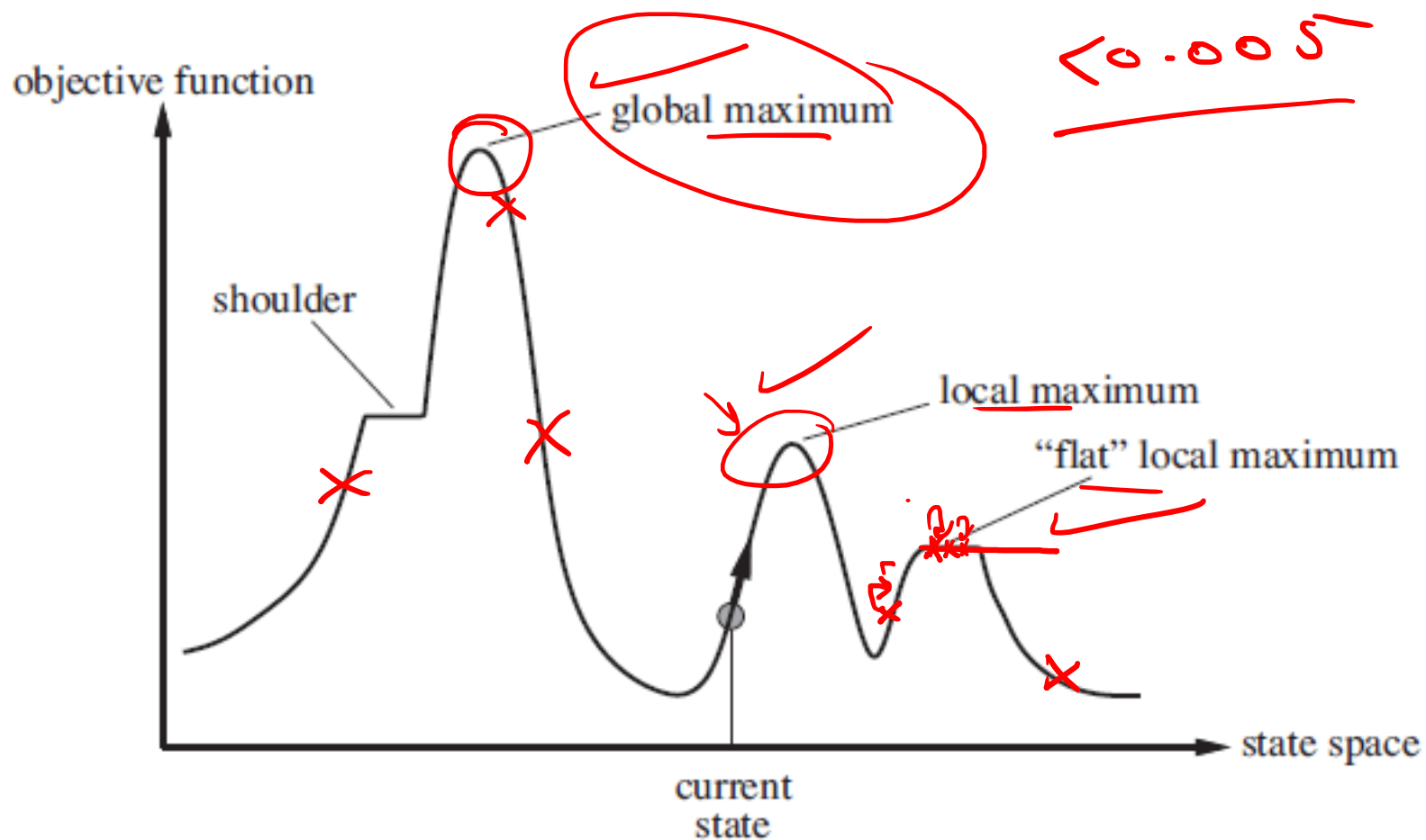
Local Search



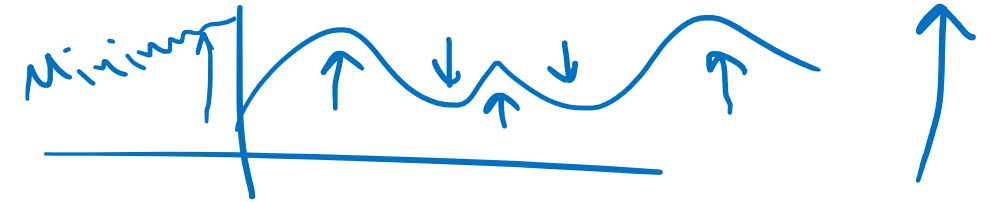
- Local search has the following main advantages
 - They use very little memory (usually a constant amount)
 - They can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable
 - Another advantage of using local search is their ability of solving optimisation problems
 - The goal of optimisation problems is to find the best state according to an objective function

minimise the error
maximise the profit

Local Search-State Space Landscape

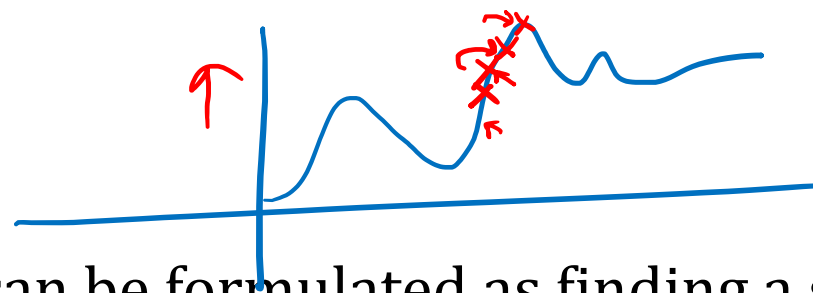


State-Space Landscape



- A landscape has both location (defined by the state) and elevation (defined by the value of the heuristic cost function or objective function)
- 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**
- A **complete** local search algorithm always finds a goal if one exists; an **optimal** algorithm always finds a global minimum/maximum

Local Search



- Local search can be used on problems that can be formulated as finding a solution **maximizing a criterion** among a number of candidate solutions.
- Local search algorithms **move from solution to solution** in the space of candidate solutions (the *search space*) **until a solution deemed optimal is found or a time bound is elapsed.** *will not converge [will fail to find true min/max]*
- For example, the travelling salesman problem, in which a solution is a cycle containing all nodes of the graph and the target is to **minimize the total length of the cycle.** *Problem* i.e. a solution can be a cycle and the criterion to maximize is a combination of the number of nodes and the length of the cycle.
- A local search algorithm **starts from a candidate solution and then iteratively moves** to a neighbor solution. *selected at Random*

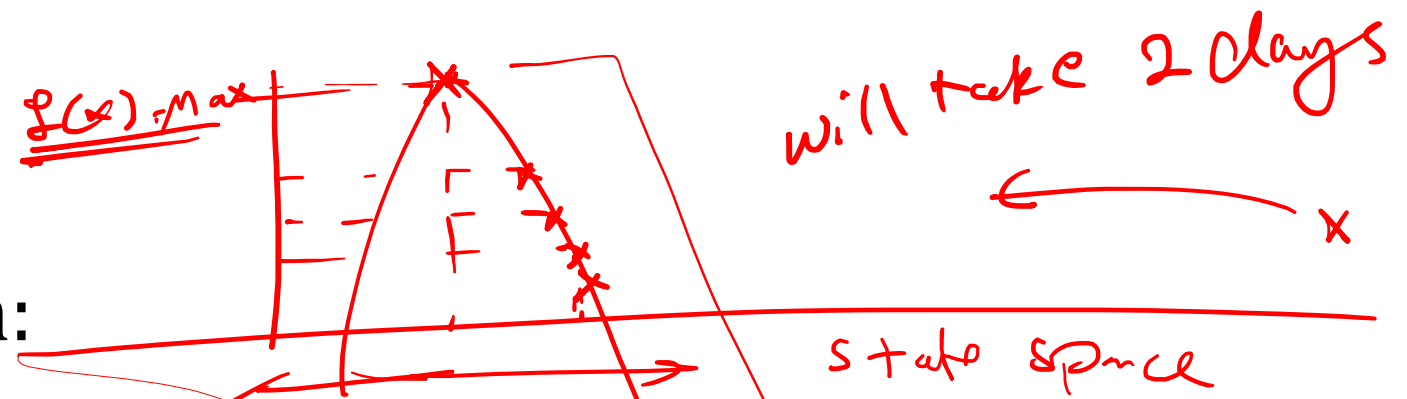
Local Search

- The idea of local search:

- Select a random initial state (generate a random initial state)
- Make local modifications to improve the current state (i.e., evaluate the current state) and move to other states (only neighbors of that node/state) that enhance over the current state

- Repeat until a goal state is found or Reaching max number of iterations

Search space is infinite



Local Search



② Better than its Neighbors
 → Local Min / Local Max

- Terminate on a time bound or if the situation is not improved after number of steps.
- **Local search algorithms are typically incomplete algorithms**, as the search may stop even if the best solution found by the algorithm is not optimal.



Local Search - Advantages

- No need to maintain a search tree.
- Use very little memory.
- Can often find good enough solutions in continuous or large state spaces.

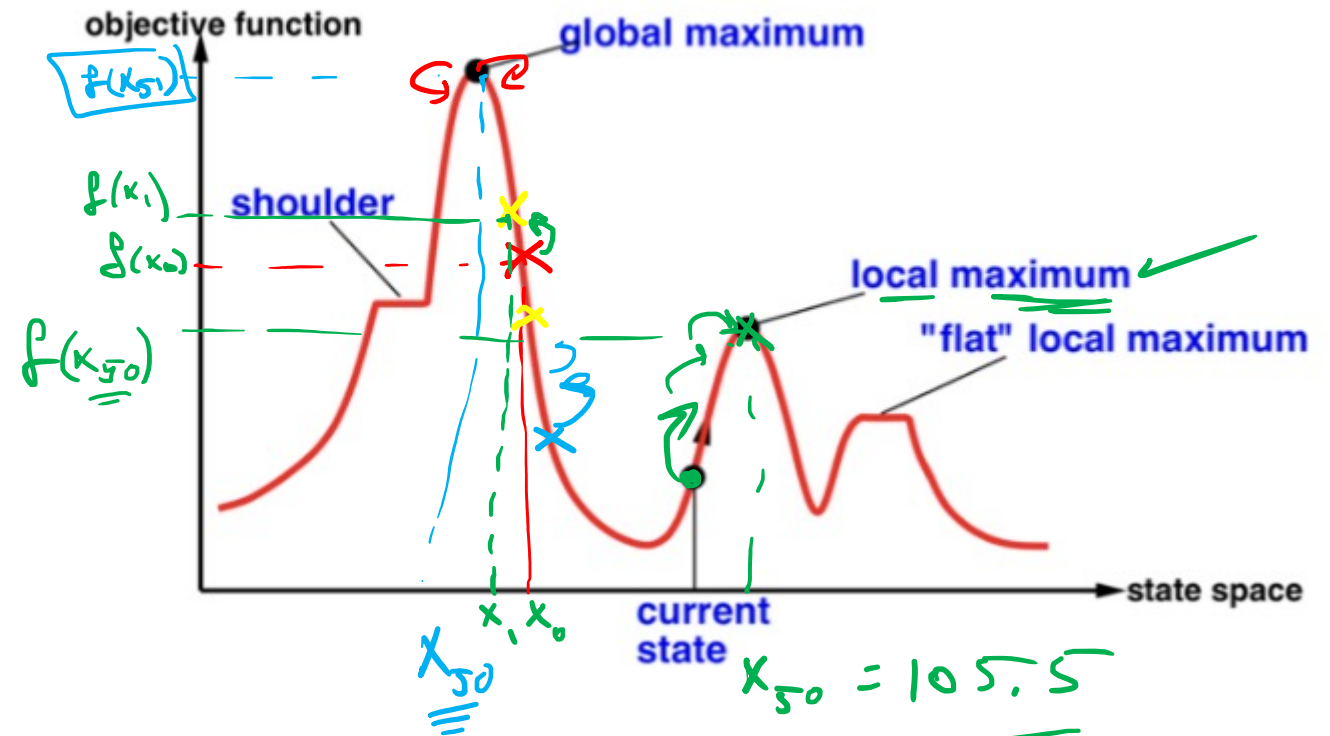
Local Search Algorithms

- Hill climbing (steepest ascent/descent).
- Simulated Annealing: inspired by statistical physics.
- Local beam search.
- Genetic algorithms: inspired by evolutionary biology.

