Chapter 4
Beyond Classical Search

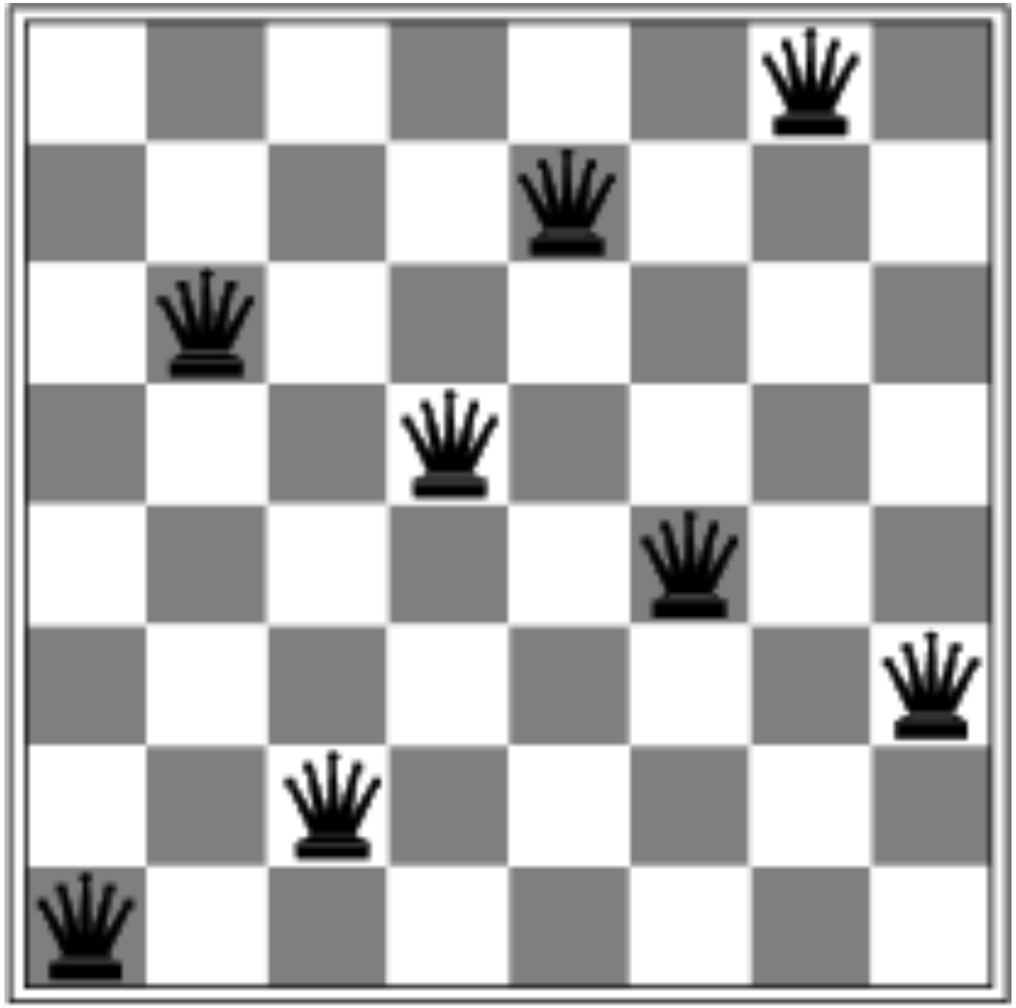
Jane Hsu
National Taiwan University

Acknowledgements: This presentation is created by Jane hsu based on the lecture slides from *The Artificial Intelligence: A Modern Approach* by Russell & Norvig, a PowerPoint version by Min-Yen Kan, as well as various materials from the web.

