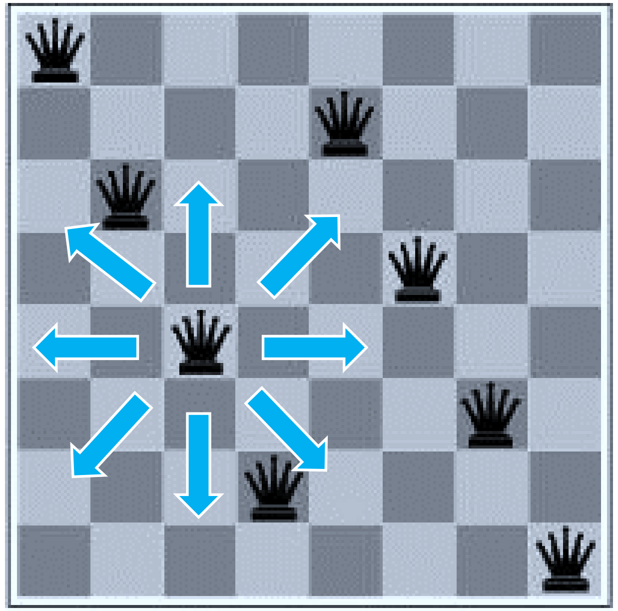
Local Search

LOCAL SEARCH AND OPTIMIZATION

- Previous methods: path to goal is a solution to a problem
  - systematic exploration of search space.
- Today approaches: a state is a solution to problem
  - for some problems path is irrelevant.
  - E.g., 8-queens
- Different algorithms can be used
  - Depth First Branch and Bound
  - Local search




Goal Satisfaction
Optimization
Reach the goal node
Optimize(objective)
Constraint satisfaction
Constraint Optimization

LOCAL SEARCH AND OPTIMIZATION

Local search

Keep track of single current state
Move only to neighboring states
Ignore paths

Advantages:

Use very little memory
Can often find reasonable solutions in large or infinite (continuous) state spaces.

"Pure optimization" problems

- All states have an objective function
- Goal is to find state with max (or min) objective value
- Does not quite fit into path-cost/goal-state formulation
- Local search can do quite well on these problems.

HILL CLIMBING

"a loop that continuously moves towards increasing value"

- terminates when a peak is reached
- Aka greedy local search

Value can be either

- Objective function value
- Heuristic function value (minimized)

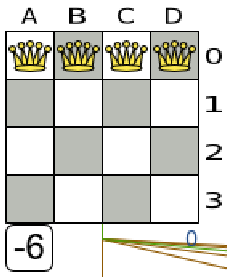
Hill climbing does not look ahead of the immediate neighbors

Can randomly choose among the set of best successors

- if multiple have the best value

"climbing Mount Everest in a thick fog with amnesia"

HILL CLIMBING



It starts from an initial solution and evolves that single solution into a mostly better and better solution.

Example n-queens (here n=4)

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

Successor function

move a single queen to another square in the same column.

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

- h = 6 for initial state

Image from https://docs.optaplanner.org/6.3.0.Beta1/optaplanner-docs/html/ch10.html

HILL CLIMBING

It starts from an initial solution and evolves that single solution into a mostly better and better solution.

Example n-queens (here n=4)

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

Successor function

move a single queen to another square in the same column.

