Local Search Algorithm and Optimization

Chapter#4

Outline

- Local Search: Hill Climbing
- Escaping Local Maxima: Simulated Annealing
- Local Beam Search
- Genetic Algorithm

Local search and optimization

- **Local search:**
  - Use single current state and move to neighboring states.
- **Idea: start with an initial guess at a solution and incrementally improve it until it is one**
- **Advantages:**
  - Use very little memory
  - Find often *reasonable* solutions in large or infinite state spaces.
- **Useful for pure optimization problems.**
  - Find or approximate best state according to some *objective function*
  - *Optimal if the space to be searched is convex*

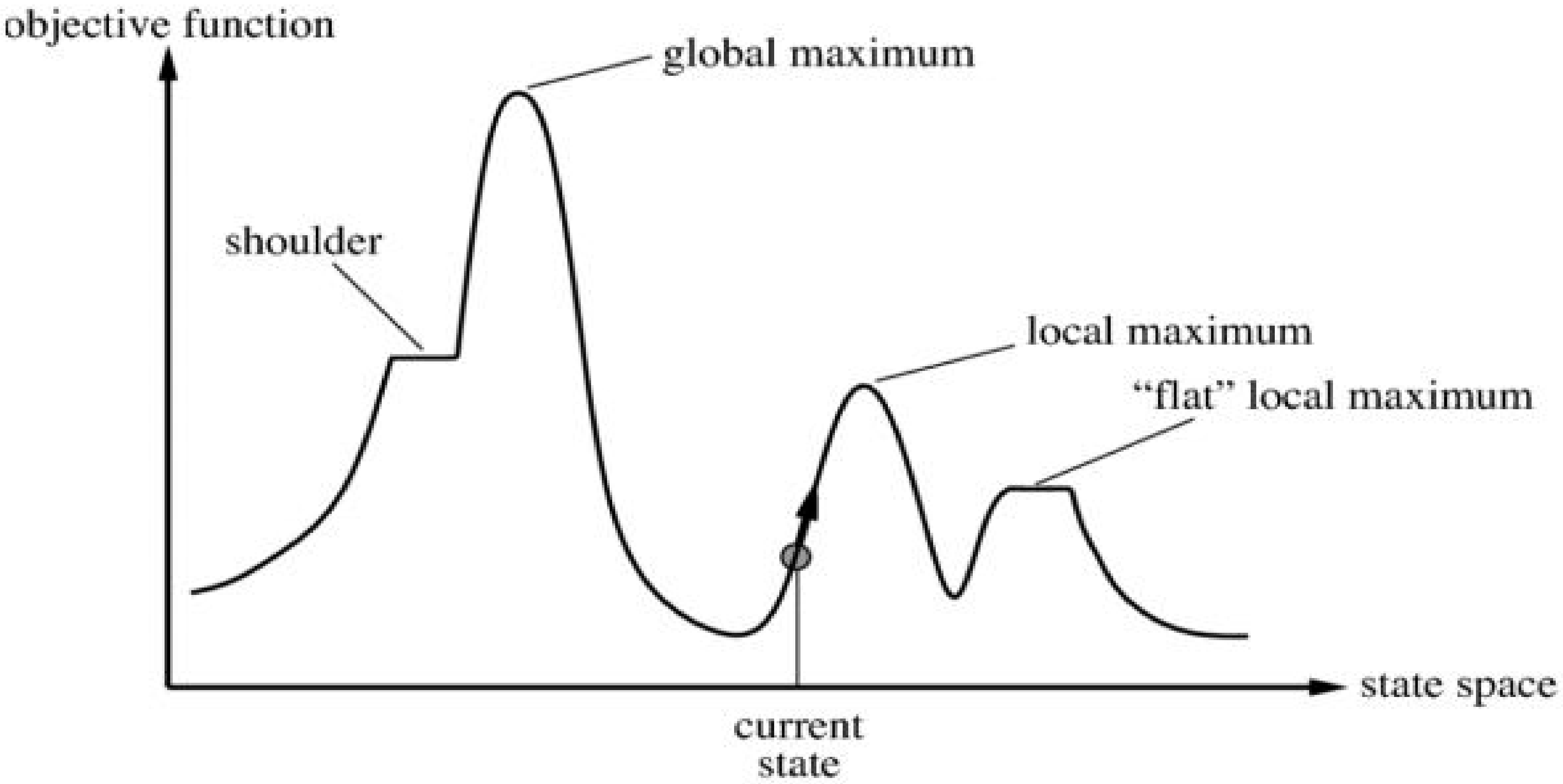
Hill-climbing search

I. **While (∃ uphill points):**
   - Move in the direction of increasing evaluation function ***f***

II. **Let** $s_{next} = \arg\max_{s} f(s)$ **,** s **a successor state to the current state n**
   - If $f(n) < f(s)$ then move to s
   - Otherwise halt at n

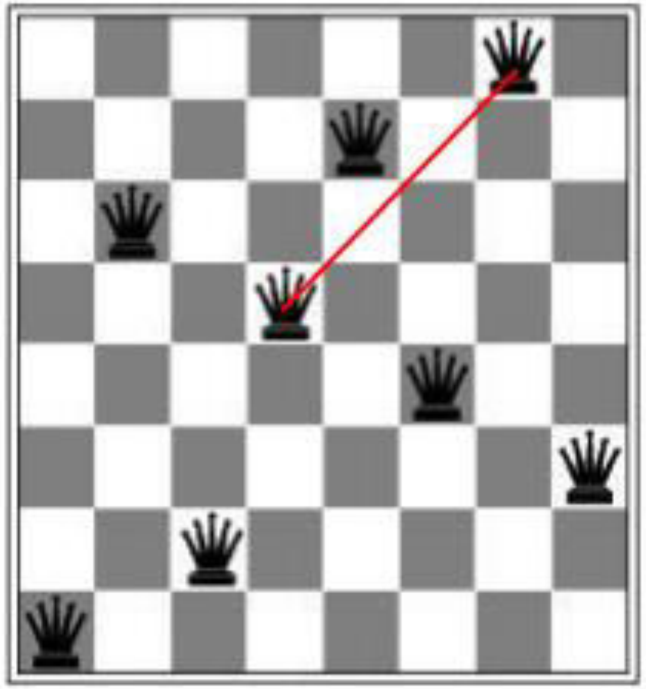
- **Properties:**
  - Terminates when a peak is reached.
  - Does not look ahead of the immediate neighbors of the current state.
  - Chooses randomly among the set of best successors, if there is more than one.
  - Doesn't *backtrack*, since it doesn't remember where it's been