HILL-CLIMBING

Hill-Climbing Search

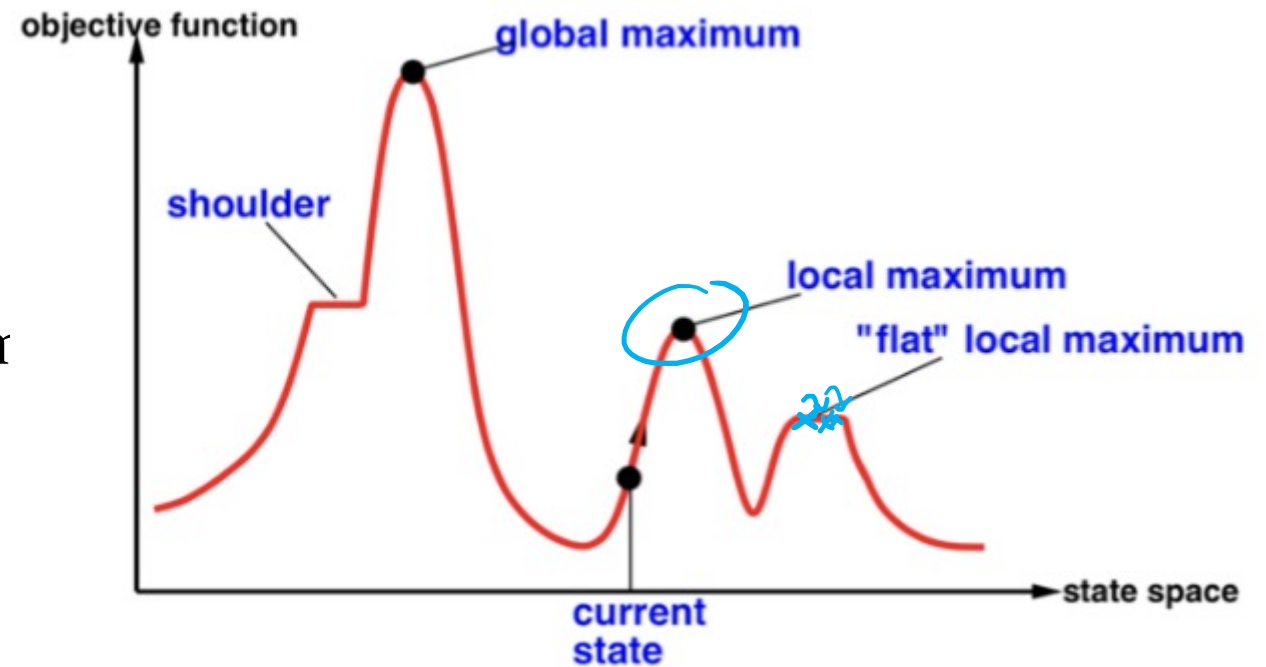
- Also called greedy local search.
- Looks only ^{for} to immediate good neighbors and not beyond.
- Continually moves in the direction of increasing value (i.e., uphill) to find the top of the mountain.
- Terminates when it reaches a “peak”, no neighbor has a higher value
- Only records the state and its objective function value.
- A node is a state and a value.



Hill-Climbing Search

- ISSUES:

- Can terminate with a local maximum global maximum or can get stuck and no progress is possible.
- Its success depends very much on the shape of the state-space land-scape: if there are few local maxima, random-restart hill climbing will find a “good” solution very quickly.



Hill-Climbing Search

function HILL-CLIMBING(initialState)

returns State that is a local maximum

initialize current **with** initialState

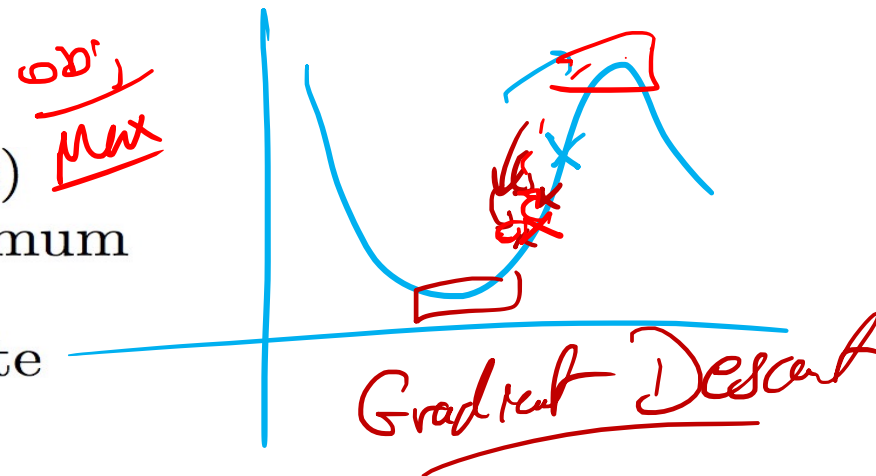
loop do

→ neighbor = a highest-valued successor of current

if neighbor.value \leq current.value:

return **current.state**

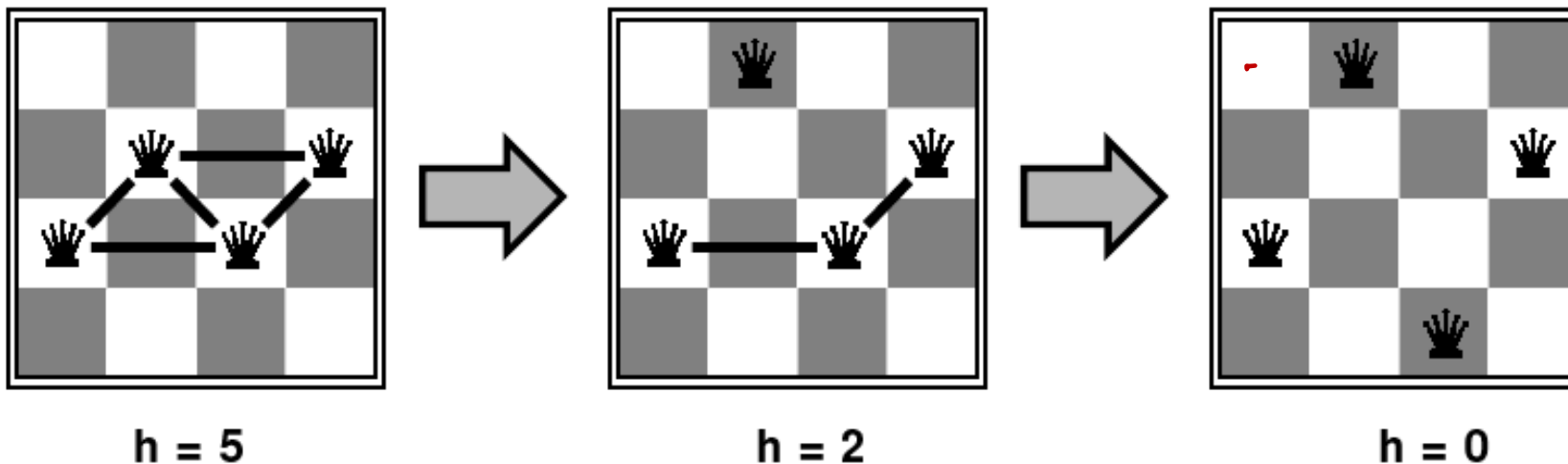
 current = neighbor



- The hill-climbing search algorithm, which is the **most basic** local search technique. At each step the current node is replaced by the **best neighbor**; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h .

Hill-Climbing Search - Example

- Back to 8-queens problem
- Place 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.



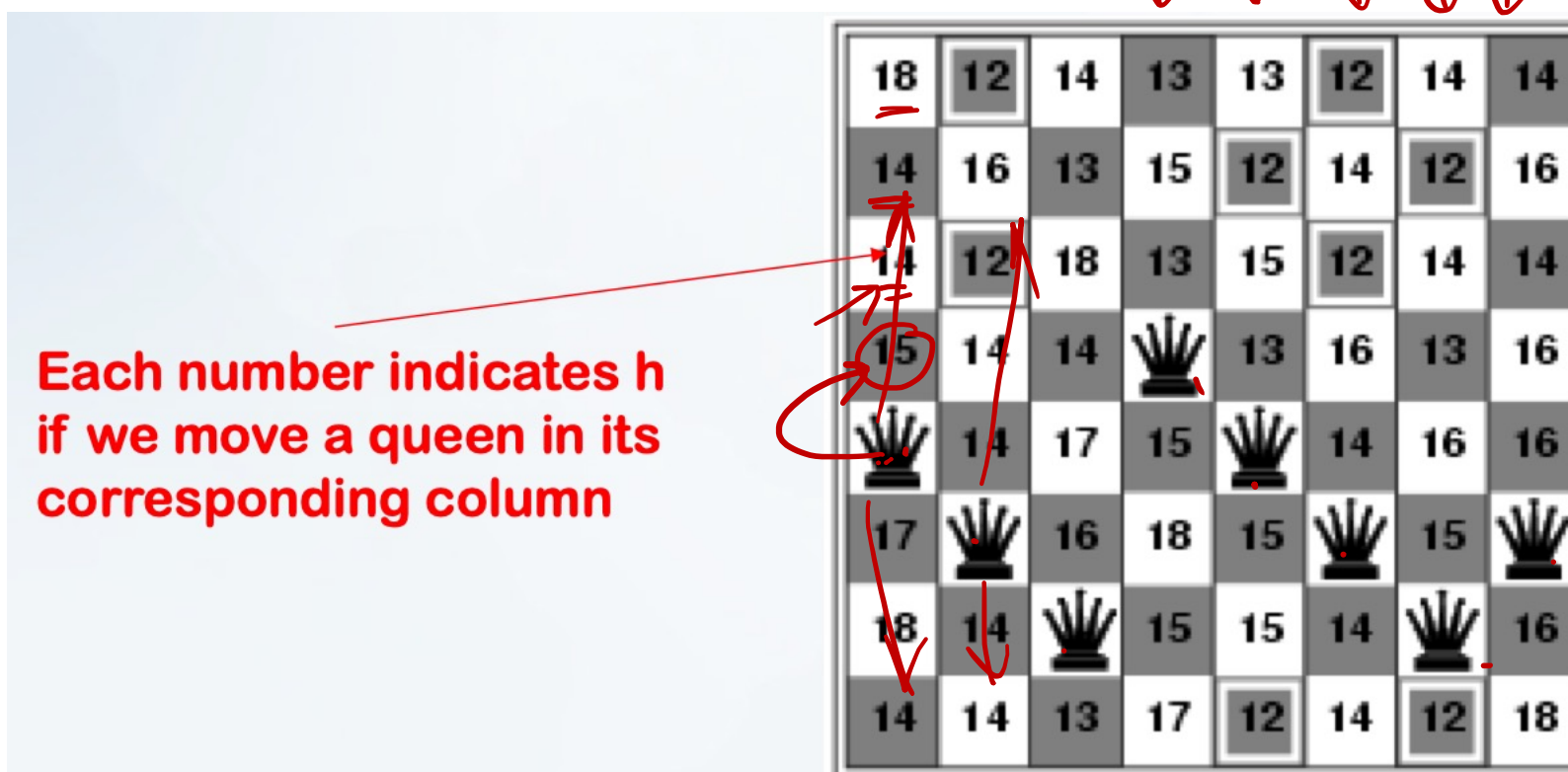
Hill-Climbing Search - Example

- The heuristic cost function h is the number of pairs of queens that are attacking each other, either directly or indirectly
- The global minimum of this function is zero, which occurs only at perfect solutions