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

- $h = 6$ for initial state

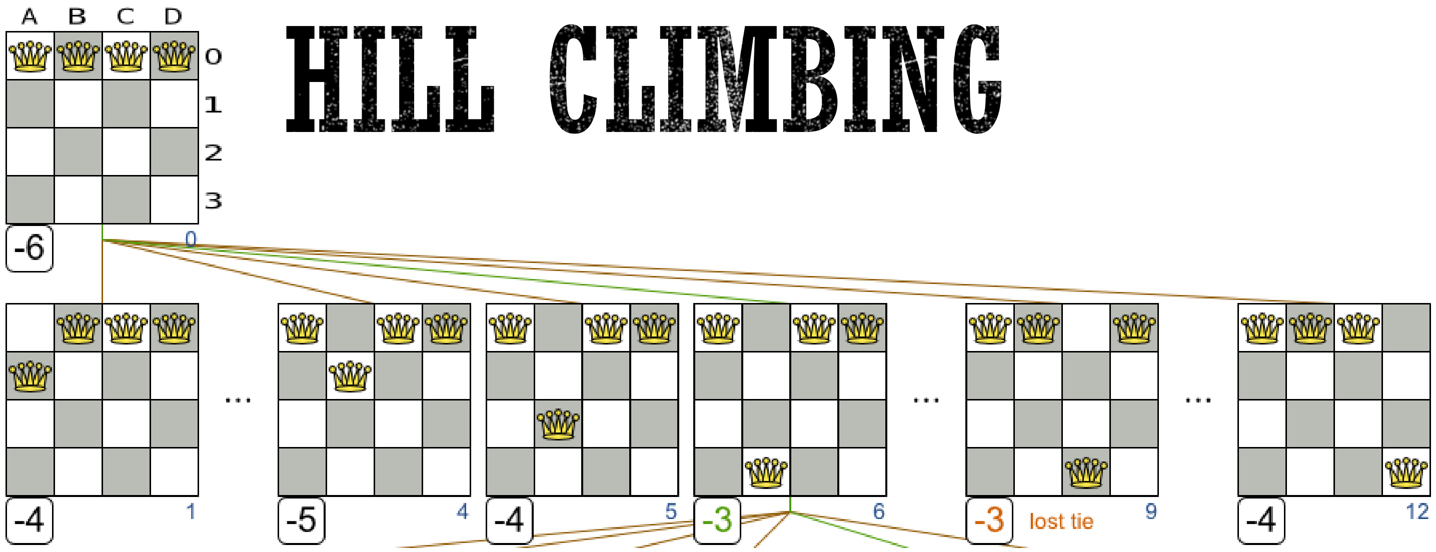
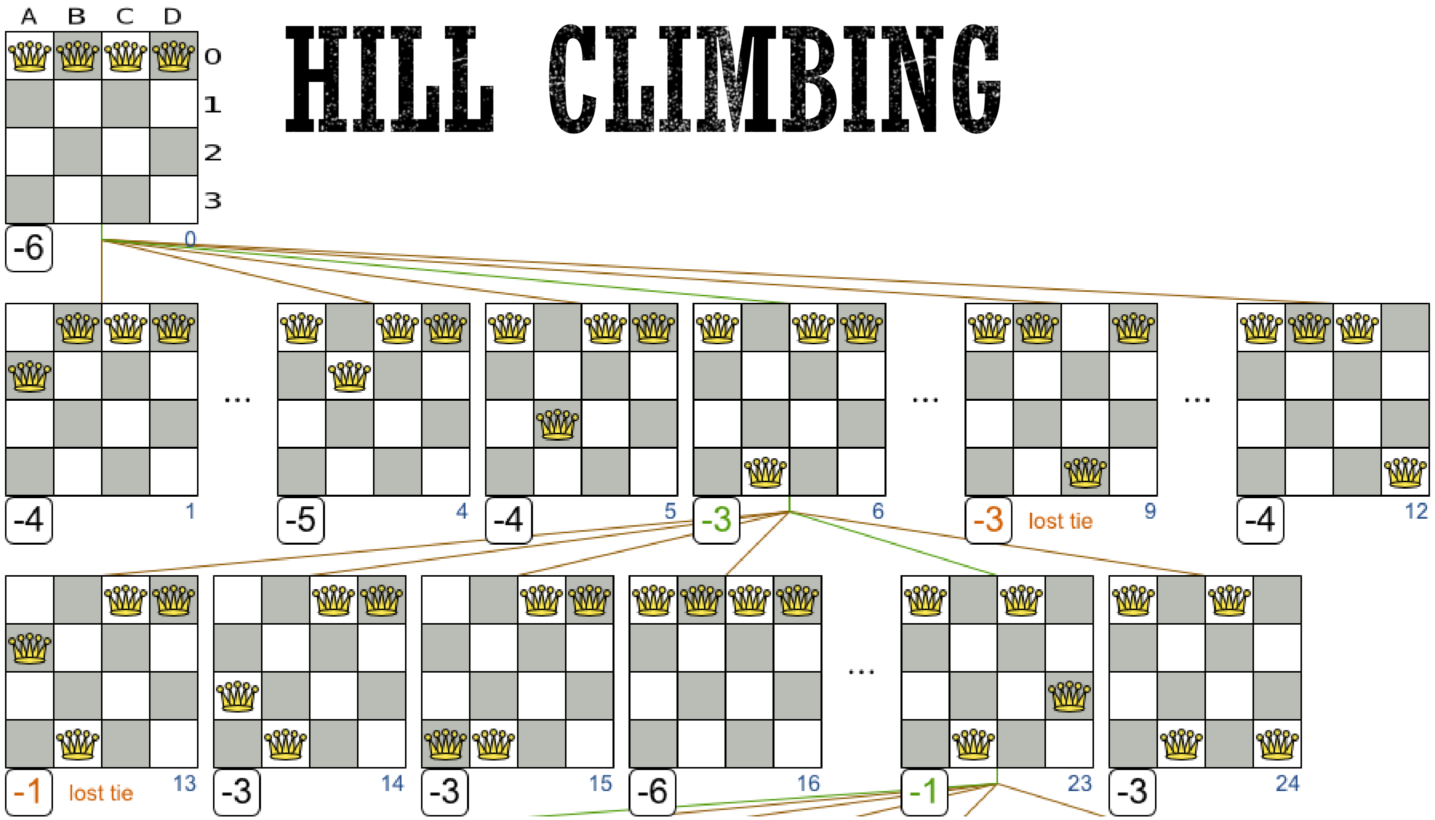


Image from https://docs.optaplanner.org/6.3.0.Beta1/optaplanner-docs/html/ch10.html

HILL CLIMBING



It starts from an initial solution and evolves that single solution into a mostly better and better solution.

Example n-queens (here n=4)

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

Successor function

move a single queen to another square in the same column.

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

- $h = 6$ for initial state

Image from https://docs.optaplanner.org/6.3.0.Beta1/optaplanner-docs/html/ch10.html

HILL CLIMBING

It starts from an initial solution and evolves that single solution into a mostly better and better solution.

Example n-queens (here n=4)

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

Successor function

move a single queen to another square in the same column.