Search Space features



Hill-climbing example: 8-queens

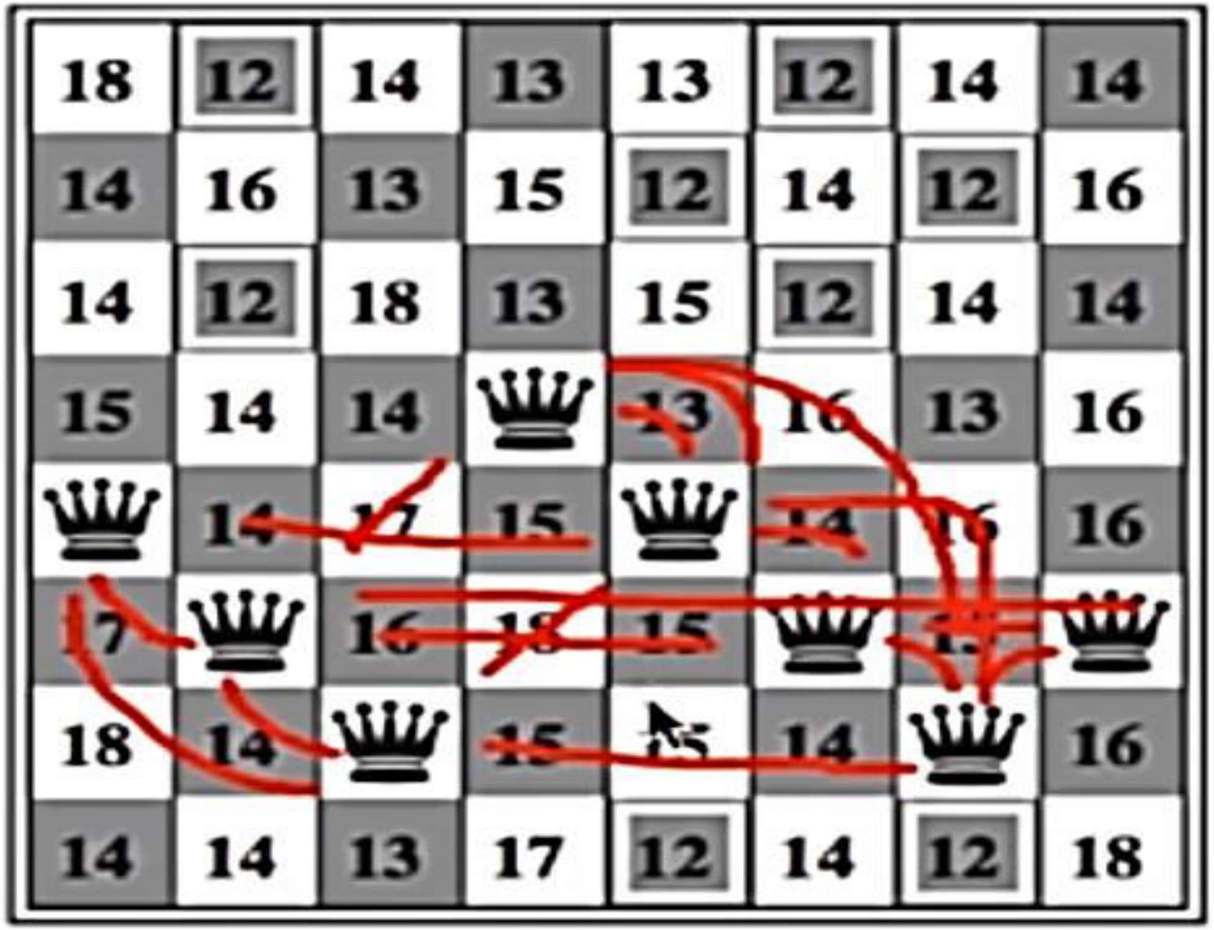
A state with *h*=17 and the *h*-value for each possible successor



A local minimum of *h* in the 8-queens state space (*h*=1).

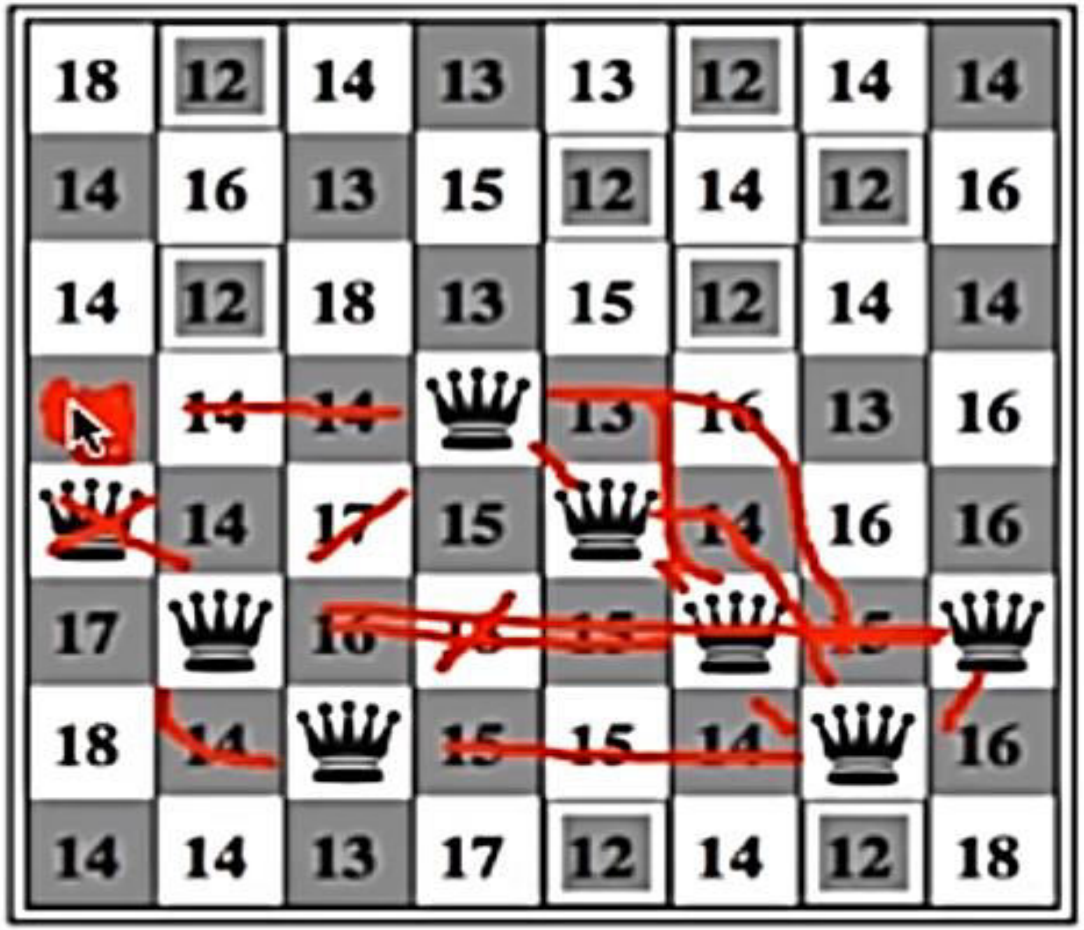
h = number of pairs of queens that are attacking each other

Hill Climbing 8 Queens Problem…..