$h = 0 \rightarrow$ No queen attacking
the other

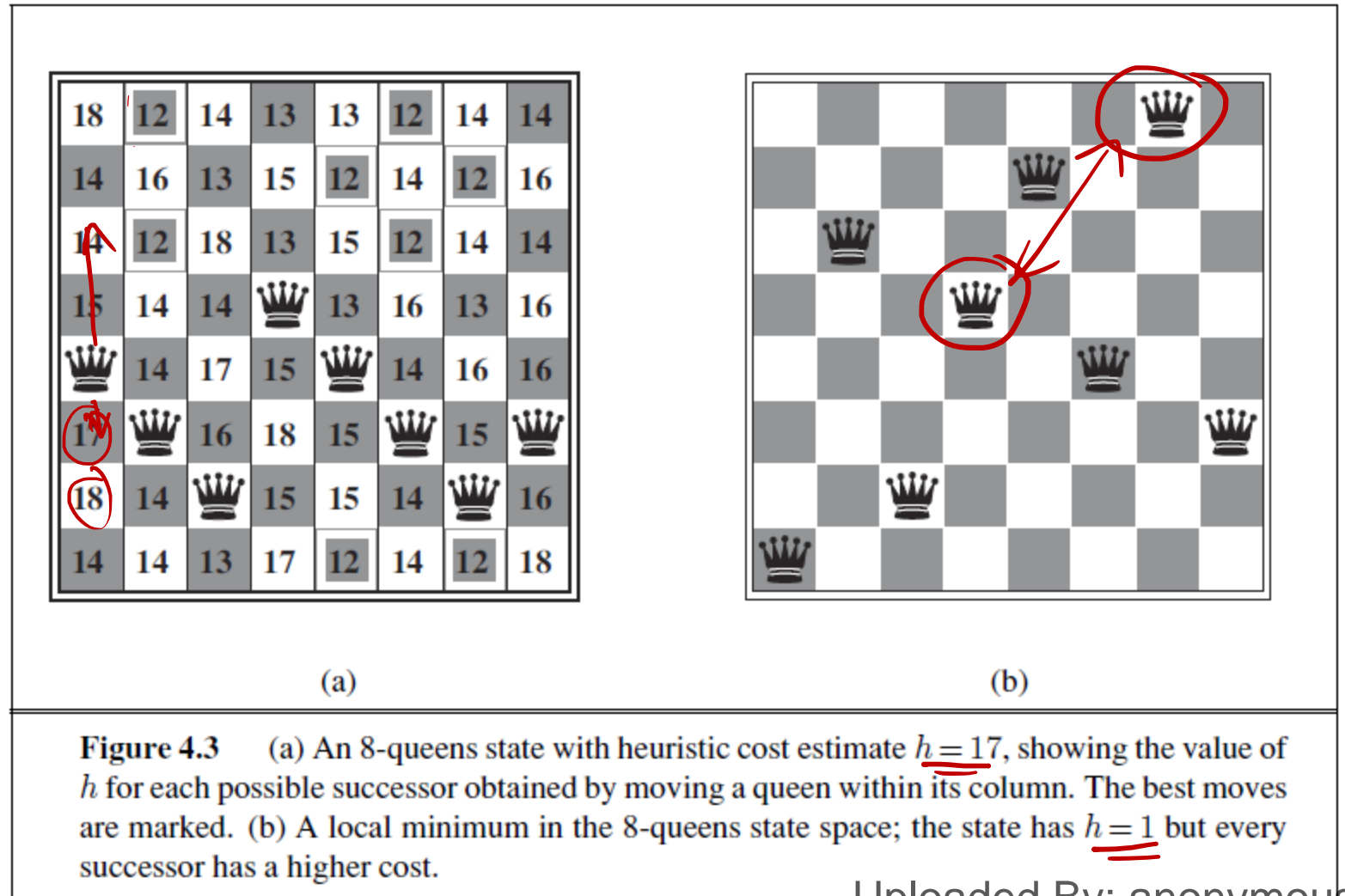
Hill-Climbing Search - Example

- h = number of pairs of queens that are attacking each other, either directly or indirectly



Hill-Climbing Search - Example

- Local search will use the complete-state formulation (in which each state has 8 queens on the board, one per column)

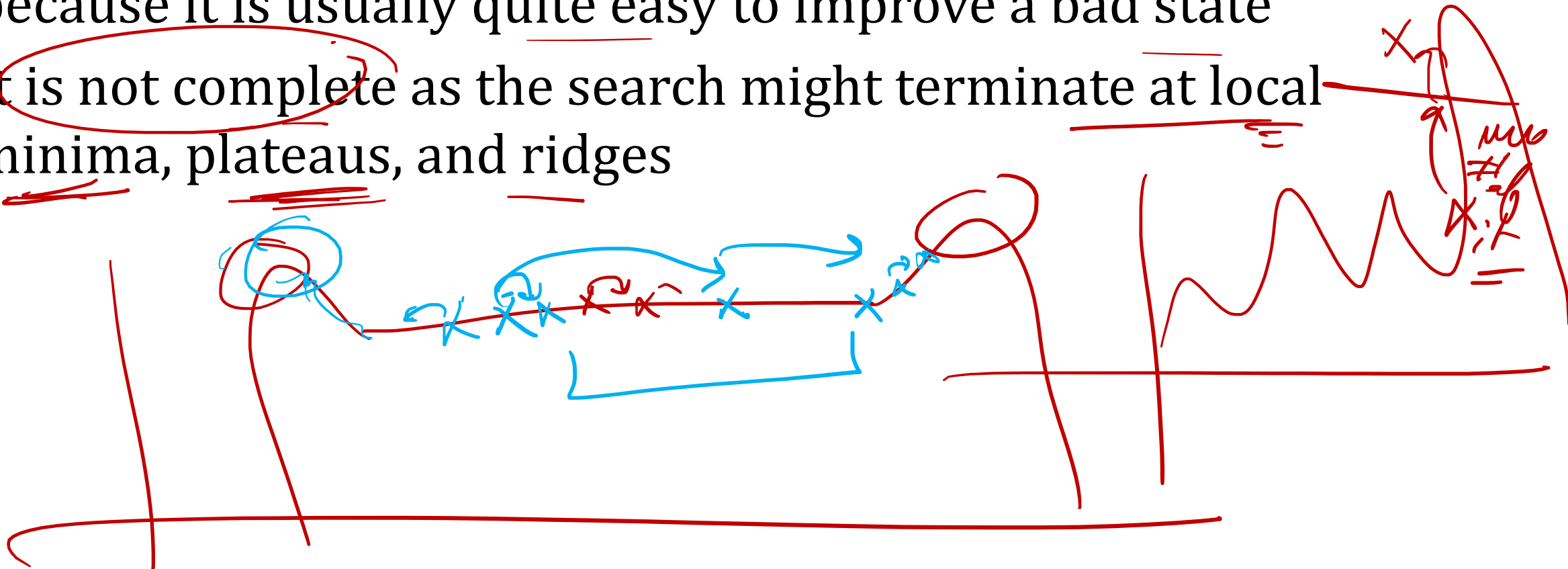


Hill-Climbing Search

- Hill-climbing continuously moves in the direction of increasing value (until the peak is reached-termination)
- It only looks one step ahead to determine if any successor is better than the current state (which if found, it will move to the best successor)

Hill-Climbing Search

- Hill climbing often makes rapid progress toward a solution because it is usually quite easy to improve a bad state
- It is not complete as the search might terminate at local minima, plateaus, and ridges



Hill-Climbing Search - Variants

- Other variants of hill climbing include
 - **Sideways moves** escape from plateau where best successor has same value as the current state.
 - **Random-restart** hill climbing overcomes local maxima: start over when no progress is made (do a random restart from a randomly generated initial state and perform hill-climbing search)
 - Stochastic hill climbing chooses at random among the uphill moves.



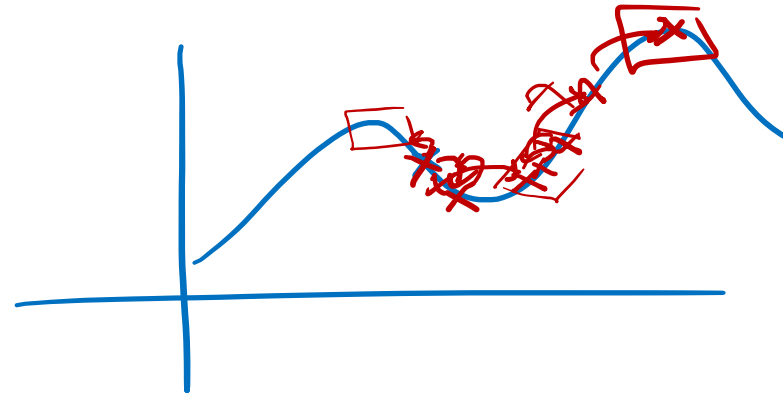
Stochastic Local Search



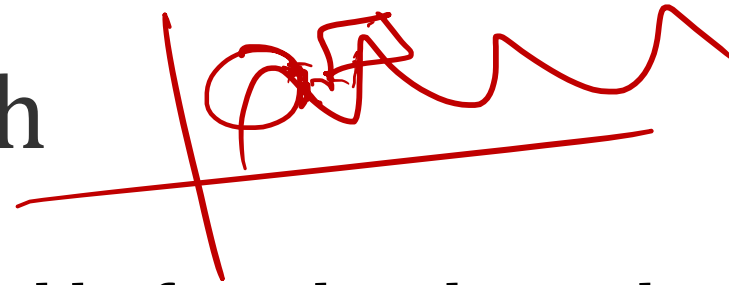
- Local search algorithms are concerned with some effective mechanisms of escaping from local minimas
- In stochastic local search, move to the best neighbour; the neighbour with the best evaluation function (Greedy search)
- Then we will use randomness to avoid getting stuck in local minima

Stochastic Local Search

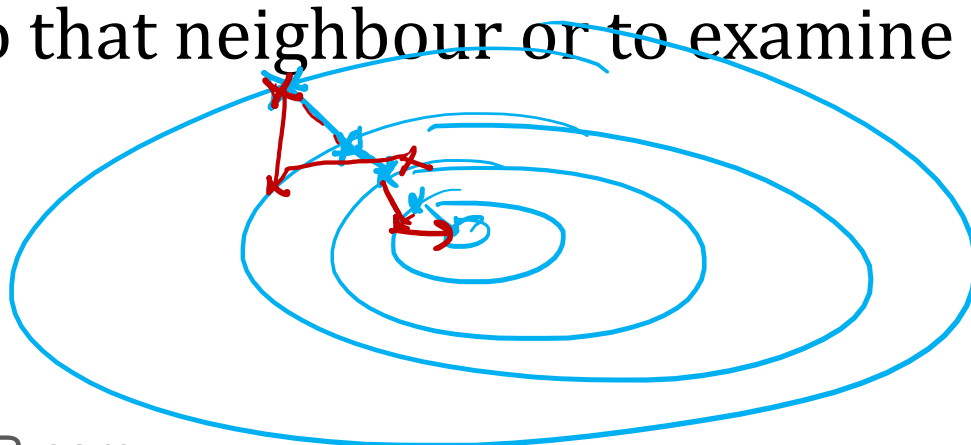
- Random initialization step
- Randomised search steps
 - Here, suboptimal/worsening steps are allowed.
 - Results in improved performance and more robust algorithm
 - The degree of randomness of the algorithm is controlled by a noisy parameter



Stochastic Hill-Climbing Search



- Not all neighbouring nodes are examined before deciding which node to select.
- A neighbouring node is selected at random, then based on the amount of improvement in that neighbour the algorithm decides whether to move to that neighbour or to examine another.