Outline

- Local Search Algorithms
  - Hill climbing
  - Simulated annealing
  - Genetic algorithms
- Local search in continuous spaces
- Searching with nondeterministic actions
- Searching with partial observations
- Online Search

Example: 8-Queens Problem



Standard Search Problem

- State space: the set of all states reachable from the initial state
- State is a “black box” – any data structure that supports
  - successor function
  - heuristic function, and
  - goal test

Local Search Algorithms

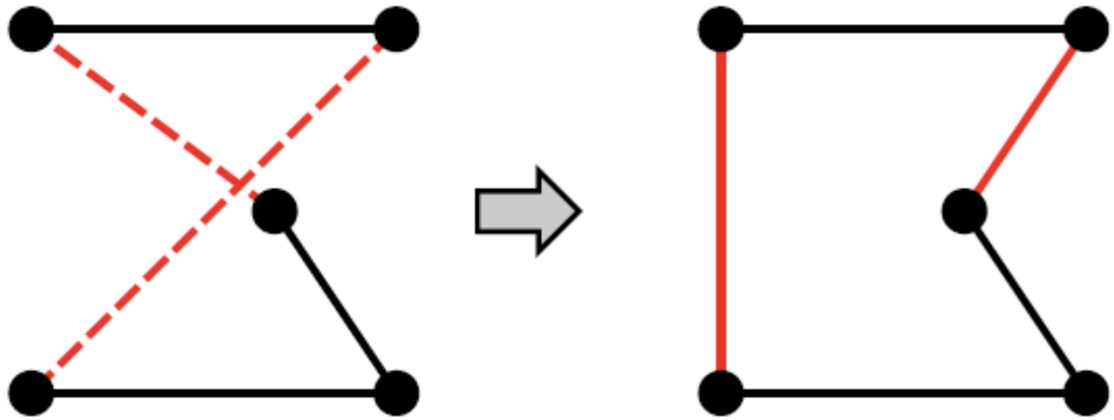
Jane Hsu
National Taiwan University

Iterative Improvement Algorithms

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

- State space = set of "complete" configurations
  - Find optimal configuration, e.g., TSP
  - Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use *iterative improvement*, also called *local search* algorithms
  - keep a single "current" state, try to improve it
  - Constant space
  - Online or offline search

Traveling Salesperson Problem

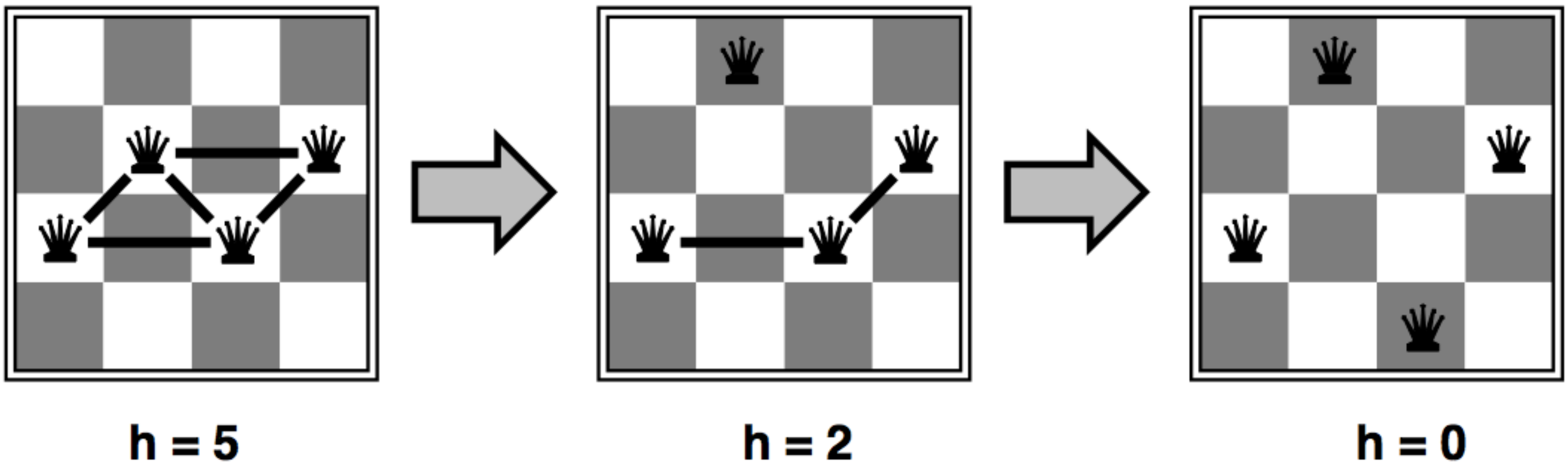


- Start with any complete tour, perform pair-wise exchanges.
- Variants of this approach get within 1% of optimal very quickly with thousands of cities.