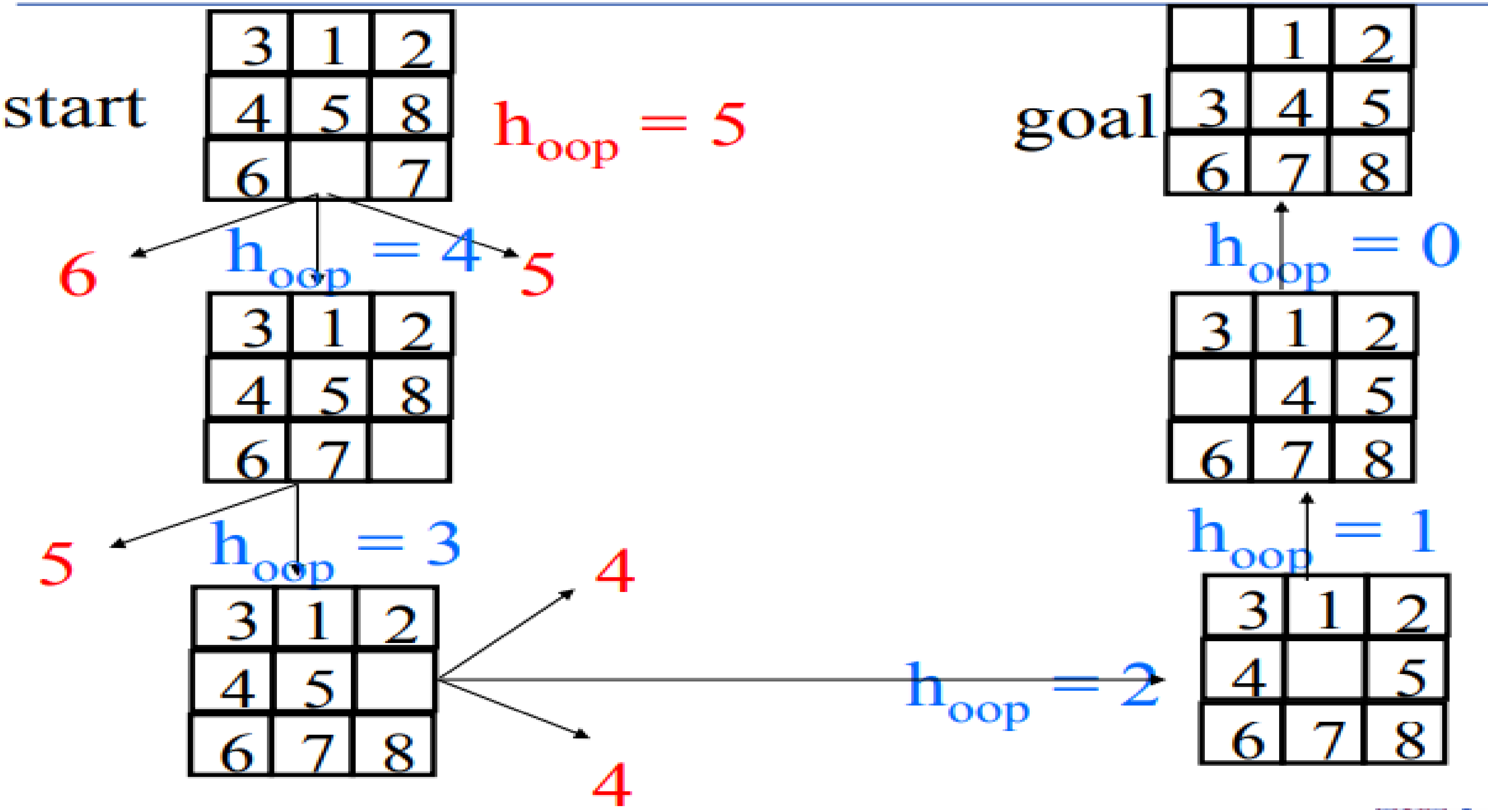
current $h = 17$. h for successors in each square.

Hill Climbing 8 Queens Problem…..



current *h* = 17. *h* for successors in each square.

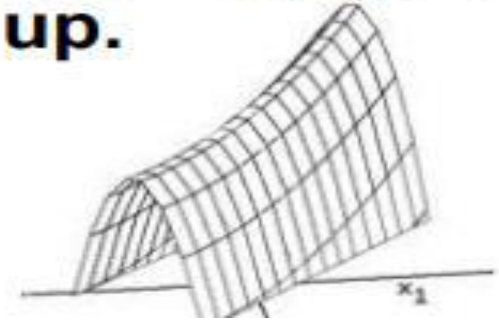
Hill climbing example I (*minimizing* h)



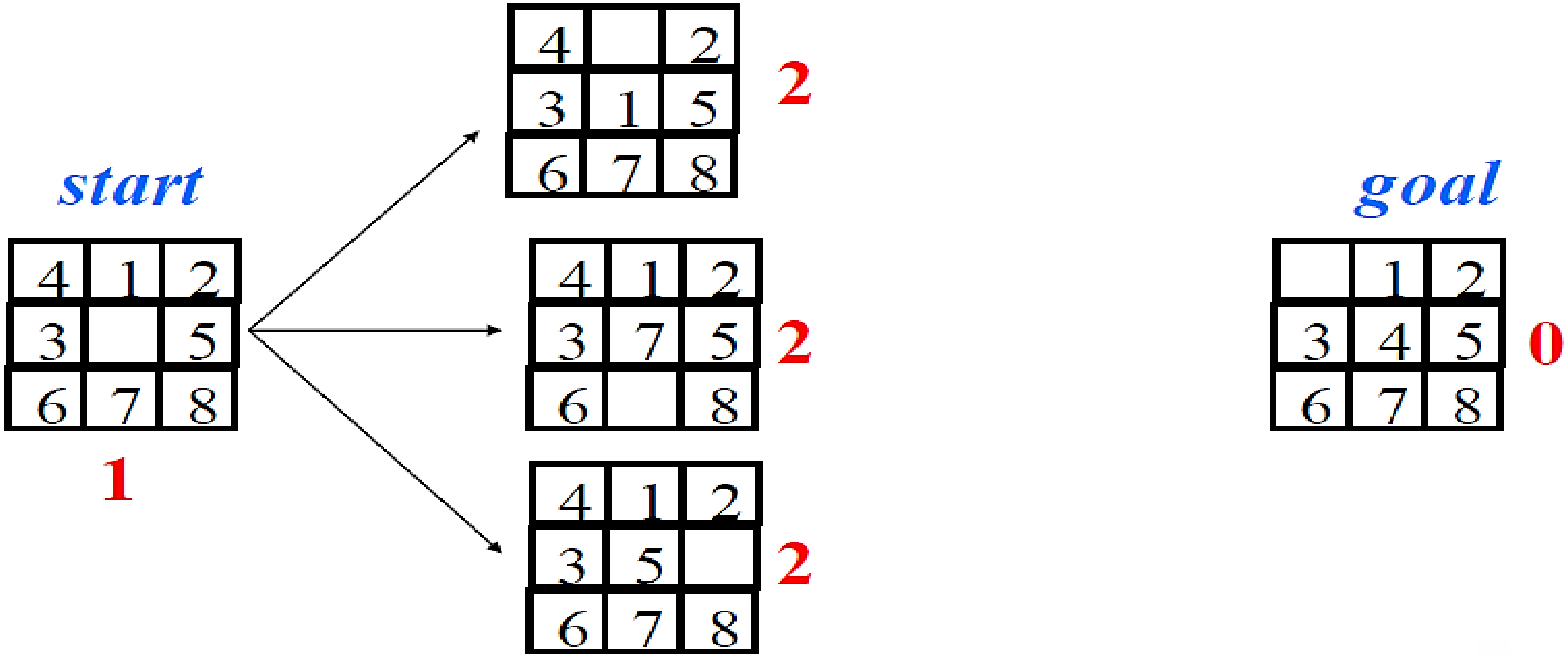
Drawbacks of hill climbing

- Local Maxima: peaks that aren't the highest point in the space
- Plateaus: the space has a broad flat region that gives the search algorithm no direction (random walk)