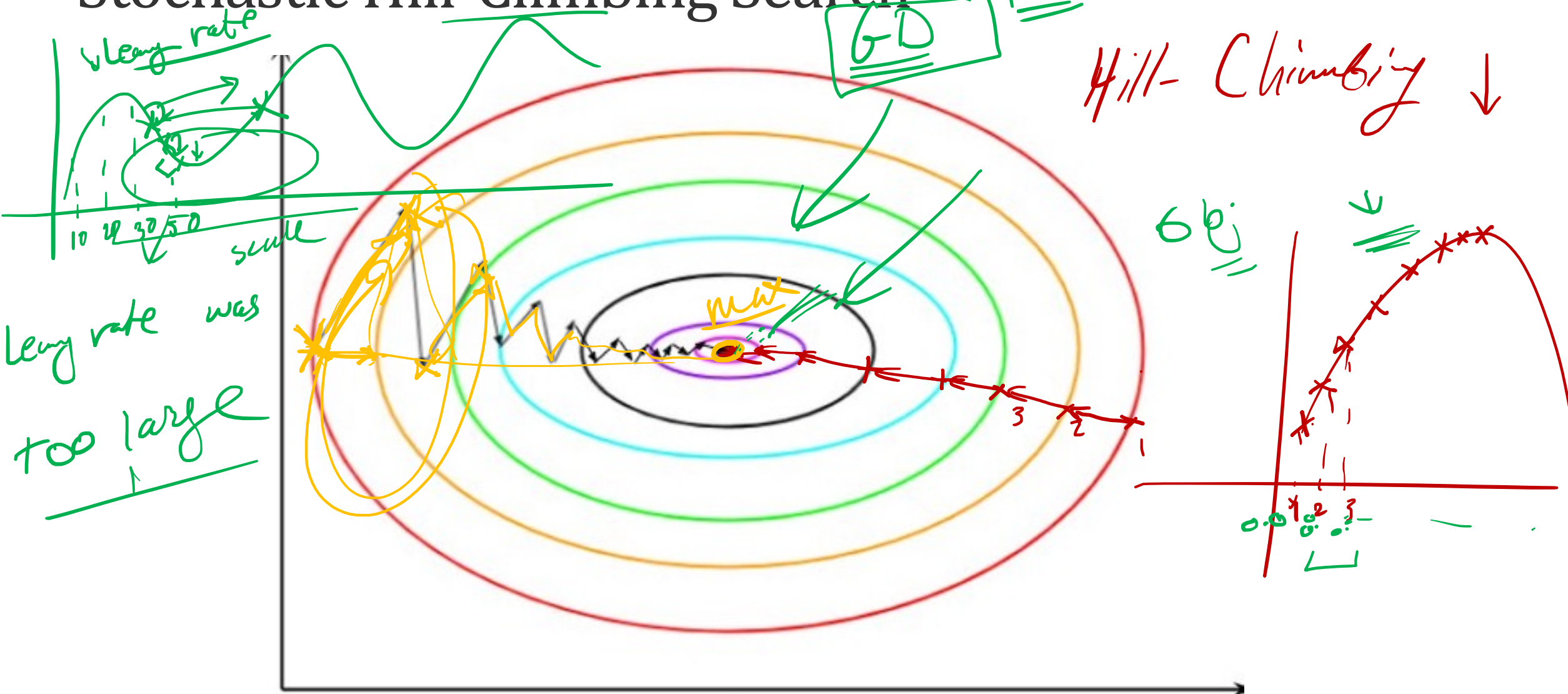
Contour plot

Stochastic Hill-Climbing Search

- **Algorithm**

- *Repeat until a solution is found or the current state does not change.*
 - *Select a state that has not been yet applied to the current state.*
 - *Apply the successor function to the current state and generate all the neighbour states.*
 - *Among the generated neighbour states which are better than the current state choose a state randomly (or based on some probability function).*
 - *If the chosen state is the goal state, then return success, else make it the current state and repeat step 2 of the second point.*
- *Exit from the function.*

Stochastic Hill-Climbing Search



Hill-Climbing Search – Advantages

- Used in various optimization problems, such as scheduling, route planning, and resource allocation
- Simple and intuitive algorithm
- Can be modified to include other heuristics or constraints
- A good choice to find a quick solution (as it finds a local maxima/minima) quickly

Hill-Climbing Search – Disadvantages

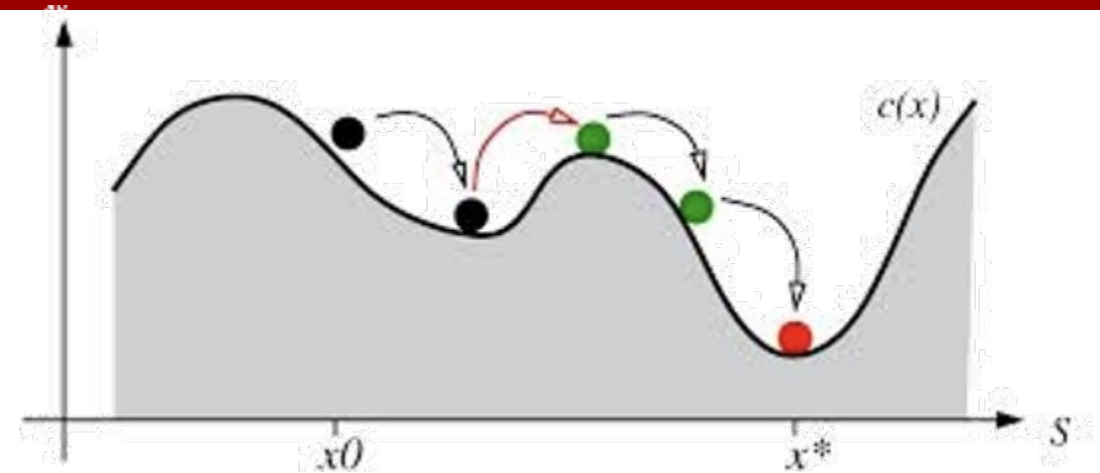
minimize
↑
maximize

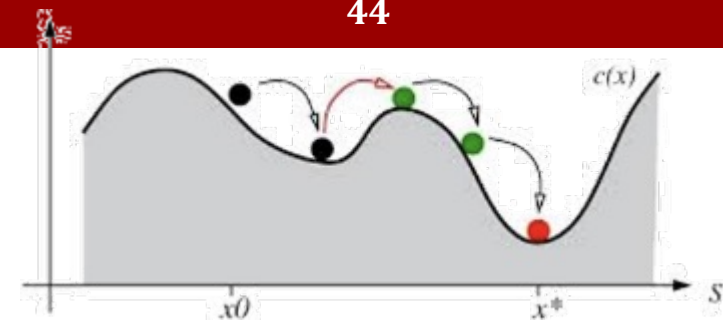
- Can get stuck in local optima, meaning that it may not find the global optimum of the problem
- As it depends on a random initial value, a poor initial solution may result in a poor final solution
- Does not explore the search space very thoroughly
- Less effective than other optimization algorithms such as genetic algorithms (at least for certain types of problems)

SIMULATED ANNEALING

Simulated Annealing

- Simulated Annealing (SA) is another iterative improvement algorithm in which randomness is incorporated to expand the search space and avoid becoming trapped in local minimum
- To avoid being stuck in a local maxima, it tries randomly (using a probability function) to move to another state, if this new state is better it moves into it, otherwise try another move... and so on.
- As the name implies, the algorithm simulates the process of annealing
- *Annealing* is a technique in metal-casting where molten metal is heated and then cooled in a gradual manner to evenly distribute the molecules into a crystalline structure





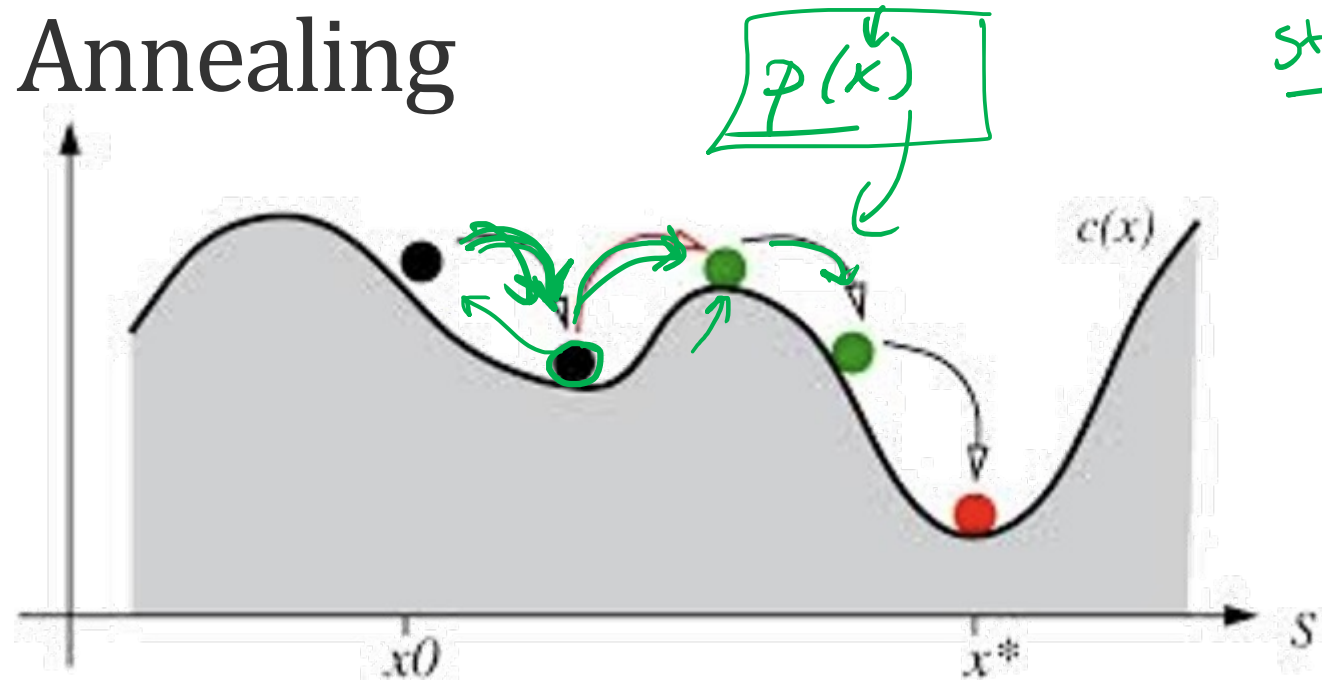
Simulated Annealing

- 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 very bumpy surface
- If we shake the surface, we can bounce the ball out of the local minimum—perhaps into a deeper local minimum, where it will spend more time. 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

- Simulated annealing combines hill climbing with random walk (that is, in each step, randomly select one of the neighbouring positions of the search space and move there) in such way that yields both completeness and efficiency
- Instead of picking the best move, a random move is picked
- If the situation is improved, then it is accepted. Otherwise, the algorithm accepts the move with some probability less than 1. The probability decreases exponentially with the “badness” of the move

Simulated Annealing



- Terminates when finding an acceptably good solution in a fixed amount of time, rather than the best possible solution.
- Locating a good approximation to the global minimum of a given function in a large search space.
- Widely used in VLSI layout, airline scheduling, etc.