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

- $h = 6$ for initial state

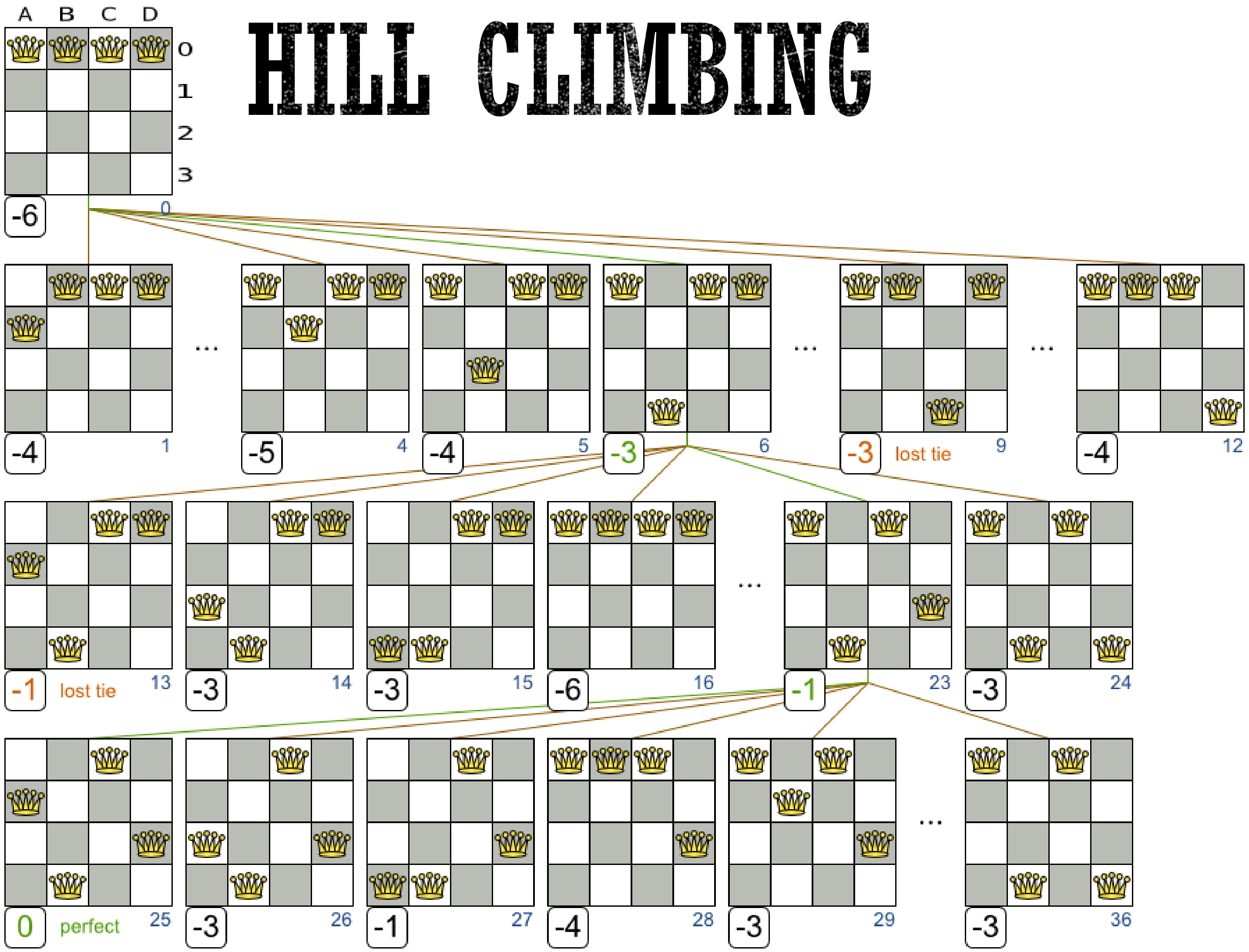


Image from https://docs.optaplanner.org/6.3.0.Beta1/optaplanner-docs/html/ch10.html

HILL CLIMBING

function HILL-CLIMBING(*problem*) **return** a state that is a local maximum

input: *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*

min version will reverse inequalities and look for lowest valued successor

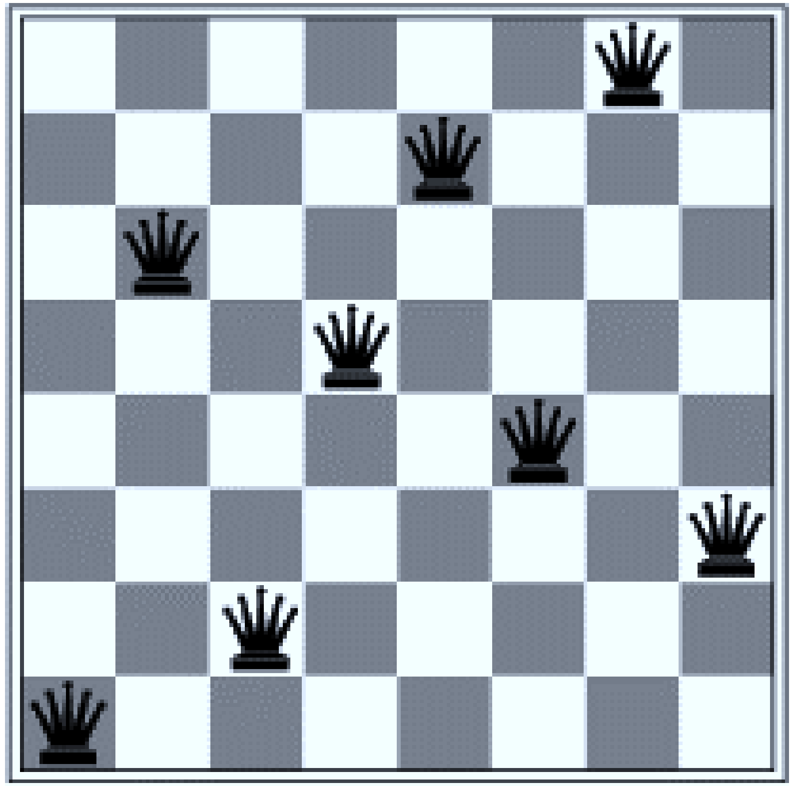
HILL CLIMBING

Example of 8 queens Problem

- Randomly generated 8-queens starting states…
- 14% the time it solves the problem
- 86% of the time it get stuck at a local minimum