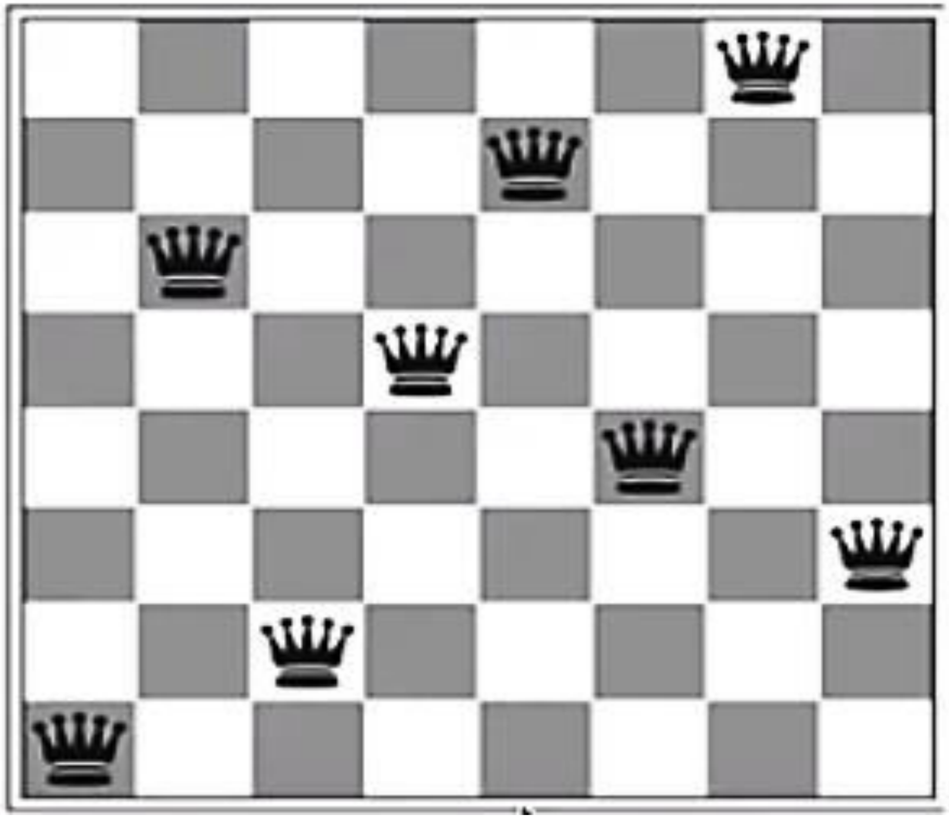
- Ridges: dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.



Toy Example of a local "maximum"



Hill Climbing 8 Queens Problem.....



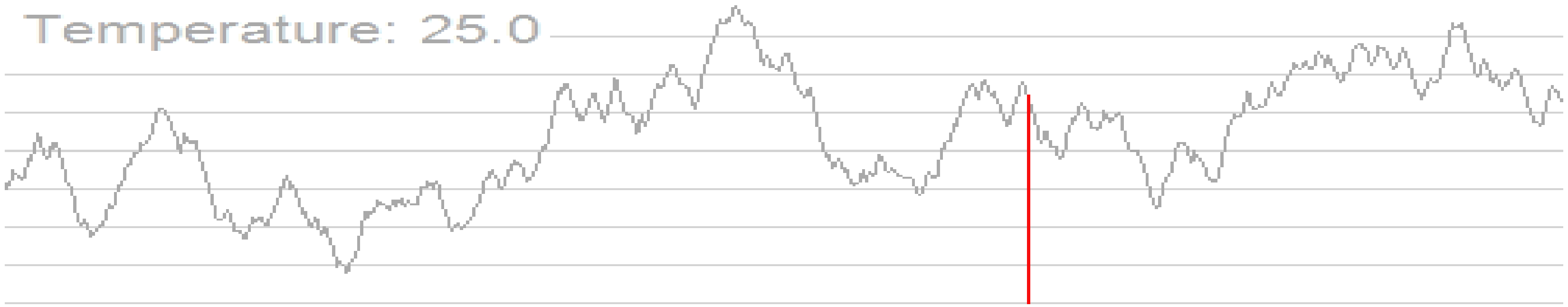
Strategies of Hill Climbing

- **Stocastic Hill Climbing**: Choose a random uphill move with certain prob.
- **First-Choice H.C.**: Generates successors until one is better than the current state[1]
- **Random-Restart H.C.**: Series of H.C. searches from random initial state.

Simulated Annealing

- *Annealing*: the process by which a metal cools and freezes into a minimum-energy crystalline structure (the annealing process)
- Conceptually SA exploits an analogy between annealing and the search for a minimum in a more general system.
  - Explore successors wildly randomly High Temp
  - As time goes by, explore less widly Cool down
  - Until there's a time where things settle. Cold

Simulated Annealing Example



AIMA Simulated Annealing Algorithm

function **SIMULATED-ANNEALING(** ***problem, schedule*****)** returns **a solution state**

input: ***problem*****, a problem**

schedule**, a mapping from time to "temperature"**

current $\leftarrow$ **MAKE-NODE(*****problem*****.INITIAL-STATE)**

for t $\leftarrow$ 1 to ∞ do

T $\leftarrow$ ***schedule*****(*****t*****)**

if ***T*** **=** ***0*** then return ***current***

next $\leftarrow$ **a randomly selected successor of** ***current***

Δ***E*** $\leftarrow$ ***next*****.VALUE –** ***current*****.VALUE**

if Δ***E*** **> 0** then ***current*** $\leftarrow$ ***next***

else ***current*** $\leftarrow$ ***next*** **only with probability** $e^{\Delta E/T}$

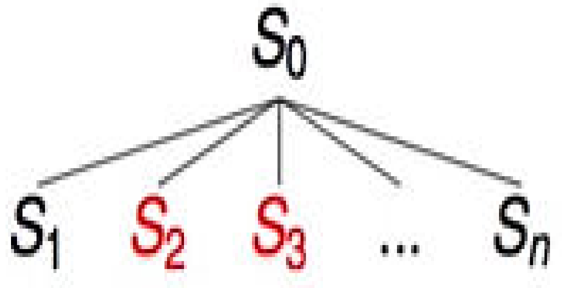
http://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/

Local Beam Search

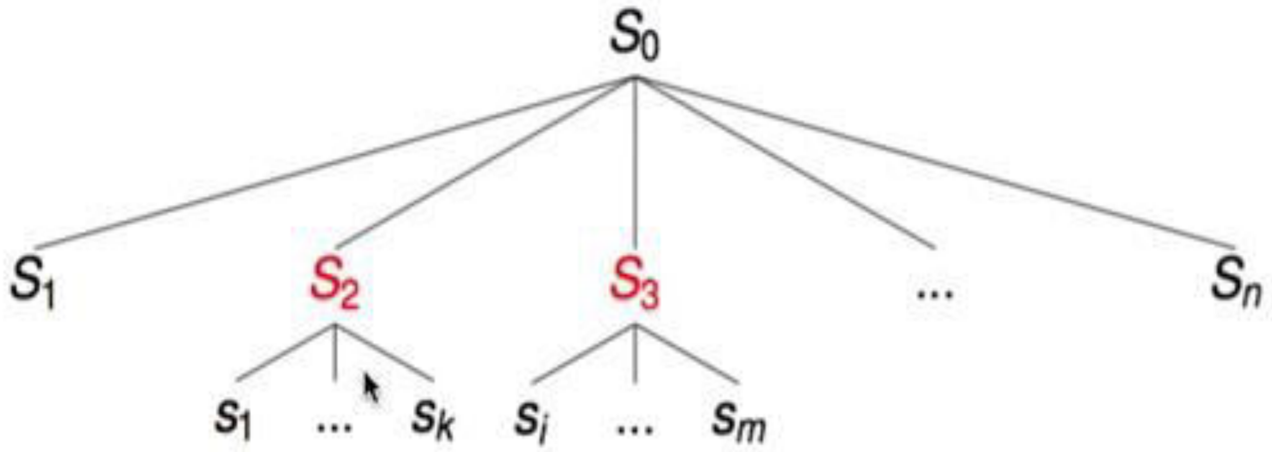
- **Keep track of *k* states instead of one**
  - Initially: *k* random states
  - Next: determine all successors of *k* states
  - If any of successors is goal $\rightarrow$ finished
  - Else select *k* best from successors and repeat.