Simulated Annealing

- The overall structure of the simulated-annealing algorithm is similar to hill climbing. Instead of picking the best move, however, it picks a random move. If the move improves the situation, it is always accepted. Otherwise, the algorithm accepts the move with some probability less than 1.

```

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

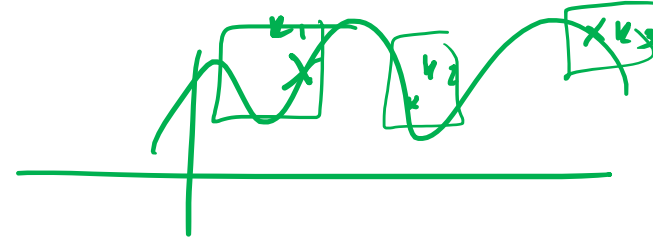
Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. The *schedule* input determines the value of the “temperature” *T* as a function of time.

Simulated Annealing

- The problem with this approach is that the neighbors of a state are not guaranteed to contain any of the existing better solutions which means that failure to find a better solution among them does not guarantee that no better solution exists.
- It will not get stuck to a local optimum.
- If it runs for an infinite amount of time, the global optimum will be found.

LOCAL BEAM SEARCH

Local Beam Search



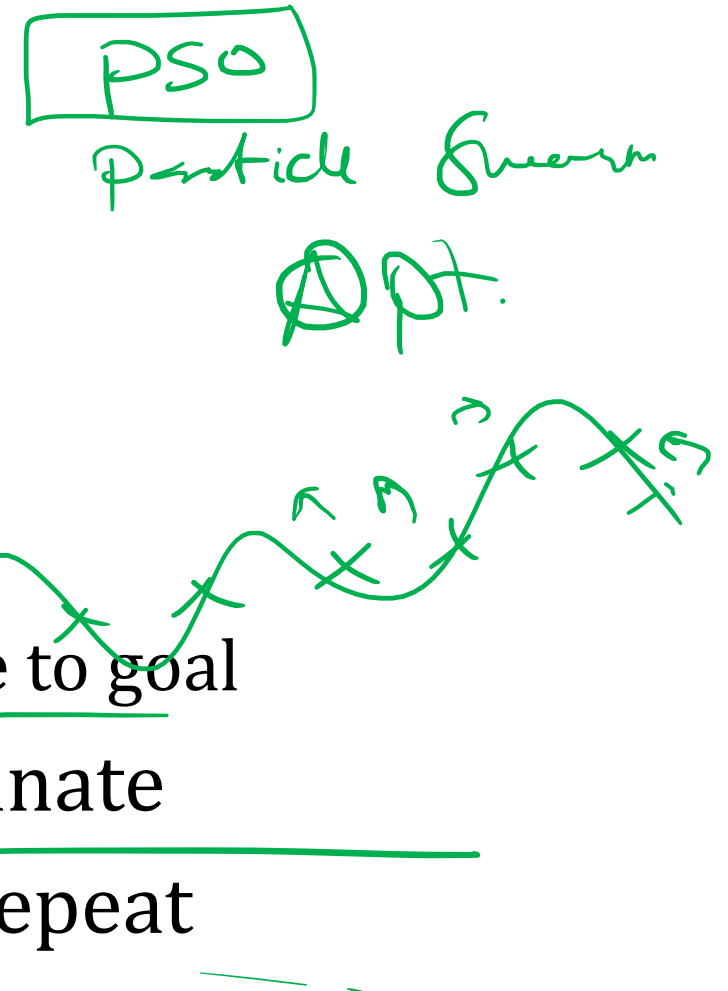
- Local beam search keeps track of k states
- It begins with k randomly generated states
- At each step, all the successors of all k states are generated. If any one is a goal, the algorithm halts.
- Otherwise, it selects the k best successors from the complete list and repeats

Local Beam Search

- It's simple idea is to keep around those states that are good, and remove the rest
- It is similar to Hill Climbing
 - Start from N initial states
 - Expand all N states and keep k best successors
- This would add the ability to avoid some local optima. It is mainly good when the search space is large

Local Beam Search - Algorithm

- Start with N random states
- Determine all successors of N states
 - Extend all paths one step
 - Reject all paths with loops
 - Sort all paths in queue by the estimated distance to goal
- Test if any of the successor is goal and terminate
- Else select N best from the successors and repeat




GENETIC ALGORITHMS

→ Genetic Algorithm

- Inspired by evolutionary biology and natural selection, such as inheritance.
- A genetic algorithm (GA) is a search technique used in computing to find true (or approximate) solutions to optimization and search problems.
- GAs are categorized as global search heuristics.

Genetic Algorithm

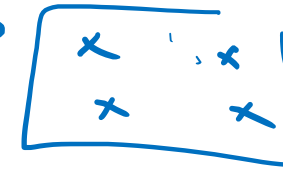
- Evolves towards better solutions.
- Starts with k randomly generated states (population); each state is an individual. 