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



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

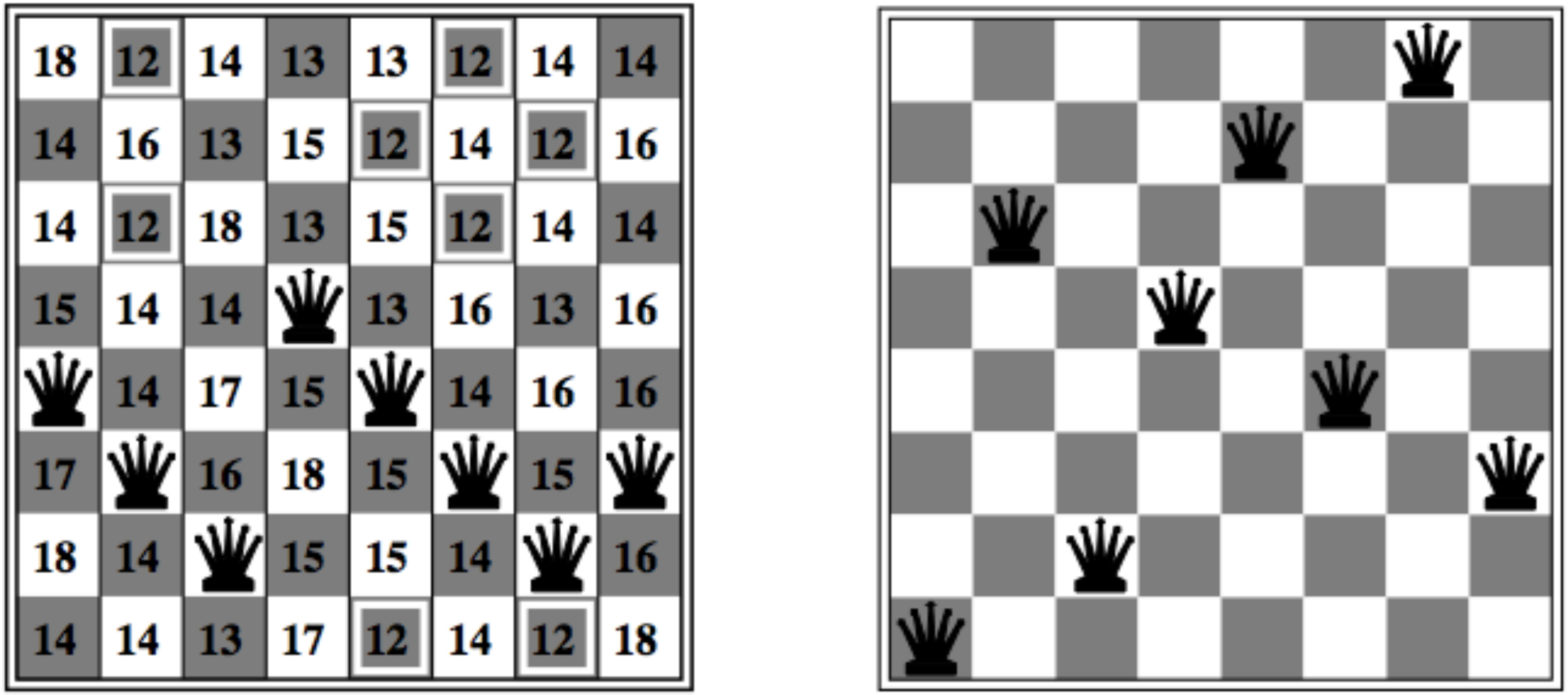
Hill-Climbing (Gradient Ascent/Descent)

"Like climbing Everest in thick fog with amnesia"

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

    current ← MAKE-NODE(*problem*.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of *current*
        if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
        current ← *neighbor*