- **Major difference with random-restart search**
  - Information is shared among *k* search threads.
- **Can suffer from lack of diversity.**
  - Stochastic variant: choose k successors proportionally to state success.

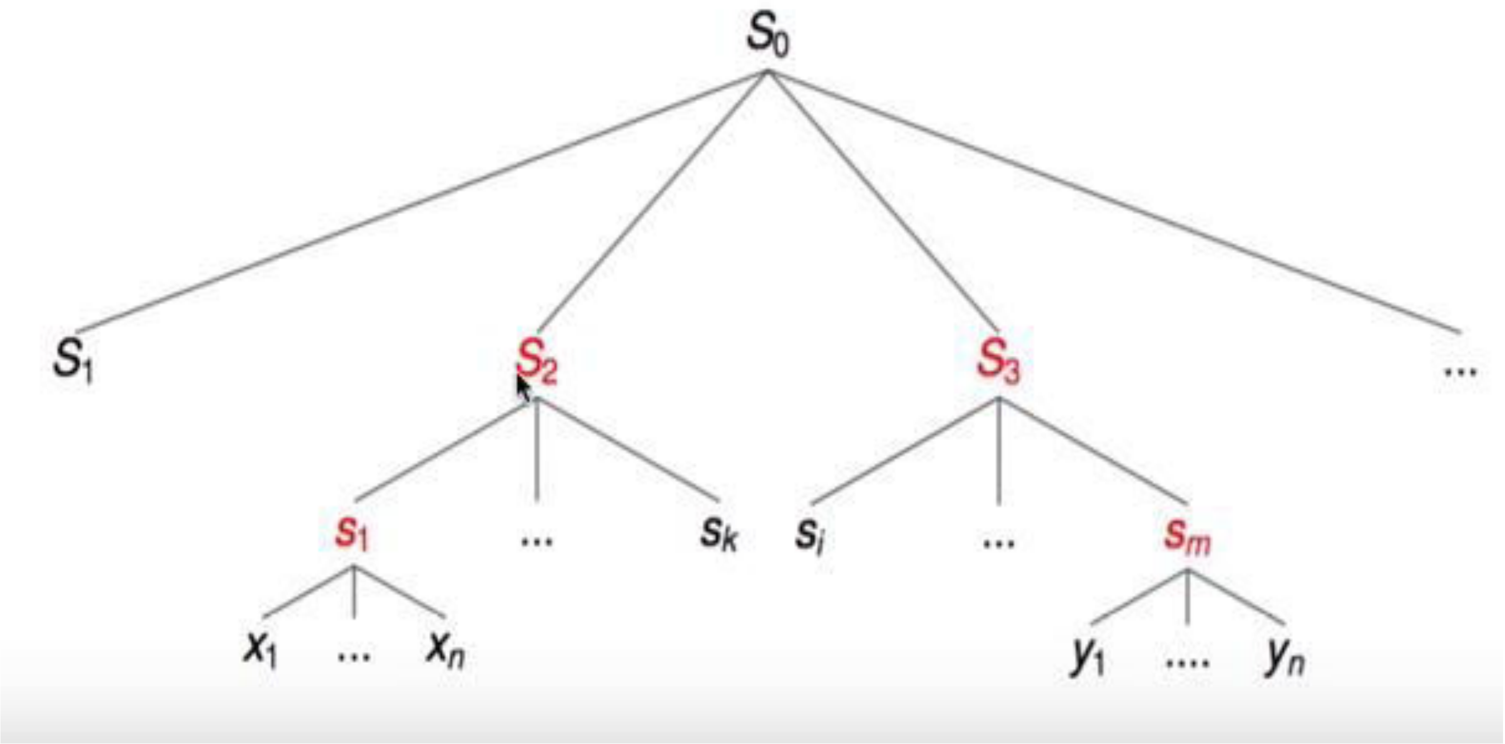
Local Beam Search



Local Beam Search



Local Beam Search

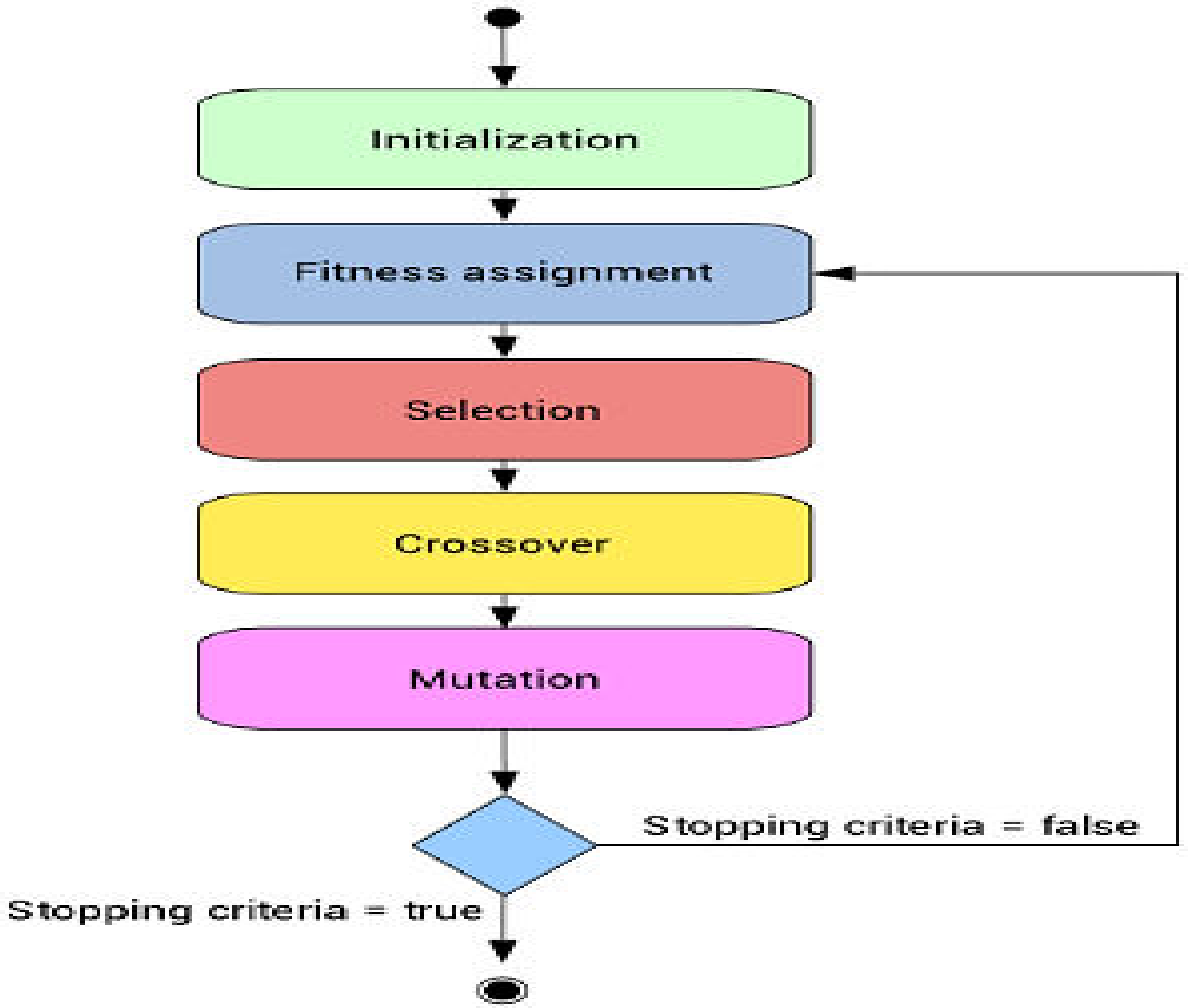


Genetic Algorithm

Introduced in the 1970s by John Holland at University of Michigan

- begin with *k* randomly generated states (population)
- each state (individual) is a string over some alphabet (chromosome)
- fitness function (bigger number is better)
- crossover
- mutate (evolve?)