- In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness), and modified to form a new population.
- A successor state is generated by combining two parent states, rather by modifying a single state.

Genetic Algorithm - Terminology

- **Individual** - Any possible solution
- **Population** - Group of all individuals
- **Fitness** – Target function that we are optimizing (each individual has a fitness)
- **Trait** - Possible aspect (features) of an individual
- **Genome** - Collection of all chromosomes (traits) for an individual.
- The **selection** process for selecting the individuals who will become the parents of the next generation.

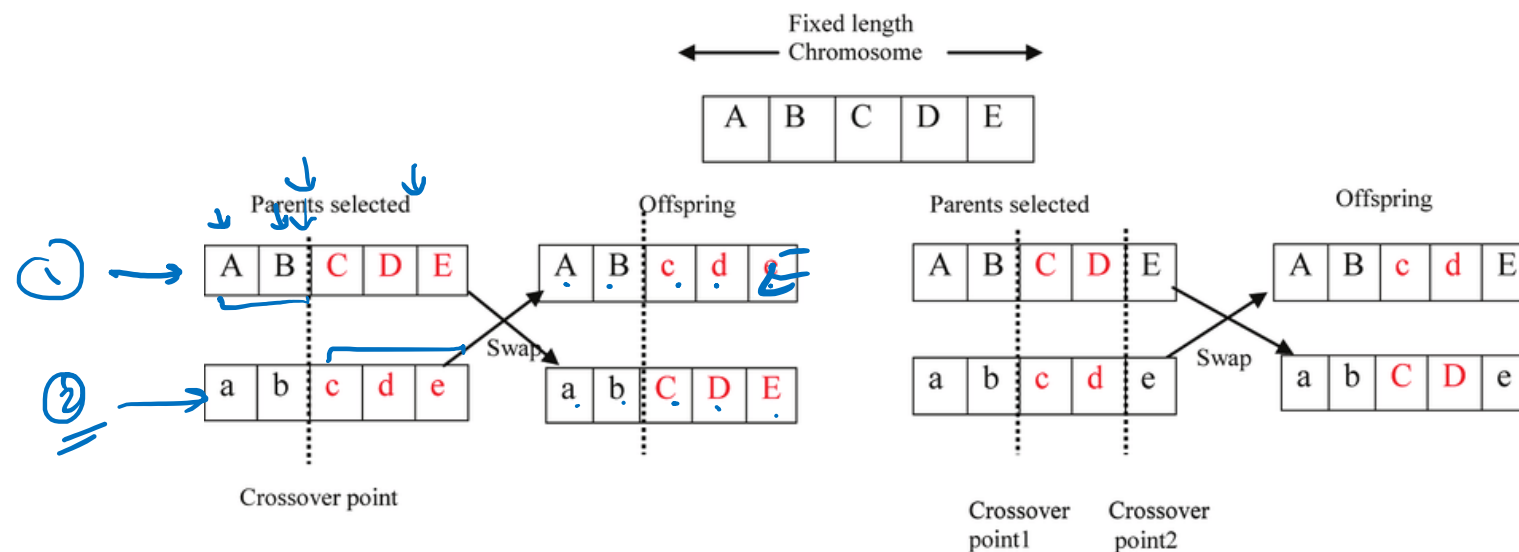
Genetic Algorithm - Terminology

iteration
high
= 500



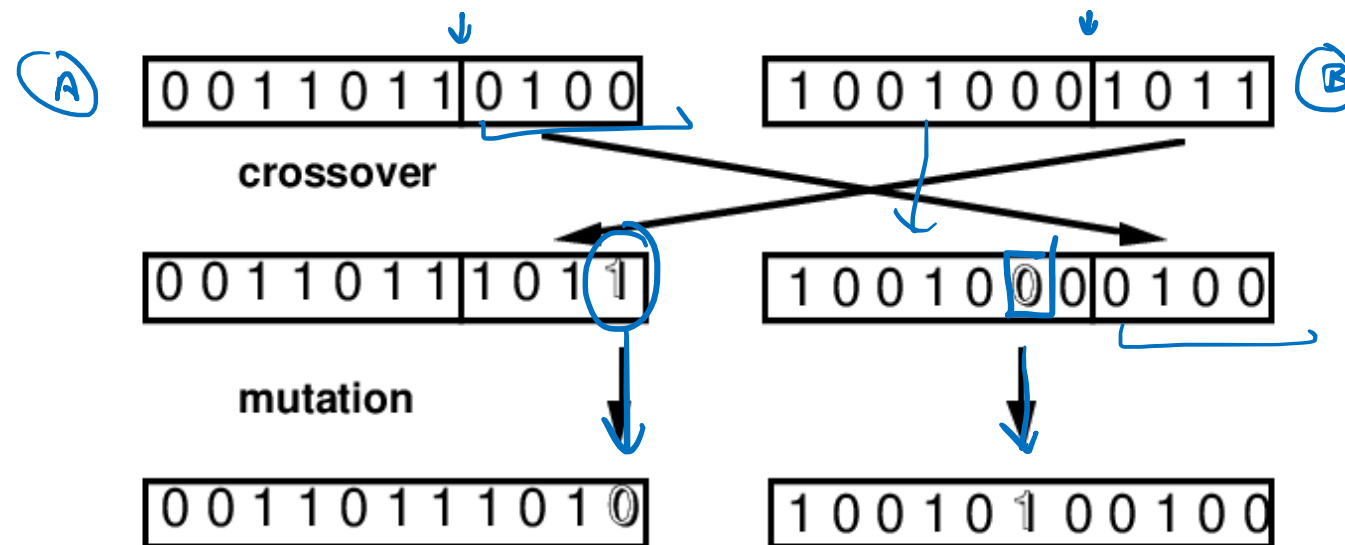
- single point
- two-point

- Crossover - the recombination procedure. One common approach (assuming $\rho = 2$), is to randomly select a crossover point to split each of the parent strings, and recombine the parts to form two children, one with the first part of parent 1 and the second part of parent 2; the other with the second part of parent 1 and the first part of parent 2.



Genetic Algorithm - Terminology

- The mutation rate: determines how often offspring have random mutations to their representation. Once an offspring has been generated, every bit in its composition is flipped with probability equal to the mutation rate.



Genetic Algorithm

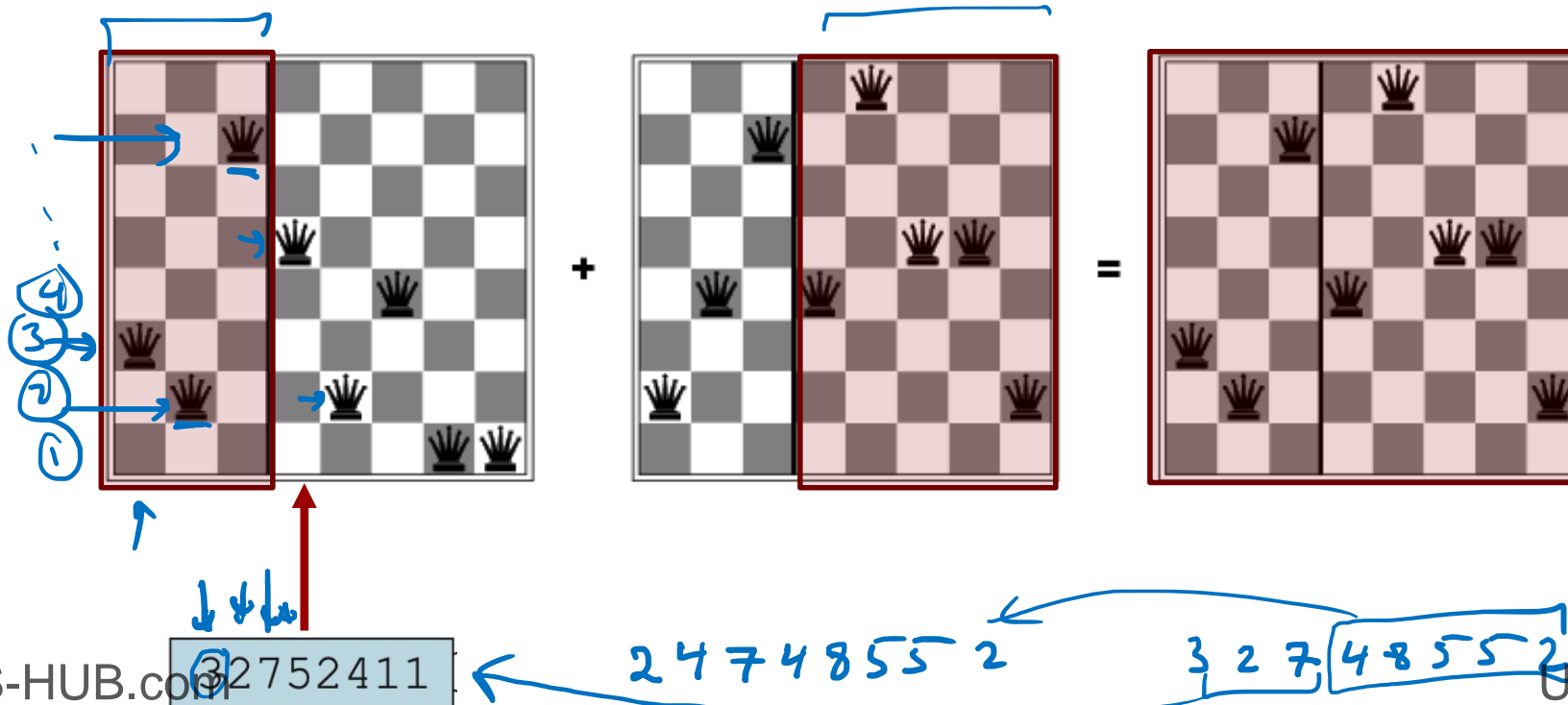


- GA follows the concept of “survival of the fittest” in which, better and better solutions evolve from previous generations until a near optimal solution is obtained
- GA works iteratively and candidates are represented as strings of genes called chromosomes
- Used to improve the performance of other AI methods (such as finding the best parameters of NN)

Genetic Algorithm

8-queens

- A better state is generated by combining two parent states.
- The good genes (features) of the parents are passed onto the children.



Genetic Algorithm

- Each state, or **individual**, is represented as a string over a finite alphabet— (often a Boolean string), just as DNA is a string over the alphabet ACGT
- For example, an 8-queens state must specify the positions of 8 queens, each in a column of 8 squares which is then represented as 8-digits

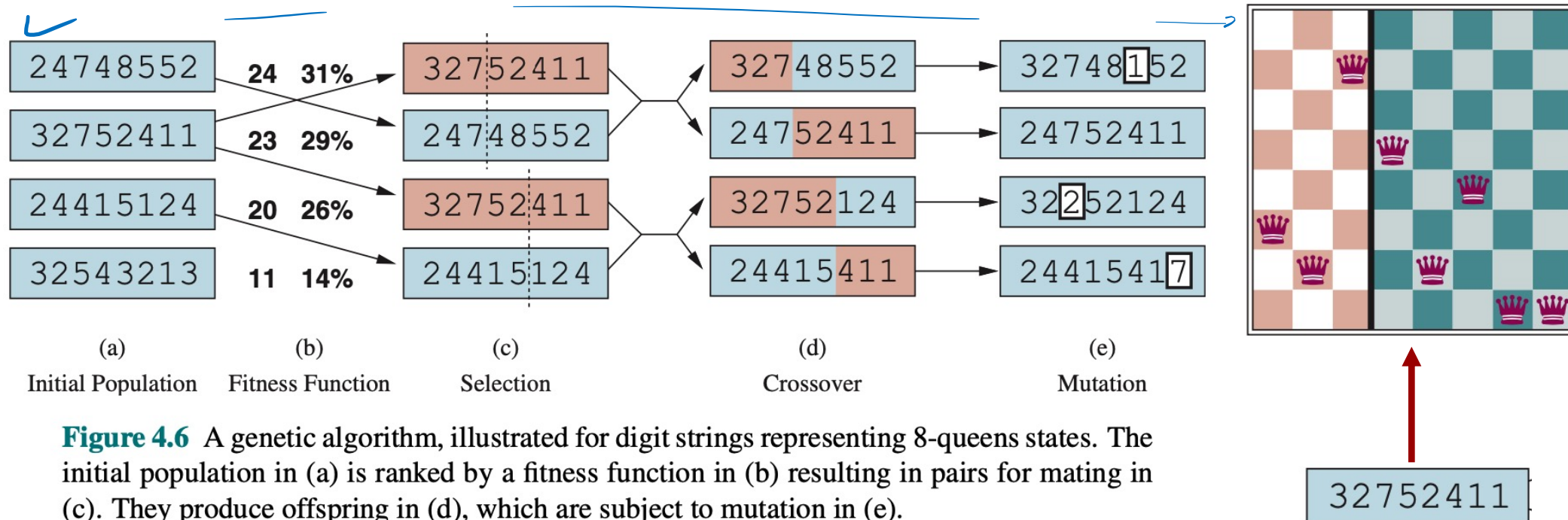


Figure 4.6 A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

90%