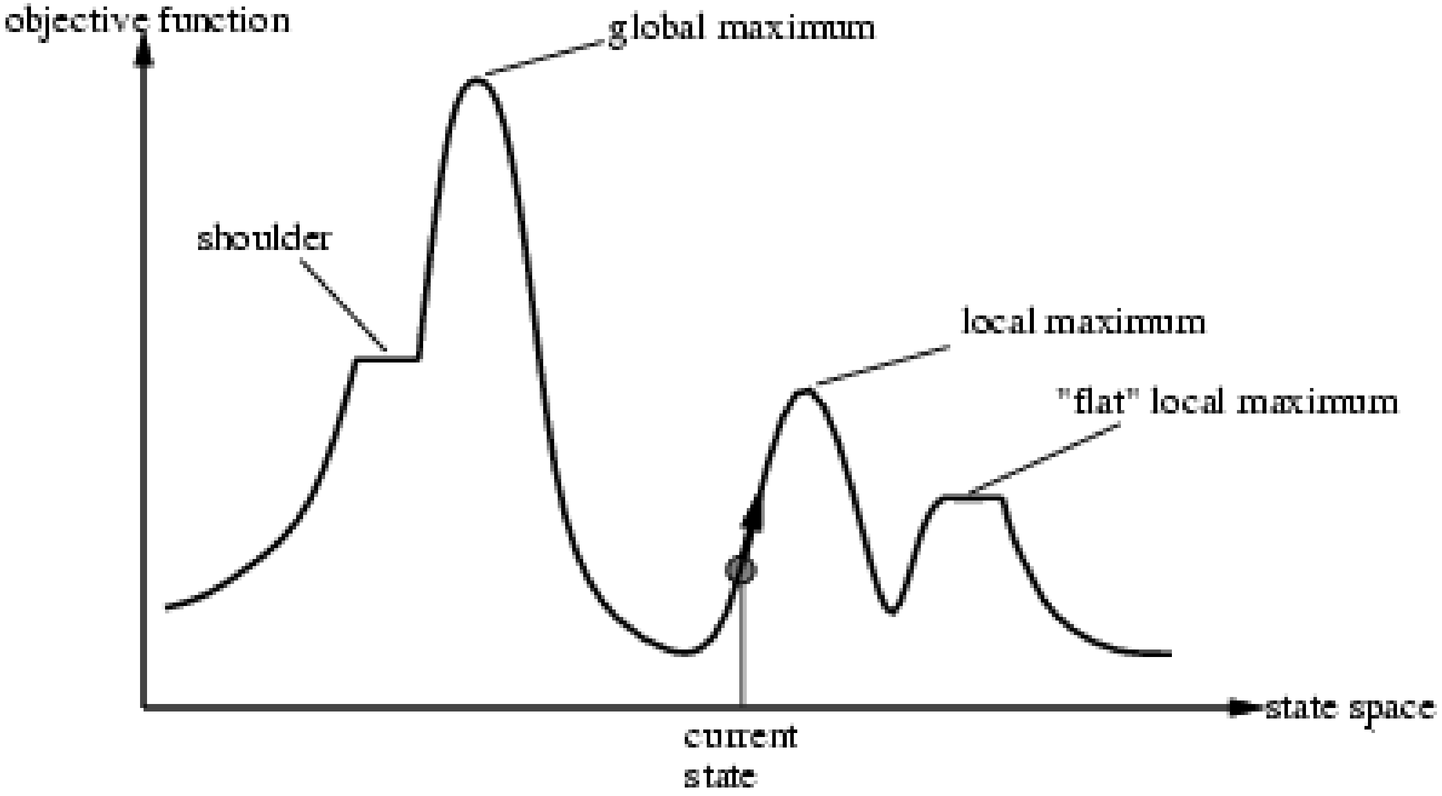
It takes only 4 steps on average when it succeeds

- And 3 on average when it gets stuck
- (for a state space with 8^8 =~17 million states)



Is this a solution?

HILL CLIMBING



HILL CLIMBING (LOCAL OPTIMA PROBLEM)

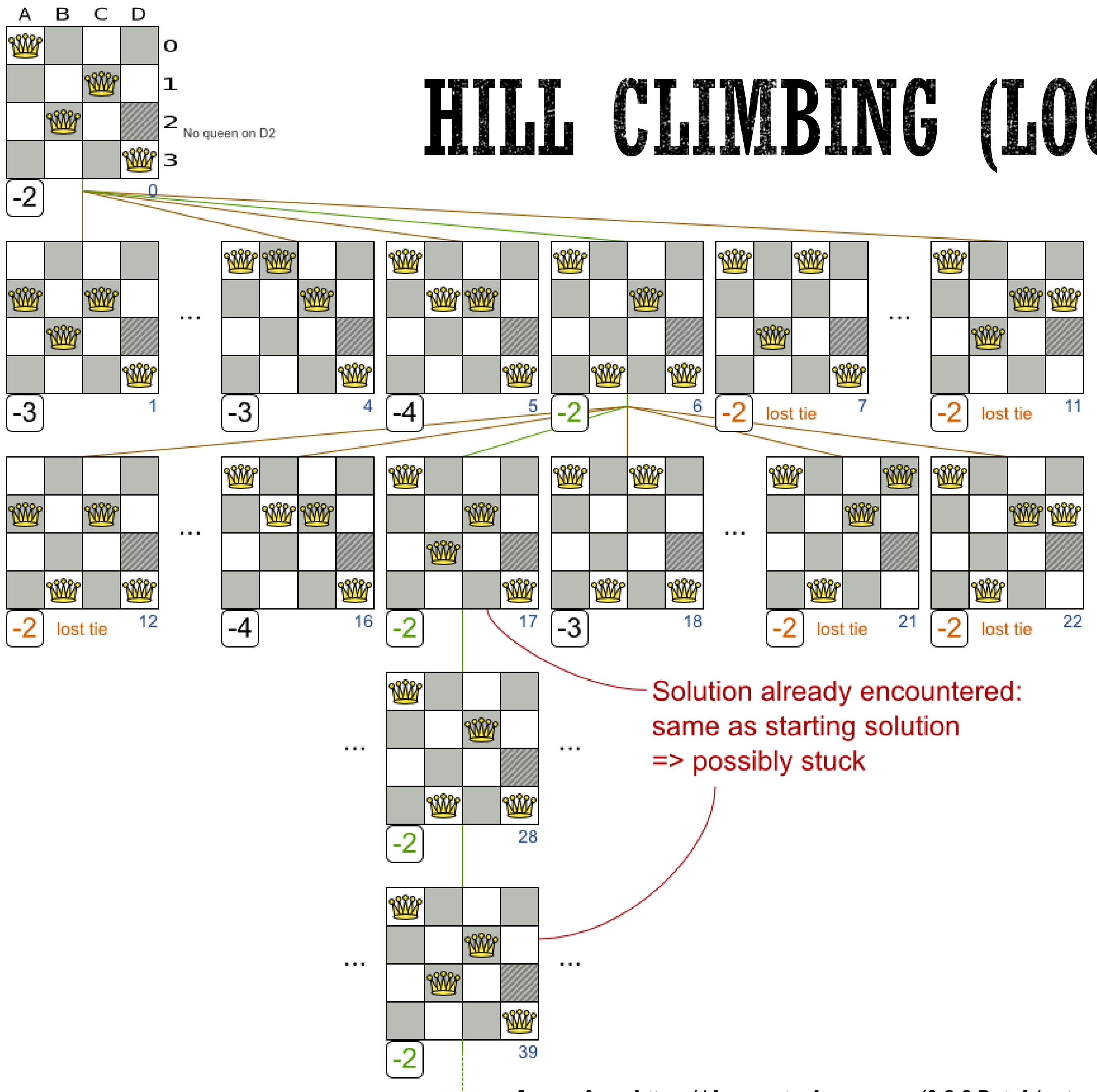


Image from https://docs.optaplanner.org/6.3.0.Beta1/optaplanner-docs/html/ch10.html

HILL-CLIMBING: STOCHASTIC VARIATIONS

Stochastic hill-climbing

- Random selection among the uphill moves.
- The selection probability can vary with the steepness of the uphill move.