Computational Model



Nature of Computer Mapping

Nature	Computer
Population	Set of solutions.
Individual	Solution to a problem.
Fitness	Quality of a solution.
Chromosome	Encoding for a Solution.
Gene	Part of the encoding of a solution.
Reproduction	Crossover

Encoding

The process of representing the solution in the form of a ***string*** *that conveys the necessary information.*

- **Binary Encoding** – Most common method of encoding. Chromosomes are strings of 1s and 0s and each position in the chromosome represents a particular characteristic of the problem.

  Permutation Encoding – Useful in ordering problems such as the Traveling Salesman Problem (TSP). Example. In TSP, every chromosome is a string of numbers, each of which represents a city to be visited.

- **Value Encoding** – Used in problems where complicated values, such as real numbers, are used and where binary encoding would not suffice.

Crossover

It is the process in which two chromosomes (strings) combine their genetic material (bits) to produce a new offspring which possesses both their characteristics.

- *Two strings are picked from the mating pool at random to cross over.*
- *The method chosen depends on the Encoding Method.*

Crossover

- **Single Point Crossover-** A random point is chosen on the individual chromosomes (strings) and the genetic material is exchanged at this point.

Chromosome1	11011 | 00100110110
Chromosome 2	11011 | 11000011110
Offspring 1	11011 | 11000011110
Offspring 2	11011 | 00100110110

Crossover

- **Two-Point Crossover-** Two random points are chosen on the individual chromosomes (strings) and the genetic material is exchanged at these points.

Chromosome1	11011 | 00100 | 110110
Chromosome 2	10101 | 11000 | 011110
Offspring 1	10101 | 00100 | 011110
Offspring 2	11011 | 11000 | 110110

NOTE: These chromosomes are different from the last example.

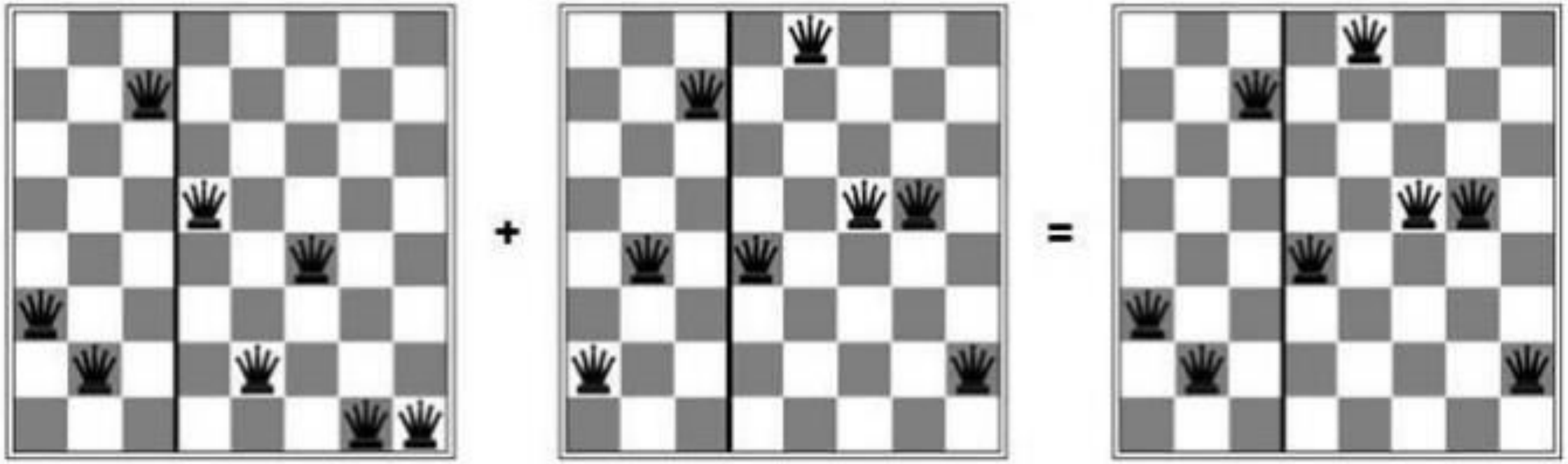
Crossover

- **Uniform Crossover-** Each gene (bit) is selected randomly from one of the corresponding genes of the parent chromosomes.

Chromosome1	11011 | 00100 | 110110
Chromosome 2	10101 | 11000 | 011110
Offspring	10111 | 00000 | 110110

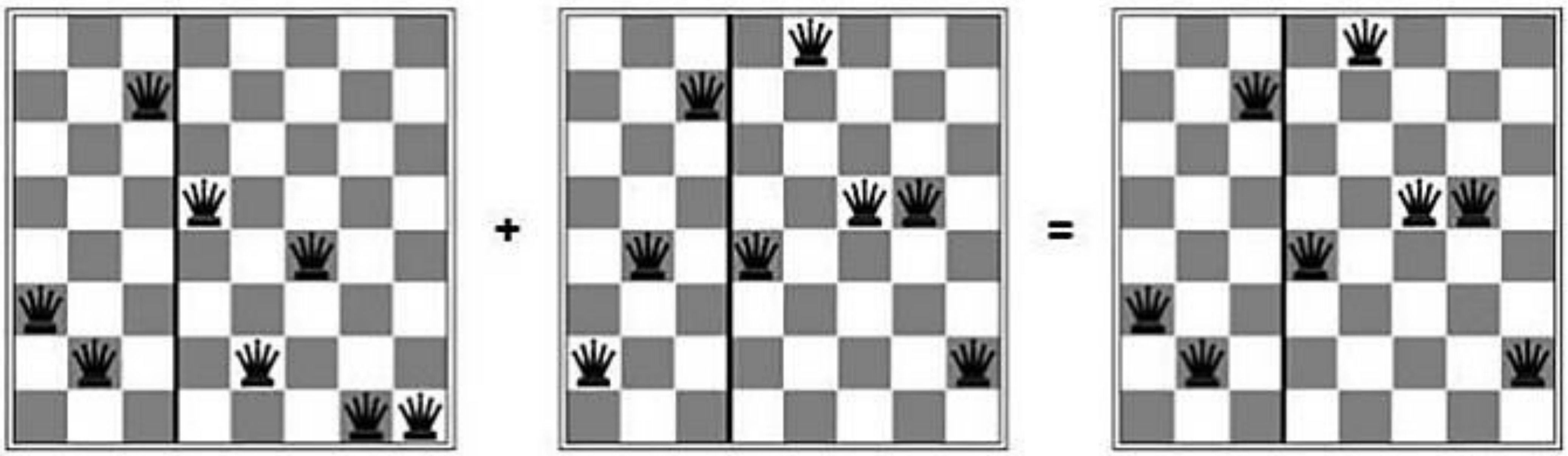
NOTE: Uniform Crossover yields ONLY 1 offspring.