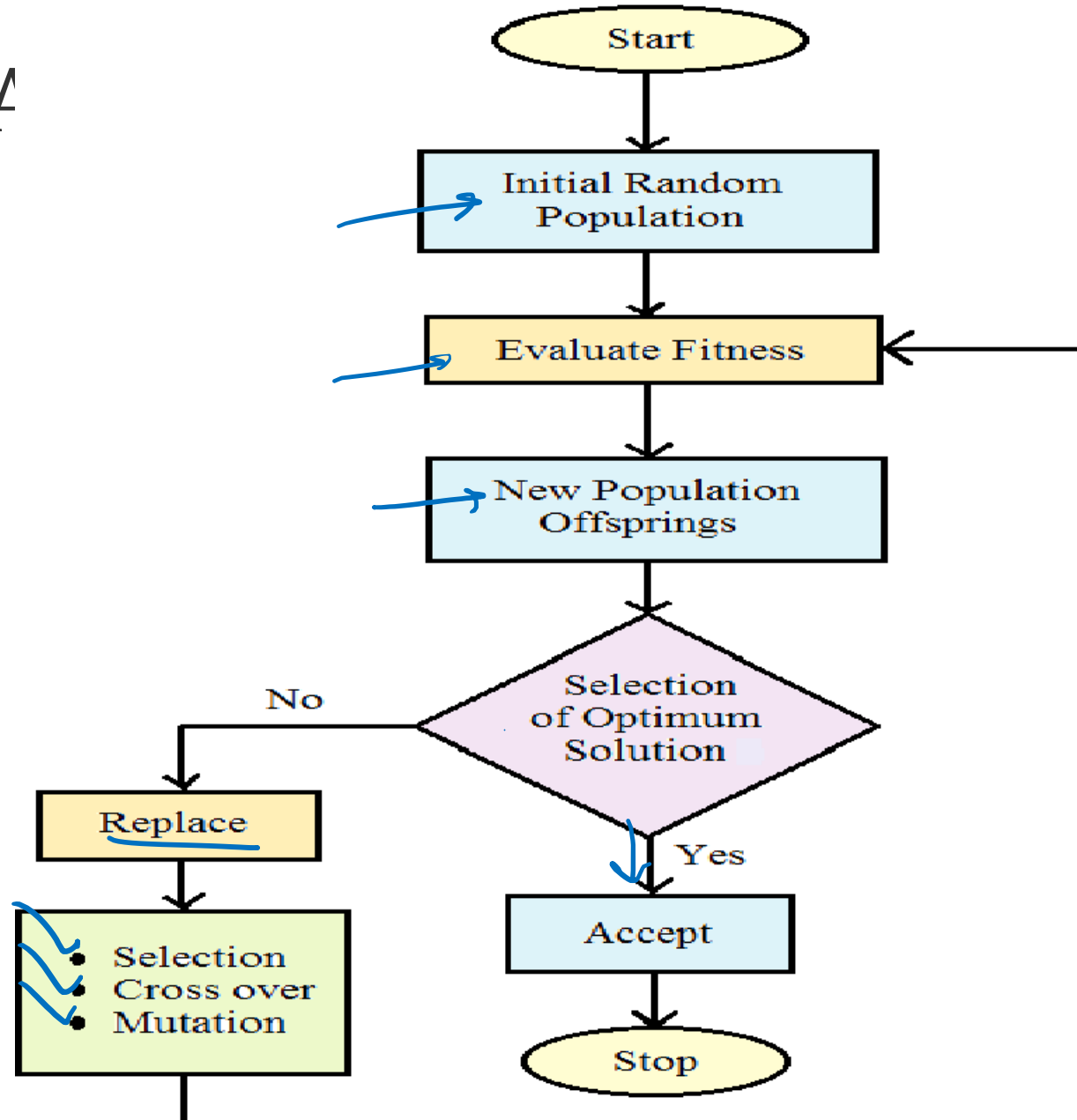
Genetic Algorithm

- In the production of the next generation of states, each state is rated by the objective function (the fitness function)
 - Higher values for better states.
- The next generation is produced as an improved solution by selecting parents with higher fitness ratings (selection), crossover, and mutation
- Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

Genetic Algorithm – Basic Algorithm

- Start with a large “population” of randomly generated “attempted solutions” to a problem
- Repeatedly do the following:
 - Evaluate each of the attempted solutions
 - (probabilistically) keep a subset of the best solutions
 - Use these solutions to generate a new population
- Quit when you have a satisfactory solution (or you run out of time)

Genetic A



Genetic Algorithm

$$\frac{8 \times 7}{2}$$

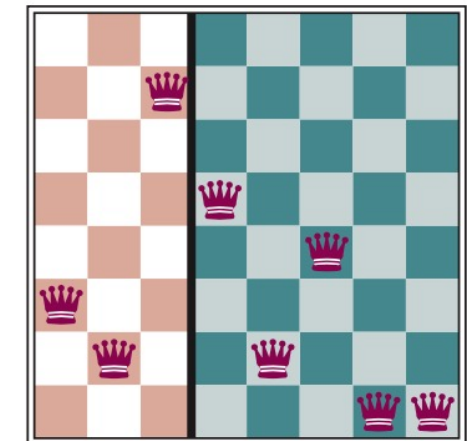
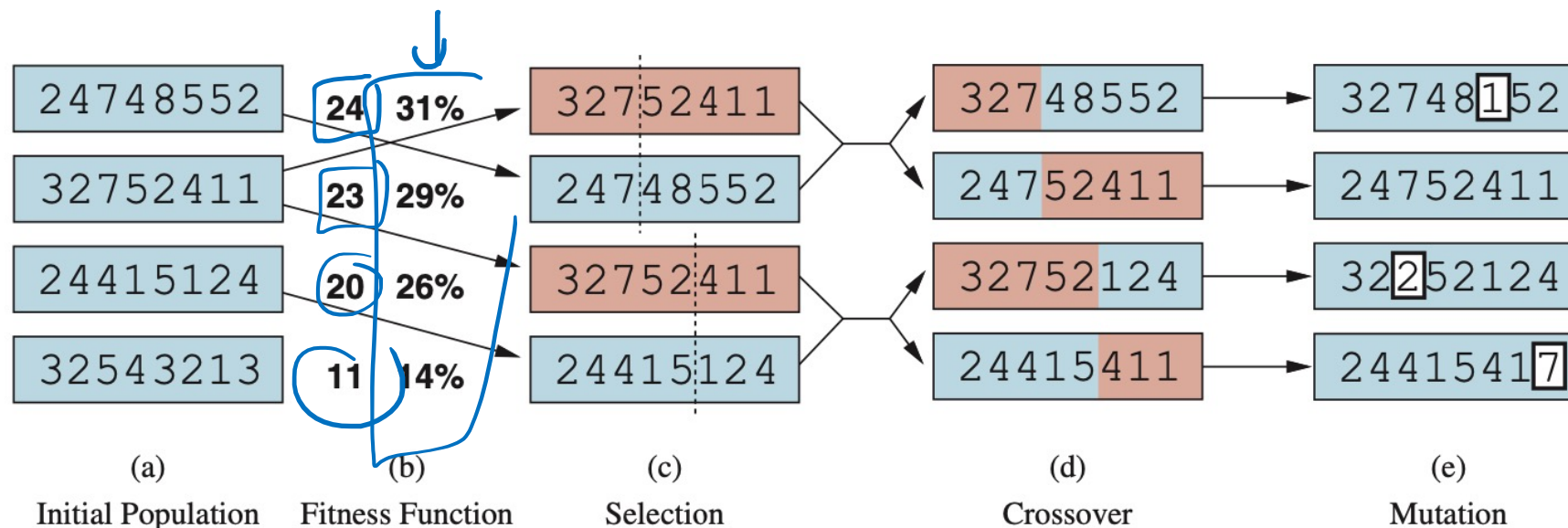
$$7 + \underline{6} + 5 + 4 + \dots$$

$$\binom{8}{2}$$

- A **fitness function** should return higher values for better states
- So for example, a possible fitness function is the number of non-attacking pairs of queens that we are interested to maximize, which has a value of 28 for a solution
 - In other words we want the number of attacking pairs of queens to be zero in the solution assignment of the queens.
- Fitness function: number of non-attacking pairs of queens (min = 0; max = $8 * 7 / 2 = 28$)

Genetic Algorithm

- Consider a population of four 8-digit strings, each representing a state of the 8-queens puzzle
 - the c -th digit represents the row number of the queen in column c

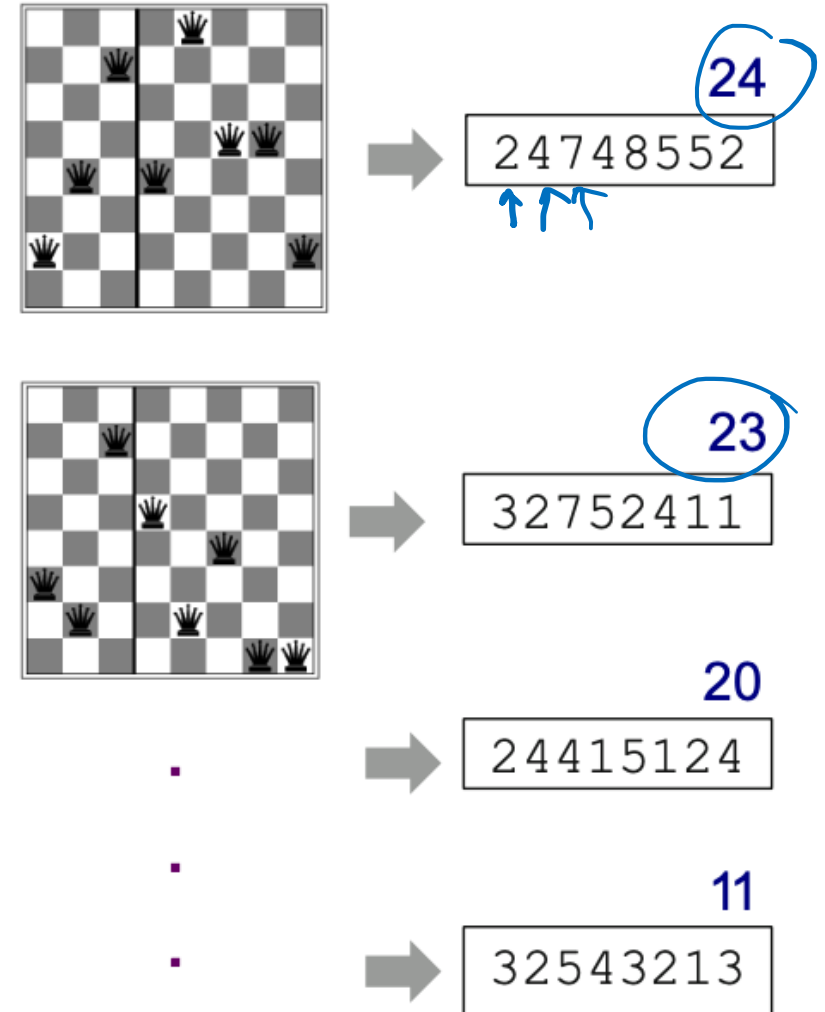


32752411

Figure 4.6 A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic Algorithm - Fitness

- Example of fitness functions

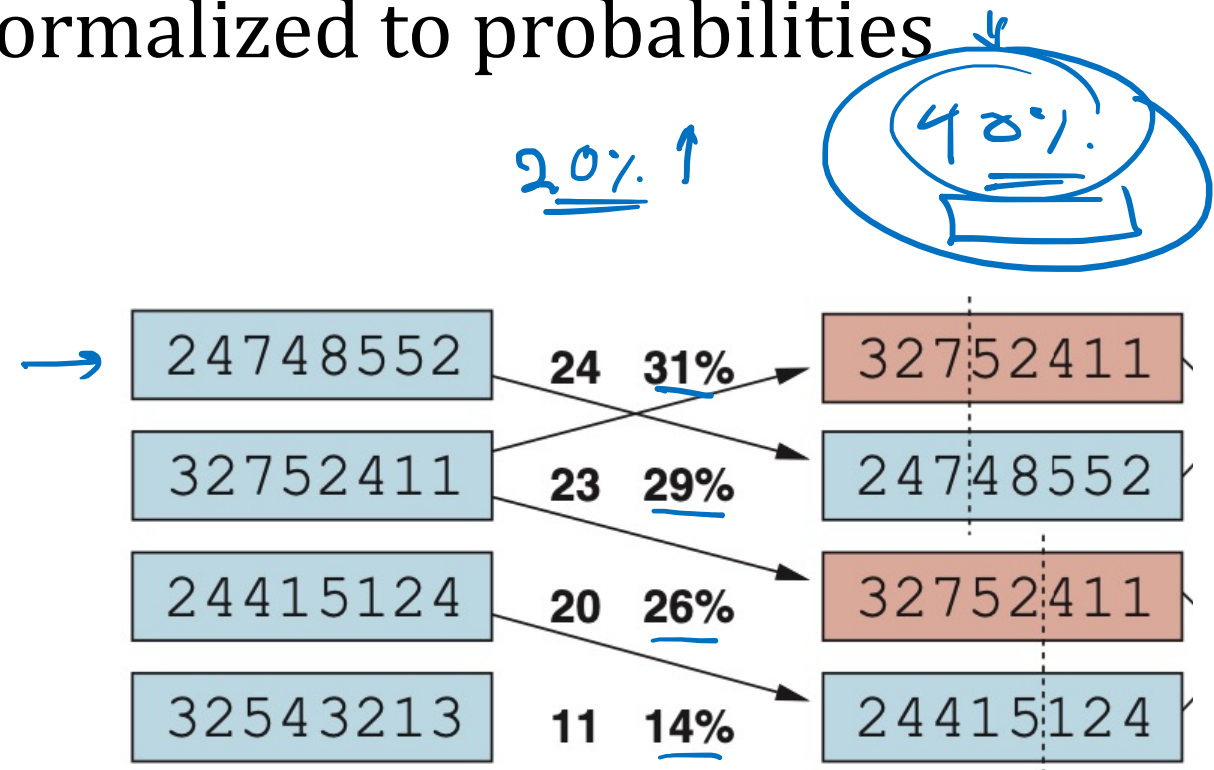


Genetic Algorithm – Relative fitness

- The values of the four states are 24, 23, 20, and 11
- The fitness scores are then normalized to probabilities

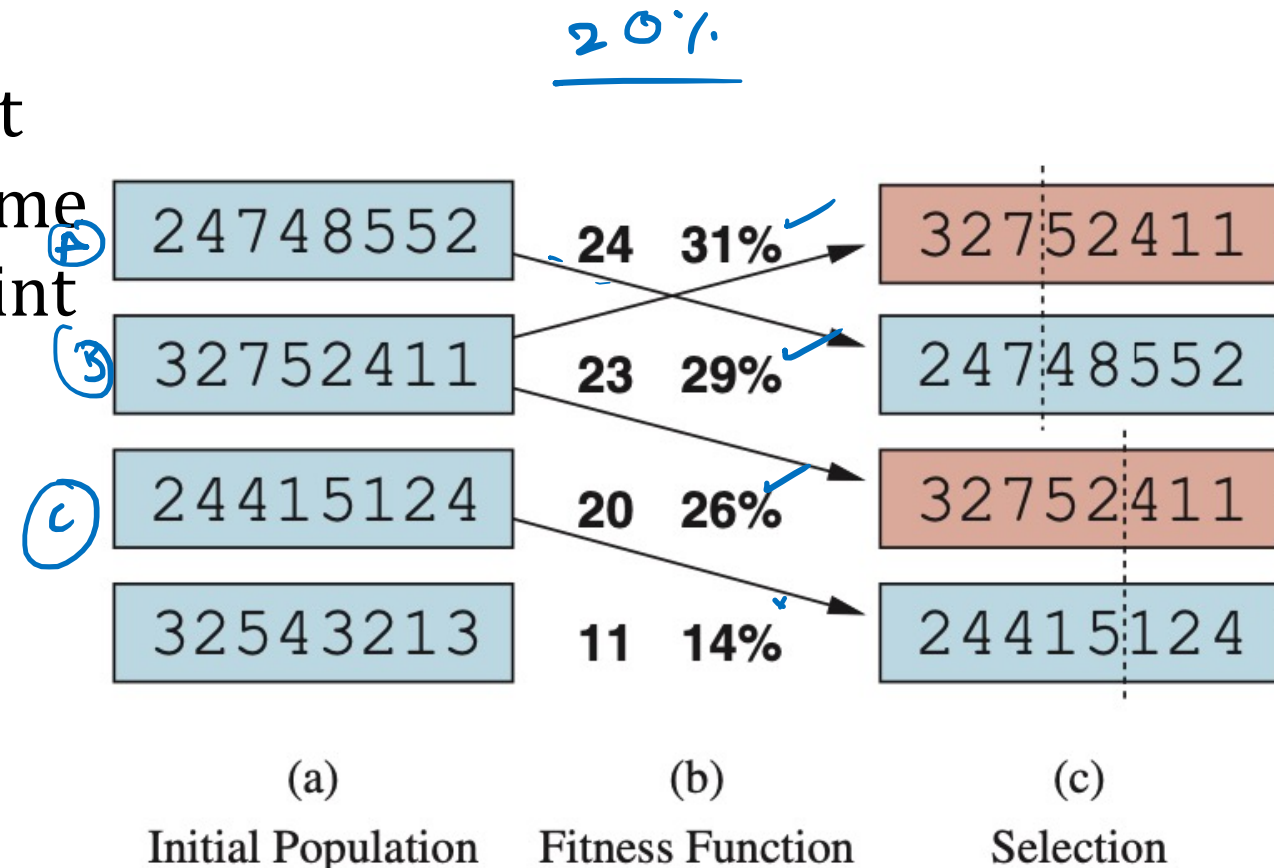
- $\frac{24}{(24+23+20+11)} = 31\%$
- $\frac{23}{(24+23+20+11)} = 29\%$
- $\frac{20}{(24+23+20+11)} = 26\%$
- $\frac{11}{(24+23+20+11)} = 14\%$

- Relative fitness = $\frac{fitness(x_i)}{\sum_{i=1}^n fitness(x_i)}$



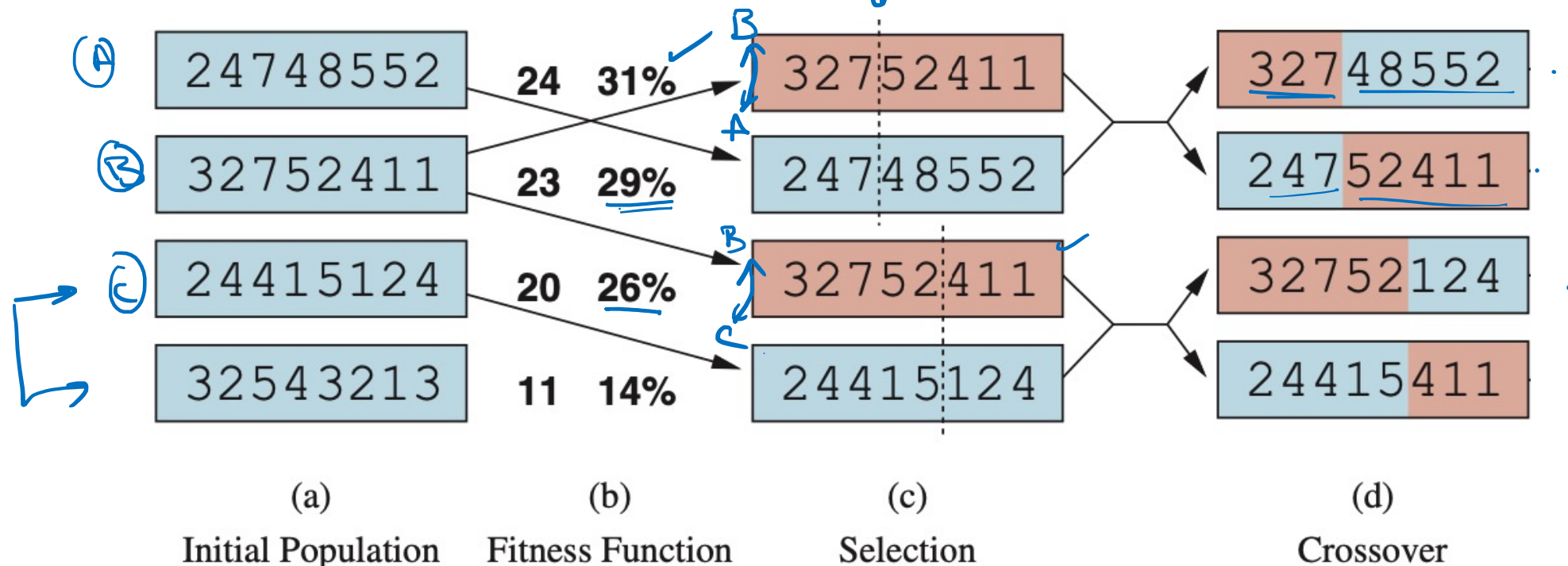
Genetic Algorithm - Selection

- Pairs of individuals are selected at random for reproduction w.r.t. some probabilities. Pick a crossover point per pair