To avoid getting stuck in local minima

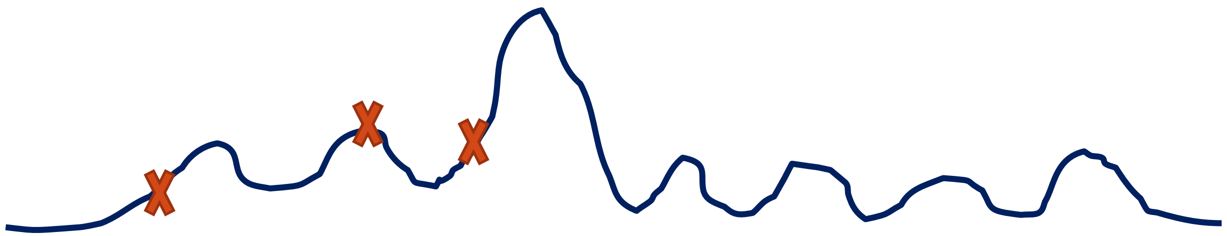
- Random-walk hill-climbing
- Random-restart hill-climbing
- Hill-climbing with both

HILL-CLIMBING WITH RANDOM RESTARTS

If at first you don't succeed, try, try again!

- Different variations
  - For each restart: run until termination vs. run for a fixed time
  - Run a fixed number of restarts or run indefinitely

***** an Interesting Approach *****



SIMULATED ANNEALING

Simulated Annealing = physics inspired twist on random walk

- Basic ideas:
  - like hill-climbing identify the quality of the local improvements
  - instead of picking the best move, pick one randomly
  - say the change in objective function is δ
  - if δ is positive, then move to that state
  - otherwise:
    - move to this state with probability proportional to δ
    - thus: worse moves (very large negative δ) are executed less often
  - however, there is always a chance of escaping from local maxima
  - over time, make it less likely to accept locally bad moves
  - (Can also make the size of the move random as well, i.e., allow "large" steps in state space)

PHYSICAL INTERPRETATION OF SIMULATED ANNEALING

- A Physical Analogy:
  - imagine letting a ball roll downhill on the function surface
    - this is like hill-climbing (for minimization)
  - now imagine shaking the surface, while the ball rolls, gradually reducing the amount of shaking
    - this is like simulated annealing

- Annealing = physical process of cooling a liquid or metal until particles achieve a certain frozen crystal state
  - simulated annealing:
    - free variables are like particles
    - seek "low energy" (high quality) configuration
    - slowly reducing temp. T with particles moving around randomly

Simulated Annealing
(Time Gradiant aware)

N queens (n = 4, startingTemperature = 2)

Temperature decreases for each step

Step 0

t
2.0

Δ	max
≥0	any
-1	0.61
-2	0.37
-3	0.22
-4	0.14

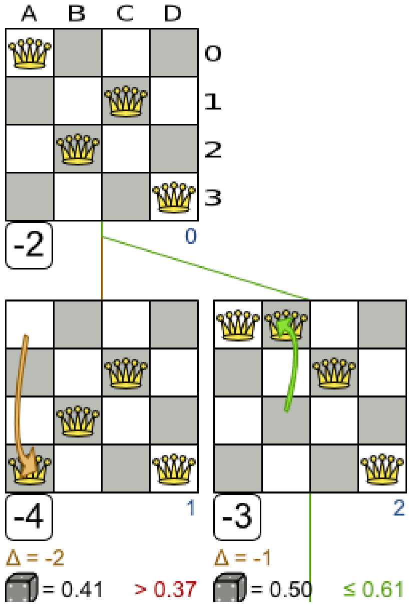


Image from https://docs.optaplanner.org/6.3.0.Beta1/optaplanner-docs/html/ch10.html

Simulated Annealing (Time Gradiant aware)

N queens (n = 4, startingTemperature = 2)

Temperature decreases for each step

Step 0

t
2.0

Δ	max
≥0	any
-1	0.61
-2	0.37
-3	0.22
-4	0.14

Step 1

t
1.6

Δ	max
≥0	any
-1	0.54
-2	0.29
-3	0.15
-4	0.08

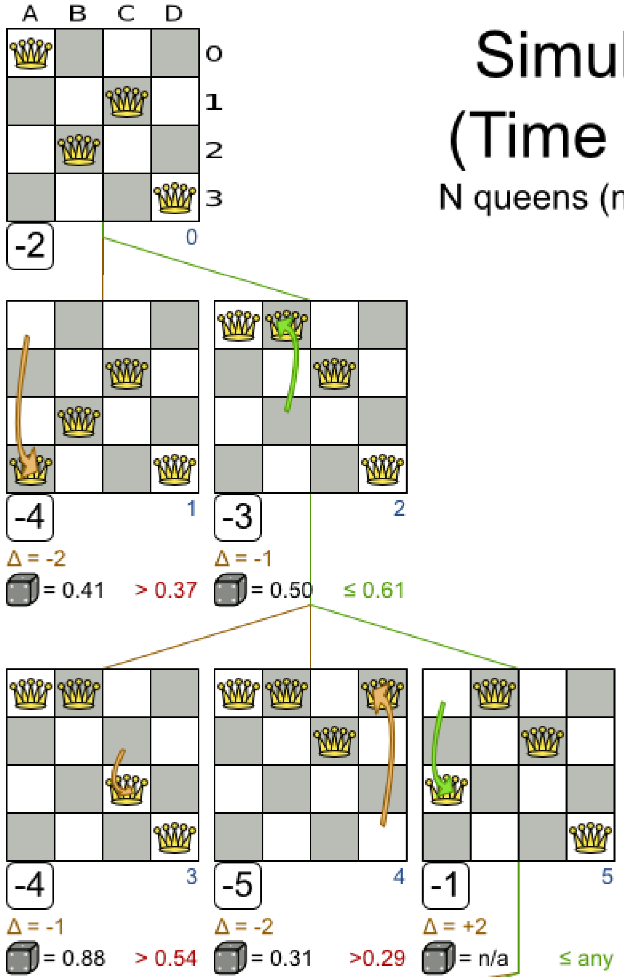


Image from https://docs.optaplanner.org/6.3.0.Beta1/optaplanner-docs/html/ch10.html