Local Maxima

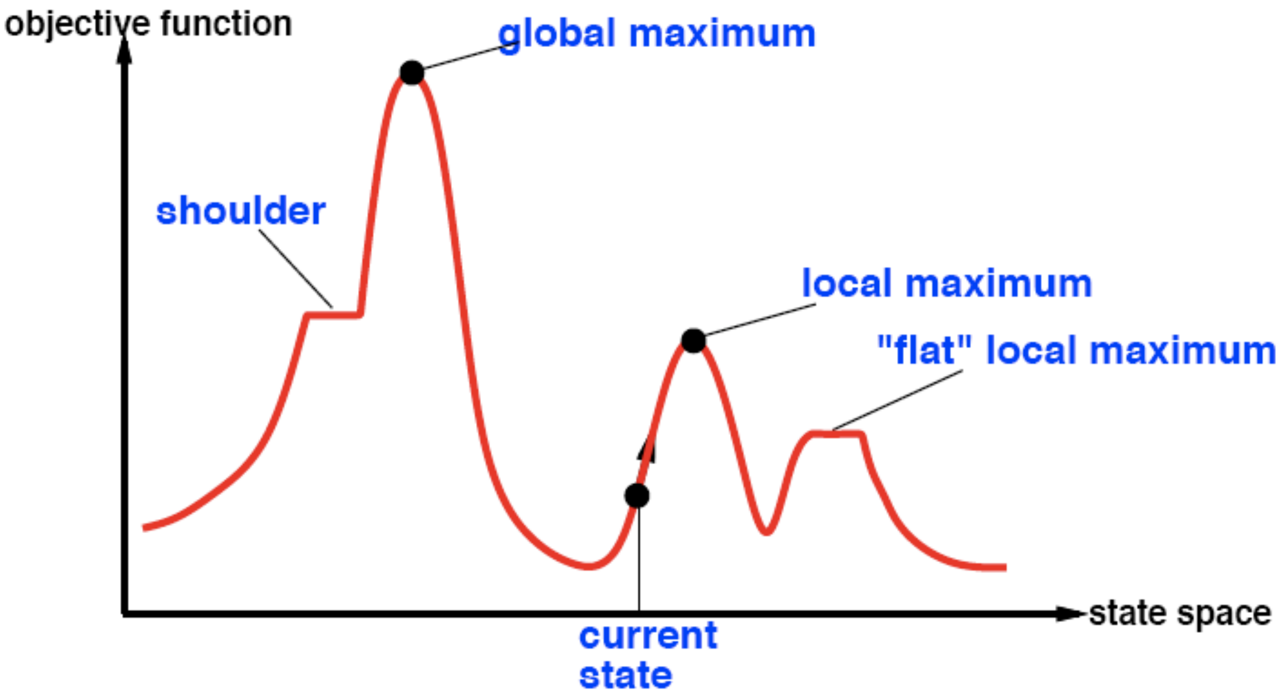


(a) (b)

Figure 4.3 FILES: figures/8queens-successors.eps figures/8queens-local-minimum.eps. (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

Problems in Hill-Climbing Search

- Consider the state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves 😊escape from shoulders ☹loop on flat maxima

Simulated Annealing

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their size & frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE – current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

Properties of Simulated Annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\Longrightarrow$ always reach best state x^* because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

Local Beam Search

- Keep track of (top) *k* states rather than just one
  - Start with *k* randomly generated states
  - At each iteration, all the successors of all *k* states are generated
  - If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.
- Question: is it *k* searches run in parallel?