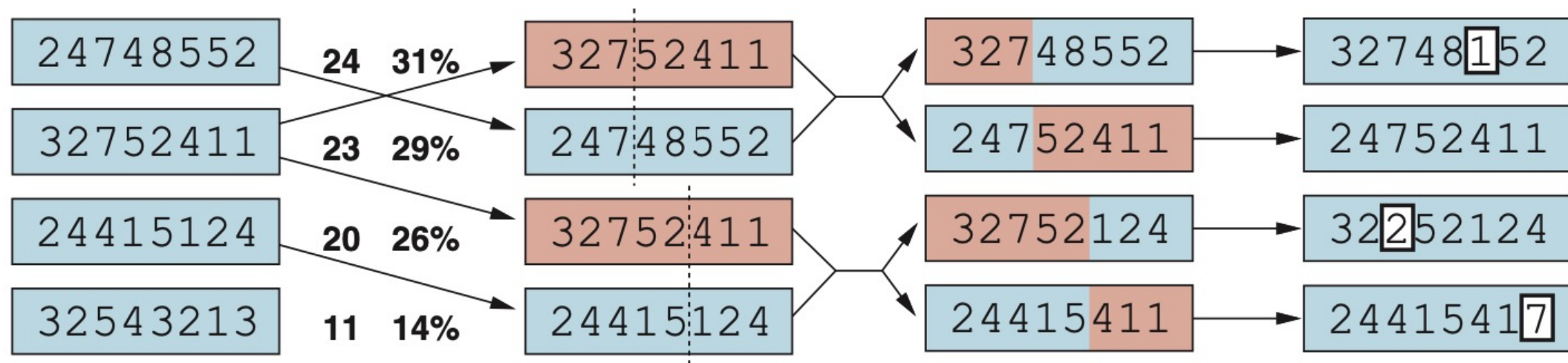
Genetic Algorithm - Crossover

- A crossover point is chosen randomly in the string. Offspring are created by crossing the parents at the crossover point.
- Various rules of selection can be applied: threshold?



Genetic Algorithm - Crossover

- Each element in the string is also subject to some mutation with a small probability
- **Crossover** means choosing a random position in the string (say, after 3 digits) and exchanging the segments either to the right or to the left of this point with another string partitioned similarly to produce two new offspring
- We cross over the parent strings at the crossover points, yielding new offspring.



(a)

(b)

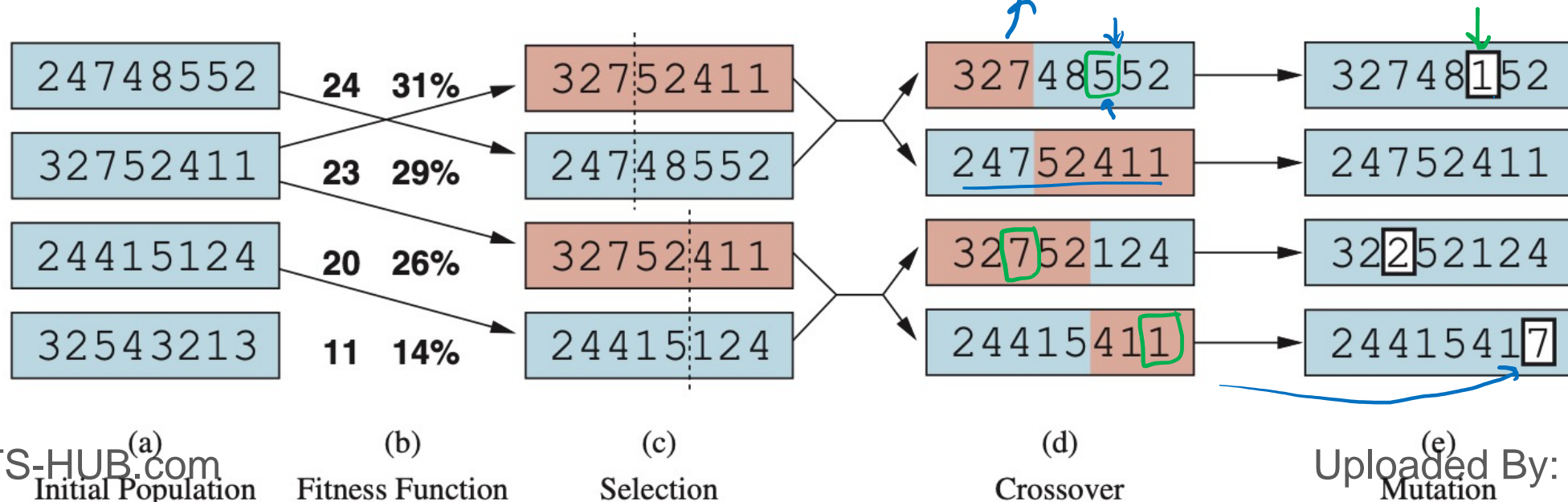
(c)

(d)

(e)

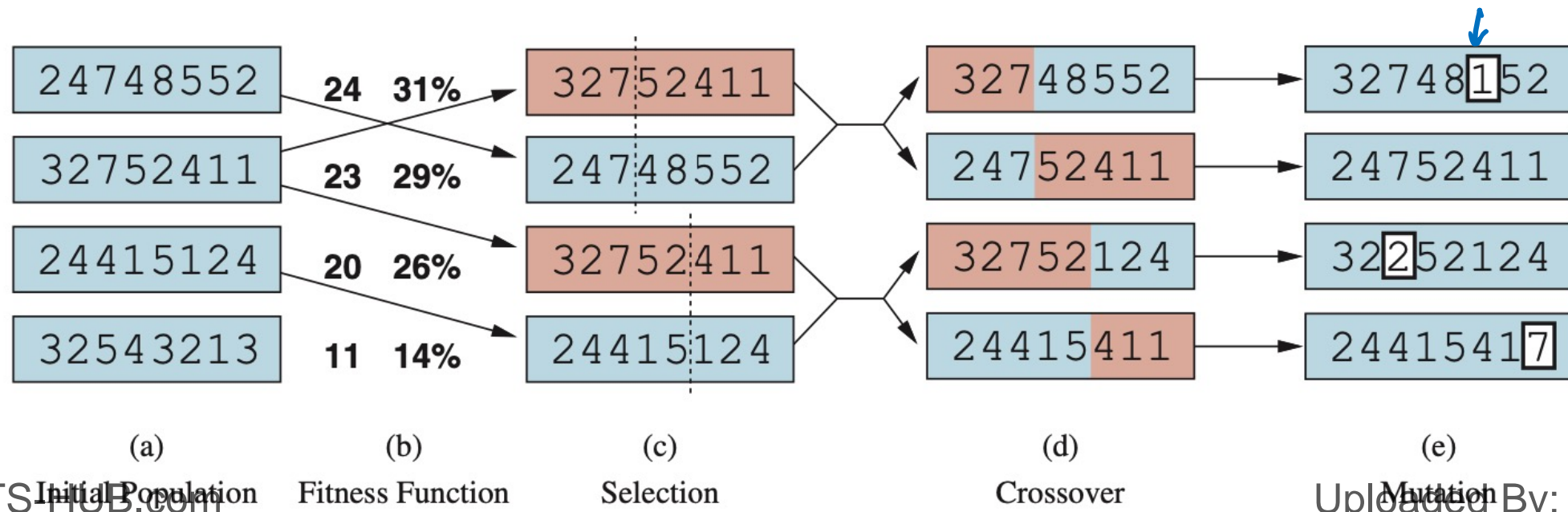
Genetic Algorithm

- Mutation: each location in each string is subject to random mutation with a small independent probability. One digit was mutated in the first, third, and fourth offspring. In the 8-queens problem, this corresponds to choosing a queen at random and moving it to a random square in its column.



Genetic Algorithm

- **Mutation** is an arbitrary change in a situation
- Sometimes it is used to prevent the algorithm from getting stuck
- The procedure changes One to Zero (in case of binary string). This change occurs with a very low probability (say 1 in 1000)



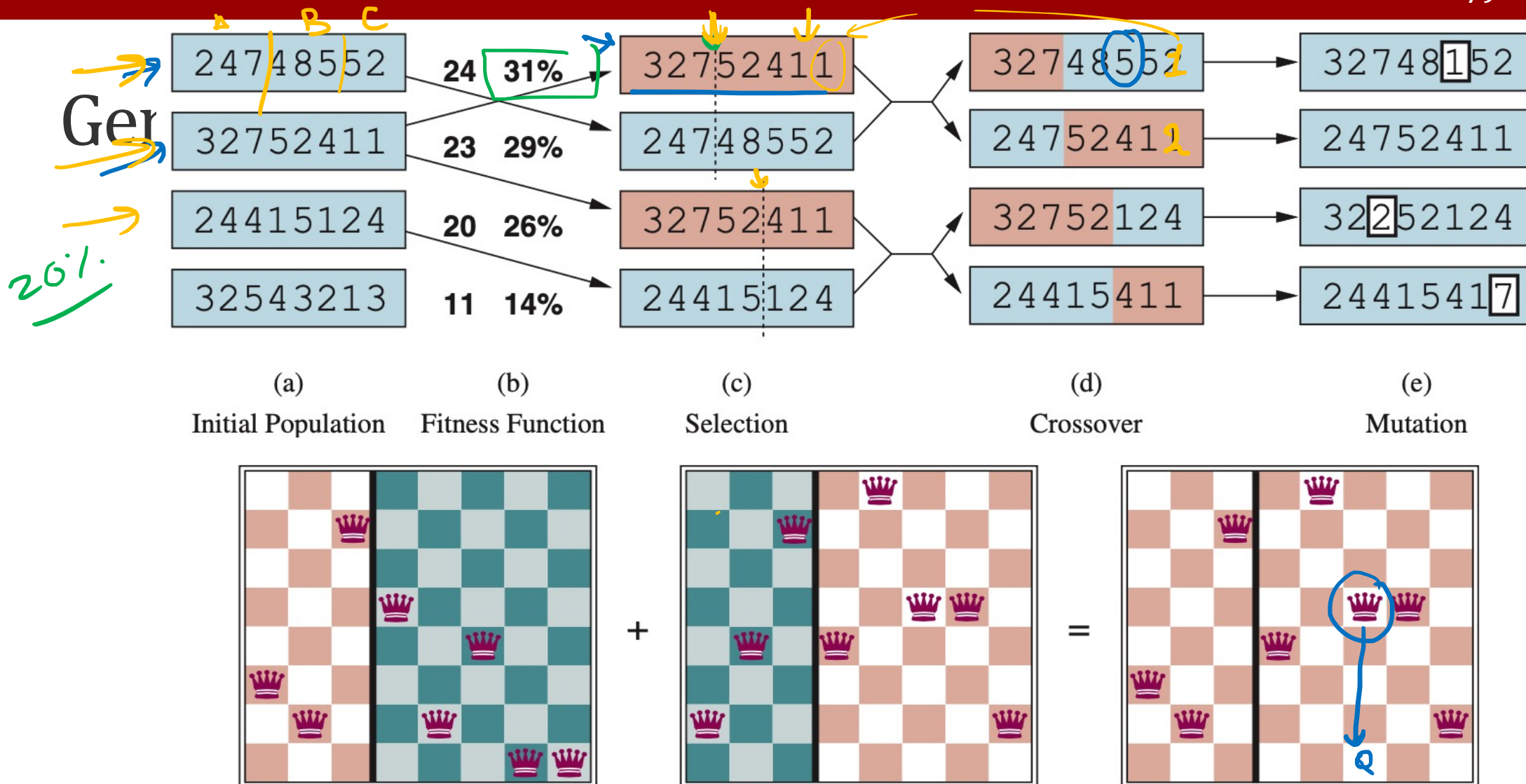


Figure 4.7 The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d). The green columns are lost in the crossover step and the red columns are retained. (To interpret the numbers in Figure 4.6: row 1 is the bottom row, and 8 is the top row.)

Genetic Algorithm

function GENETIC-ALGORITHM(population, fitness-function)

returns an individual

repeat

initialize new-population **with** \emptyset

for $i=1$ to size(population) **do**

→ $x = \text{random-select}(\text{population}, \text{fitness-function})$

→ $y = \text{random-select}(\text{population}, \text{fitness-function})$

→ $\text{child} = \text{cross-over}(x, y)$

→ mutate (child) with a small random probability

→ add child to new-population

population = new-population

→ **until** some individual is fit enough or enough time has elapsed

return the best individual in population w.r.t. fitness-function

Genetic Algorithm

- Genetic Algorithms are considered robust search algorithm
- Suitable for optimization problem over large space state
- Robust to noise or change of data