Genetic algorithms:8-queens



Genetic Algorithm

- Fitness function= Pair of non-attacking queens
- That way higher scores are better



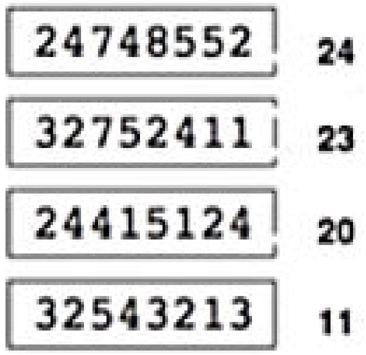
23 fitness

24748552

string

Fitness function

Represent states and compute fitness function.



(a)
Initial Population

Probability= 24+23+20+11= 78

Probability

Compute probability of being chosen (from fitness function).

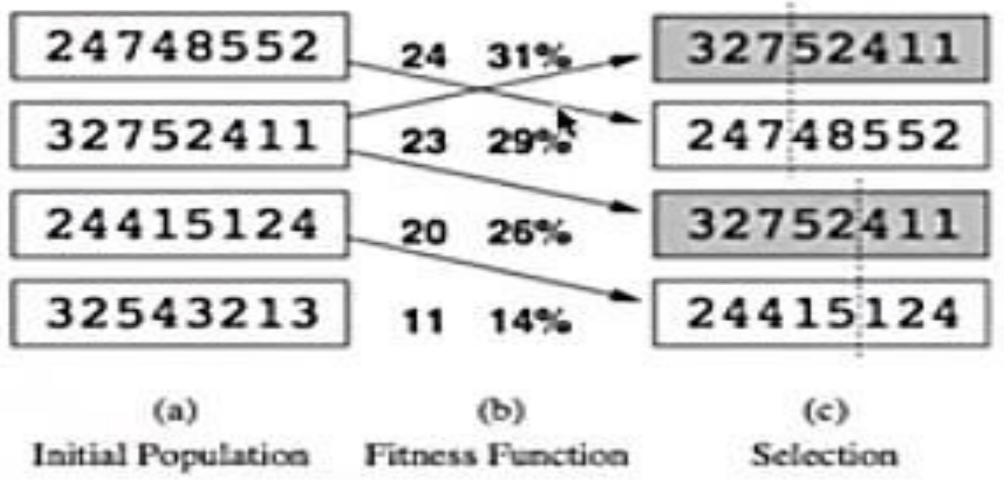
24748552	24	31%
32752411	23	29%
24415124	20	26%
32543213	11	14%

(a)
Initial Population

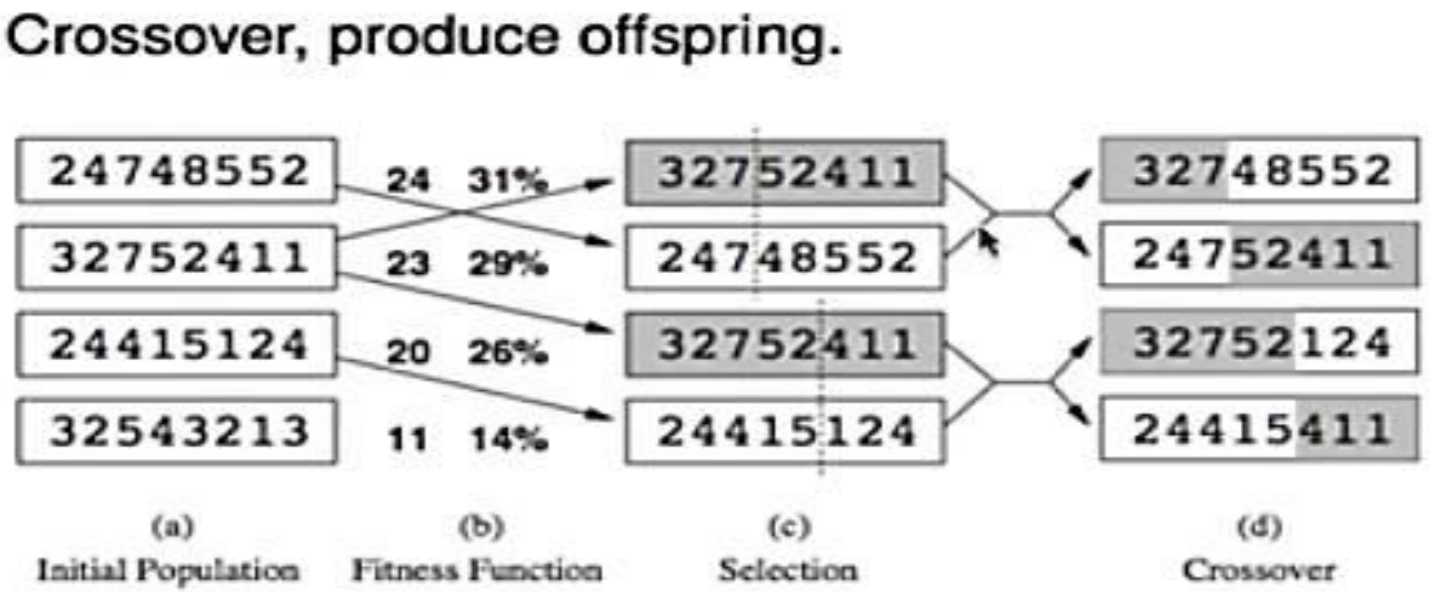
24/78= 0.307 Normalize $0.307 \times 100 = 30.7\%$ chance of being chosen probably

Reproduction

Randomly choose two pairs to reproduce based on probabilities. Pick a crossover point per pair.



Crossover


Crossover, produce offspring.
24748552
32752411
24415124
32543213
24 31%
23 29%
20 26%
11 14%
32752411
24748552
32752411
24415124
32748552
24752411
32752124
24415411
(a)
Initial Population
(b)
Fitness Function
(c)
Selection
(d)
Crossover

Mutation

May mutate.