Simulated Annealing (Time Gradiant aware)

N queens (n = 4, startingTemperature = 2)

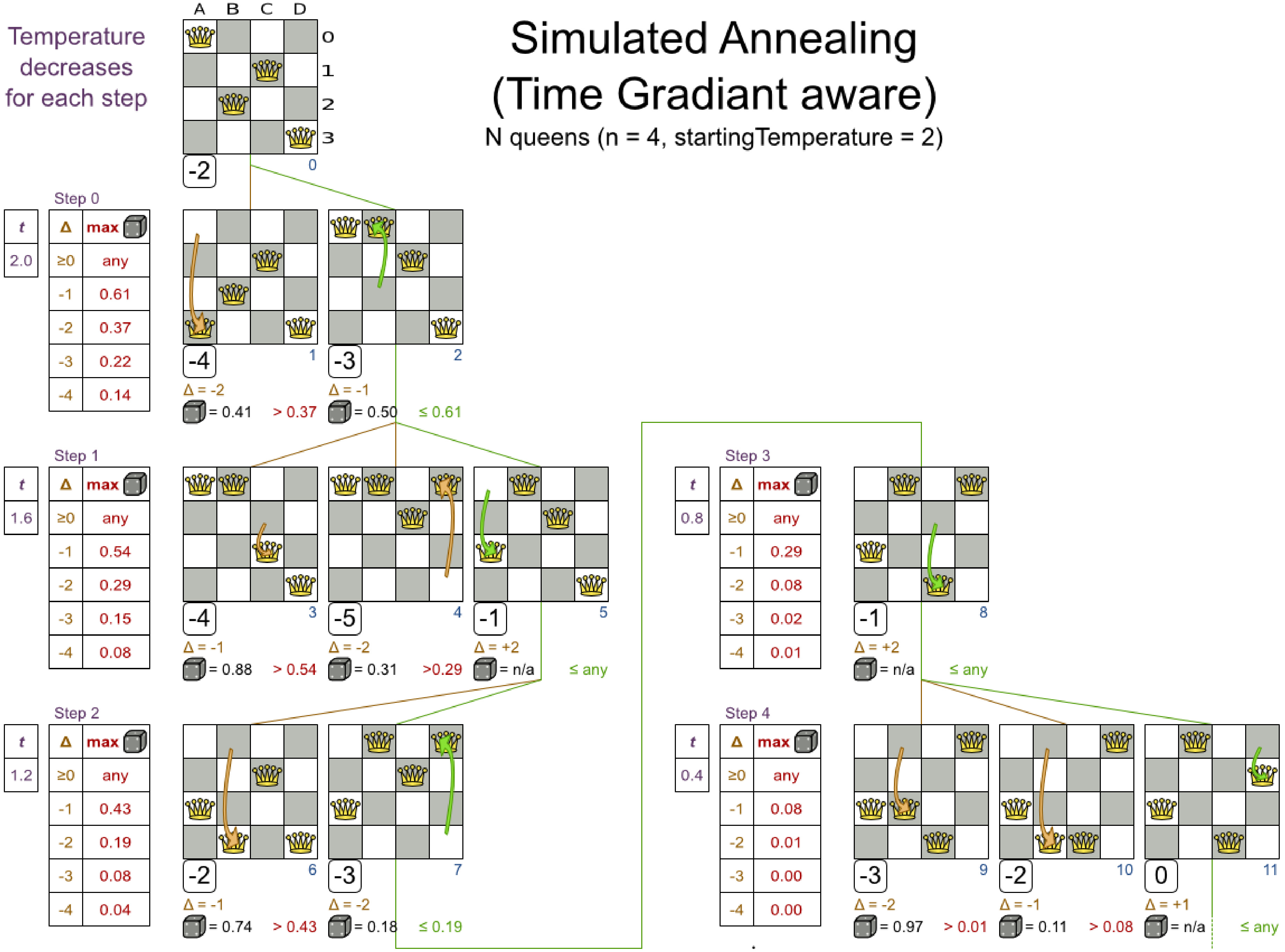


Image from https://docs.optaplanner.org/6.3.0.Beta1/optaplanner-docs/html/ch10.html

SIMULATED ANNEALING

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

input: *problem*, a problem

schedule, a mapping from time to temperature

local variables: *current*, a node.

next, a node.

T, a "temperature" controlling the prob. of downward steps

current ← MAKE-NODE(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* only with probability $e^{\Delta E/T}$

high T: probability of "locally bad" move is higher

low T: probability of "locally bad" move is lower

typically, T is decreased as the algorithm runs longer

i.e., there is a "temperature schedule"

GENETIC ALGORITHMS

Twist on Local Search: successor is generated by combining two parent states

A state is represented as a string over a finite alphabet (e.g. binary)

- 8-queens
  - State = position of 8 queens each in a column

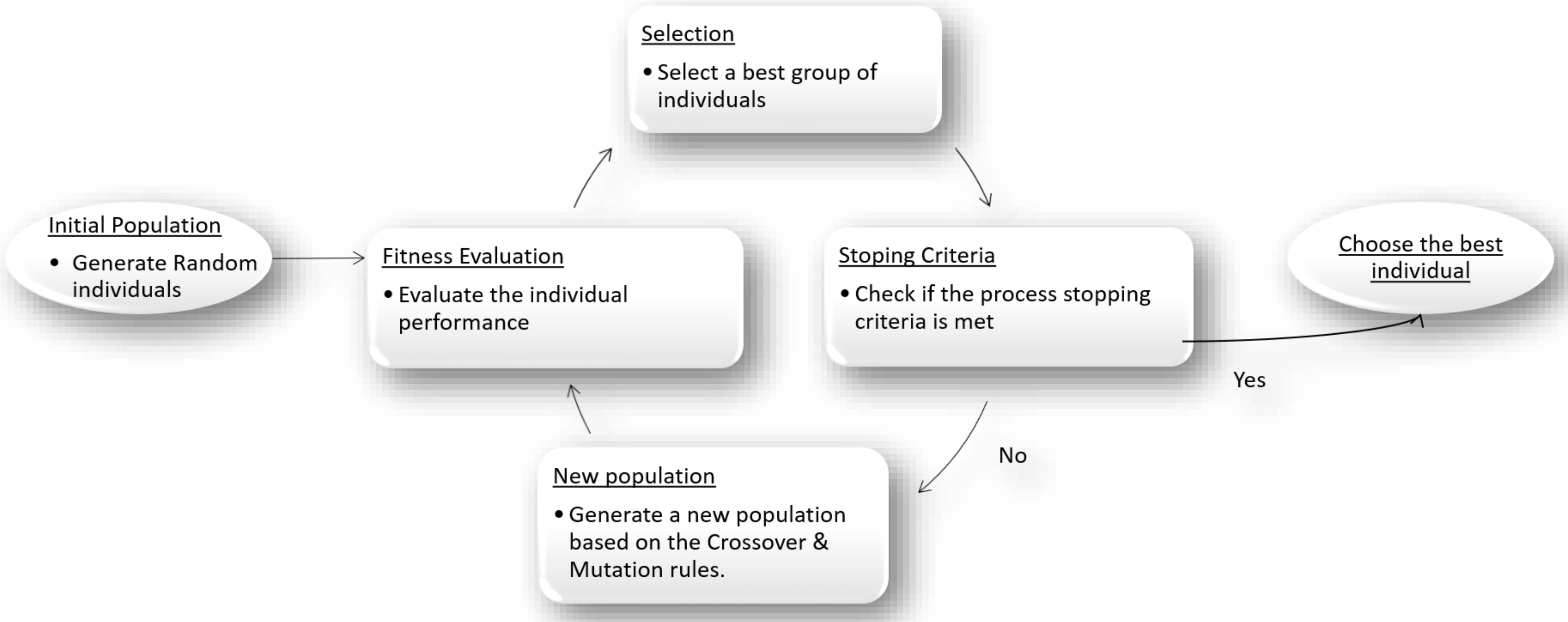
Start with k randomly generated states (population)

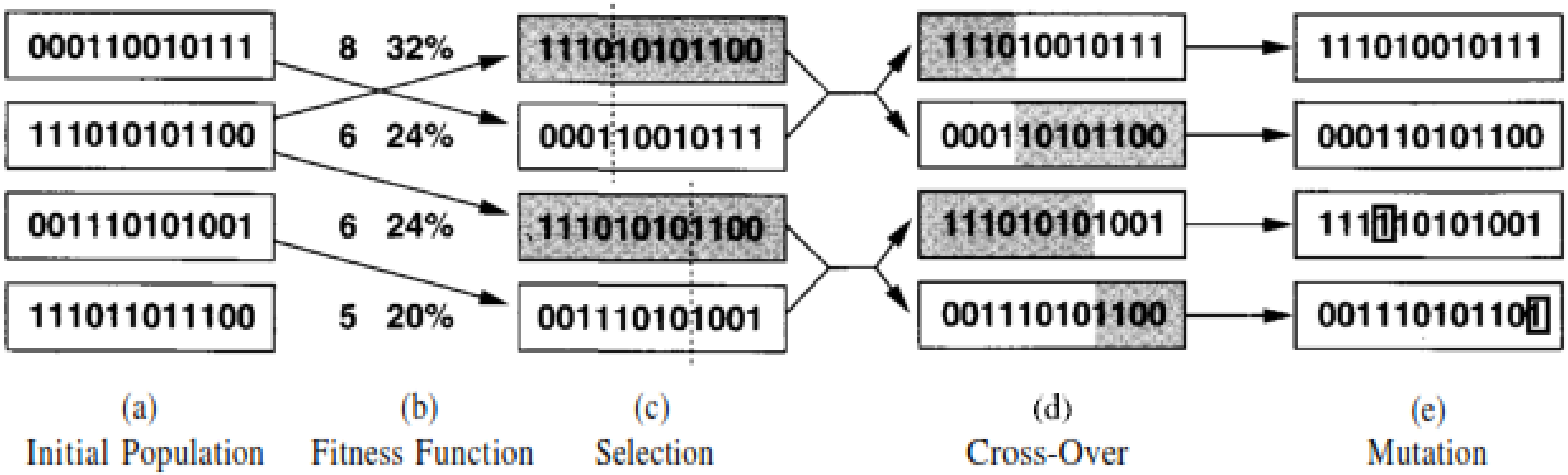
Evaluation function (fitness function):

- Higher values for better states.
- Opposite to heuristic function, e.g., # non-attacking pairs in 8-queens

Produce the next generation of states by “simulated evolution”

- Random selection
- Crossover
- Random mutation

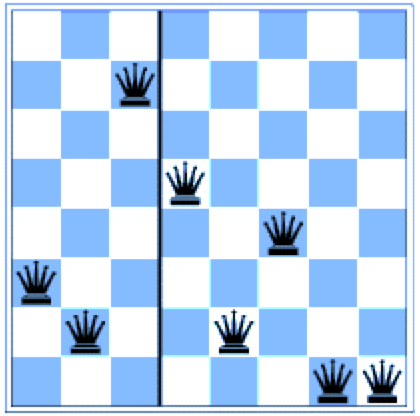

Initial Population
Generate Random individuals
Fitness Evaluation
Evaluate the individual performance
Selection
Select a best group of individuals
Stoping Criteria
Check if the process stopping criteria is met
Yes
Choose the best individual
No
New population
Generate a new population based on the Crossover & Mutation rules.