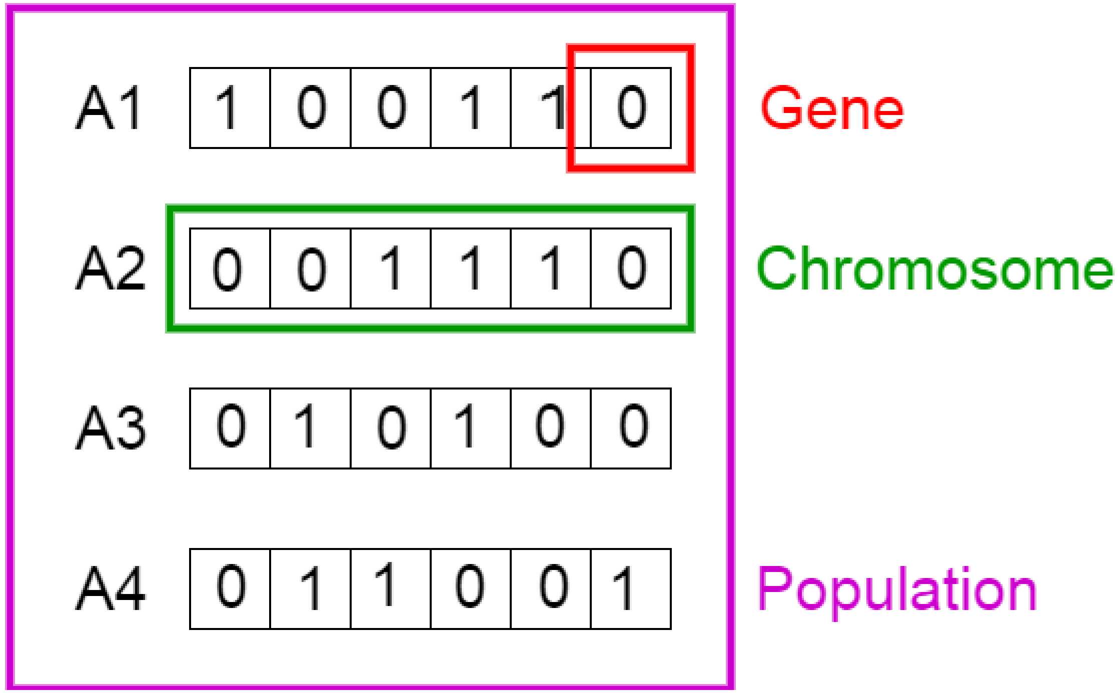
Knapsack Problem

- Let’s say, you are going to spend a month in the wilderness. Only thing you are carrying is the backpack which can hold a maximum weight of **30 kg**. Now you have different survival items, each having its own “Survival Points” (which are given for each item in the table). So, your objective is maximize the survival points.
- Here is the table giving details about each item.

ITEM	WEIGHT	SURVIVAL POINTS
SLEEPING BAG	15	15
ROPE	3	7
POCKET KNIFE	2	10
TORCH	5	5
BOTTLE	9	8
GLUCOSE	20	17

Initialization

- We know that, chromosomes are binary strings, where for this problem 1 would mean that the following item is taken and 0 meaning that it is dropped



Fitness Function

- We will calculate fitness points for our first two chromosomes.
- For A1 chromosome [100110], A2 chromosome [001110],

ITEMS	WEIGHT	SURVIVAL POINTS
Sleeping bag	15	15
Torch	5	5
Bottle	9	8
TOTAL	29	28

ITEMS	WEIGHT	SURVIVAL POINTS
Pocket Knife	2	10
Torch	5	5
Bottle	9	8
TOTAL	16	23

Therefore chromosome 1 is more fit than chromosome 2.

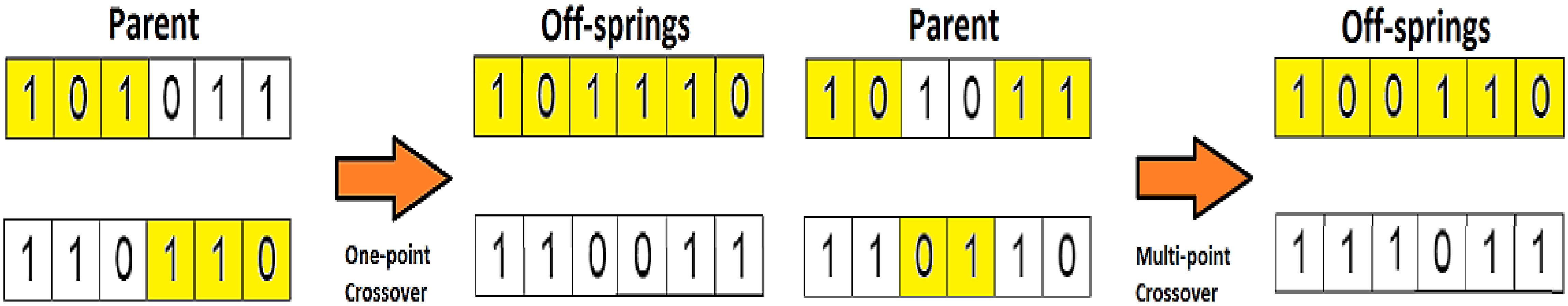
Selection

- Now, we can select fit chromosomes from our population which can mate and create their off-springs.

	Survival Points	Percentage
Chromosome 1	28	28.9%
Chromosome 2	23	23.7%
Chromosome 3	12	12.4%
Chromosome 4	34	35.1%

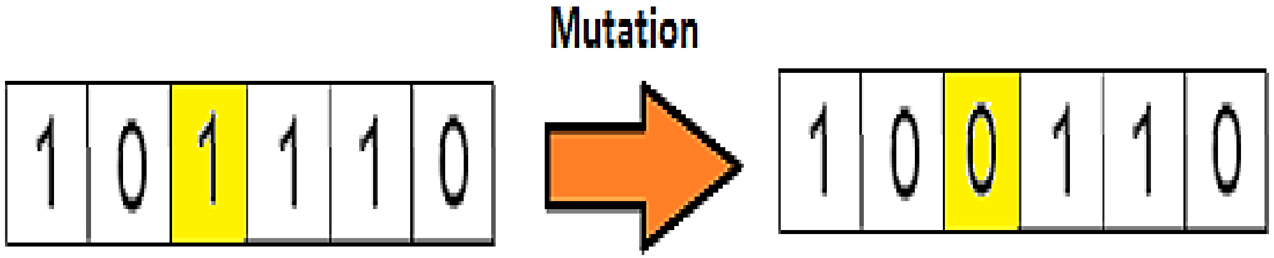
Cross Over

- So now we find the crossover of chromosome 1 and 4, which were selected in the previous step. Take a look at the image below.



Mutation

- A random tweak in the chromosome

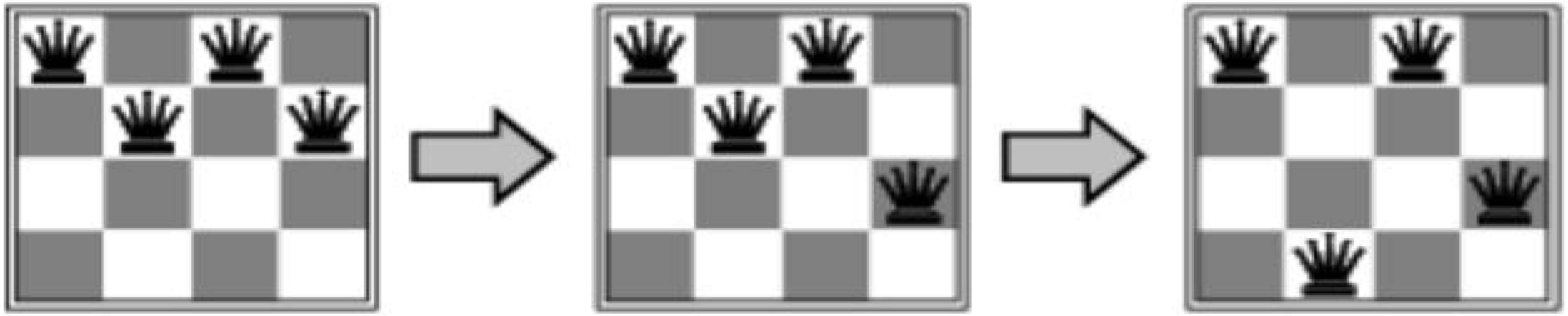


Properties

- Work well for mixed (continuous and discrete) problems
- They are less suceptible to get stuck at local optima
- Computationally expensive
- However, easy to perform in parallel
- No math in the process. The objective (fitness) function may be hard

Hill-climbing Example: *n*-queens

- *n*-queens problem: Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- *Good heuristic:* *h* = number of pairs of queens that are attacking each other



h=5 *h=3* *h=1*

(for illustration)