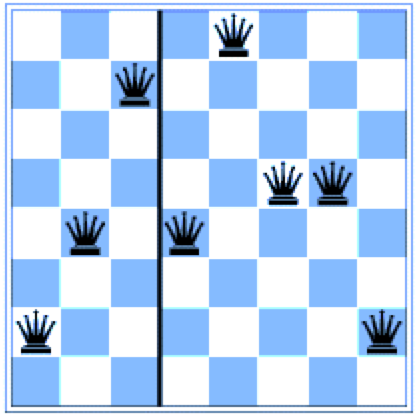
Source : Peter norvig genetic algorithm illustration

[2,1,6,4,1,3,0,0]

f()=5

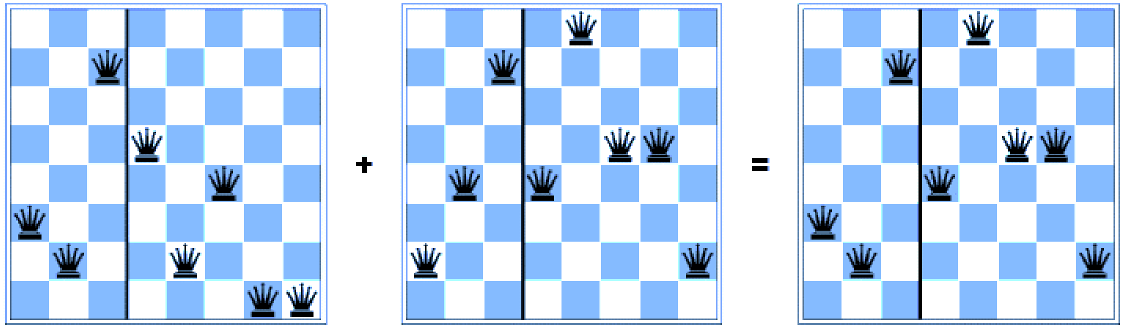


[1,3,6,3,7,4,4,1]



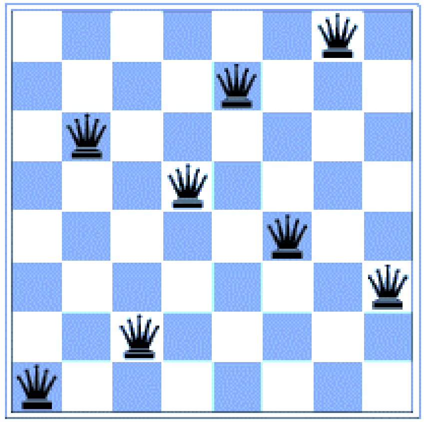
f()=4

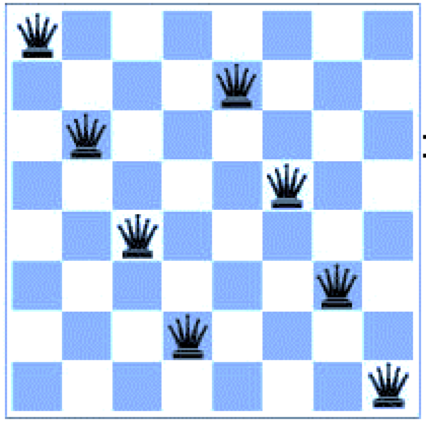
[2,1,6, 3,7,4,4,1]



f()=5

f()=1





f()=1

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

[0,5,1,4,6,3,7,2]

[7,5,3,1,6,4,2,0]

[7,5,1,4,6,3,2,0]

[1,1,1,2,3,3,4,5]

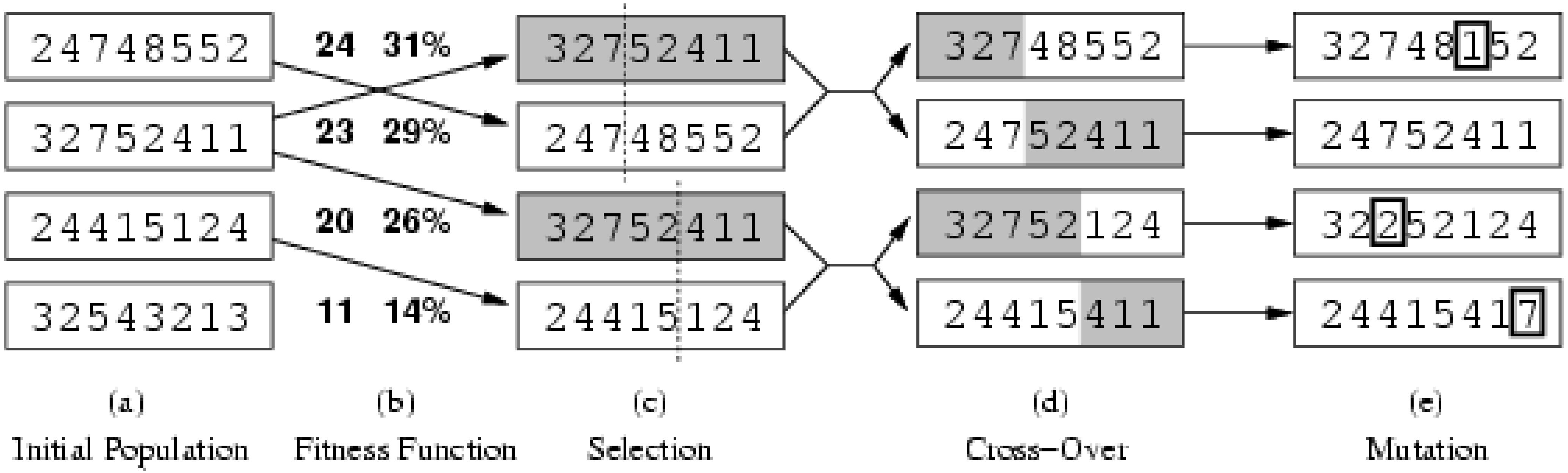
[2,4,4,1,5,1,2,4]

[2,4,7,4,8,5,5,2]

[3,2,5,4,3,2,1,3]

[3,2,7,5,2,4,1,1]

[0,1,2,6,6,7,3,1]


24748552
32752411
24415124
32543213
24 31%
23 29%
20 26%
11 14%
32752411
24748552
32752411
24415124
32748552
24752411
32752124
24415411
32748152
24752411
32252124
24415417
(a)
Initial Population
(b)
Fitness Function
(c)
Selection
(d)
Cross–Over
(e)
Mutation

COMMENTS ON GENETIC ALGORITHMS

- Positive points
  - Random exploration can find solutions that local search can't
    - (via crossover primarily)
  - Appealing connection to human evolution
    - "neural" networks, and "genetic" algorithms are **metaphors**!

- Negative points
  - Large number of "tunable" parameters
    - Difficult to replicate performance from one problem to another