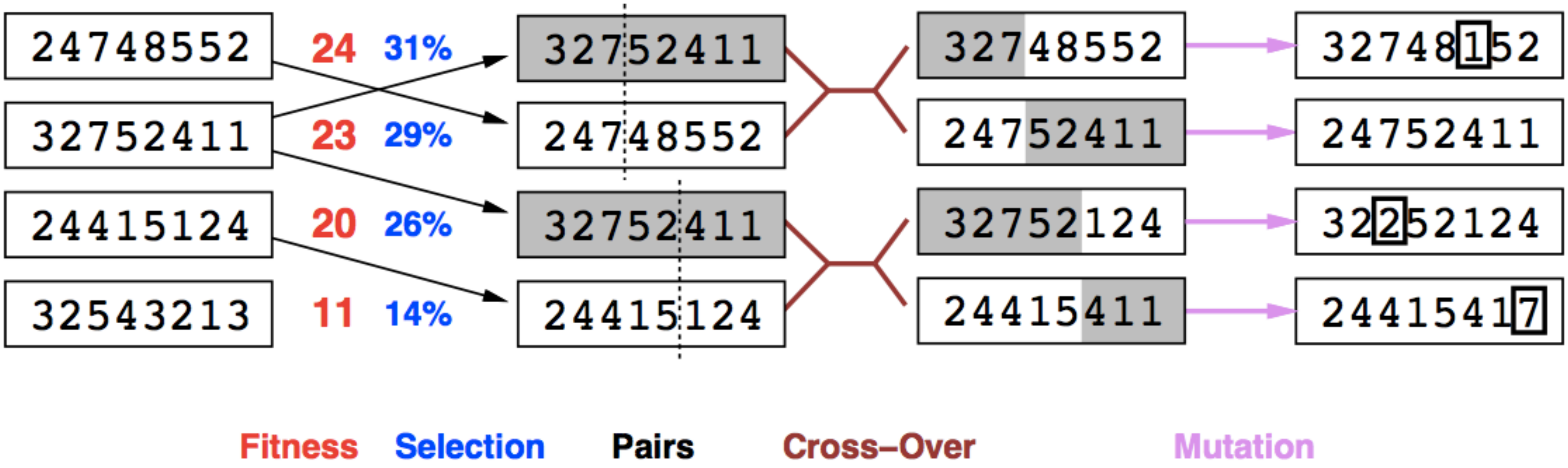
- Problem: all *k* states end up on same local hill
- Solution idea: choose *k* successors randomly, biased towards good ones.

Genetic Algorithms

- Population: Start with *k* randomly generated individuals (i.e. states)
  - Individual: each is represented as a string over a finite alphabet (often a string of 0s and 1s)
  - Fitness function: evaluation of the "goodness" of a given state.
- A successor is generated by combining two parents from the current population.
- Produce the next generation of states by selection, crossover, and mutation

Genetic Algorithms

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

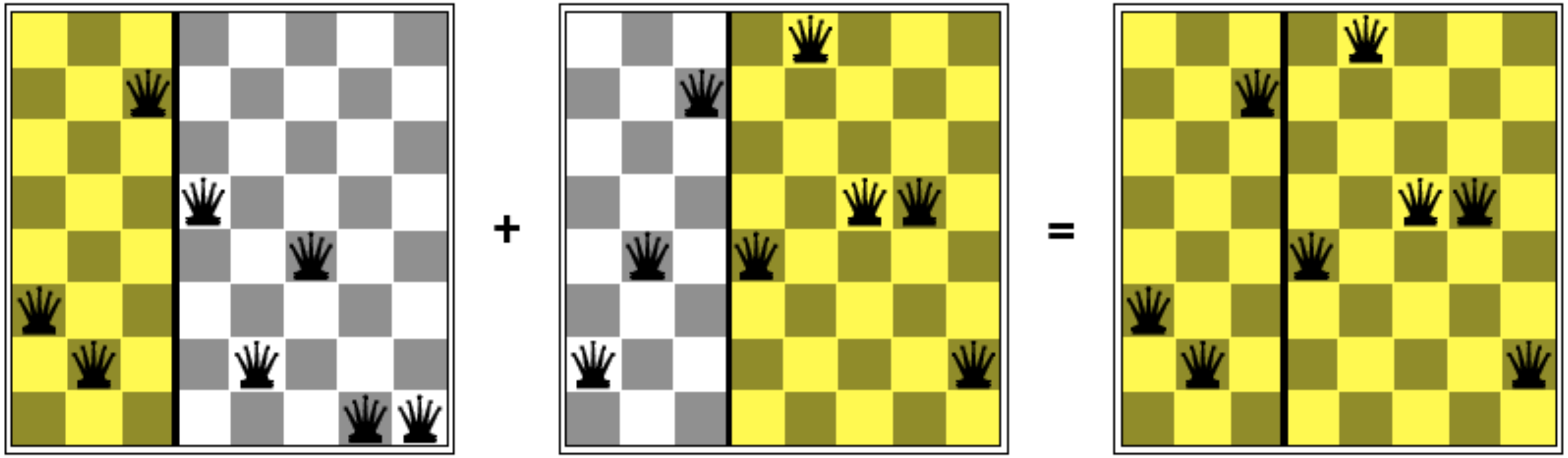


- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Crossover Operators

- States are encoded as strings

Crossover helps iff substrings are meaningful components

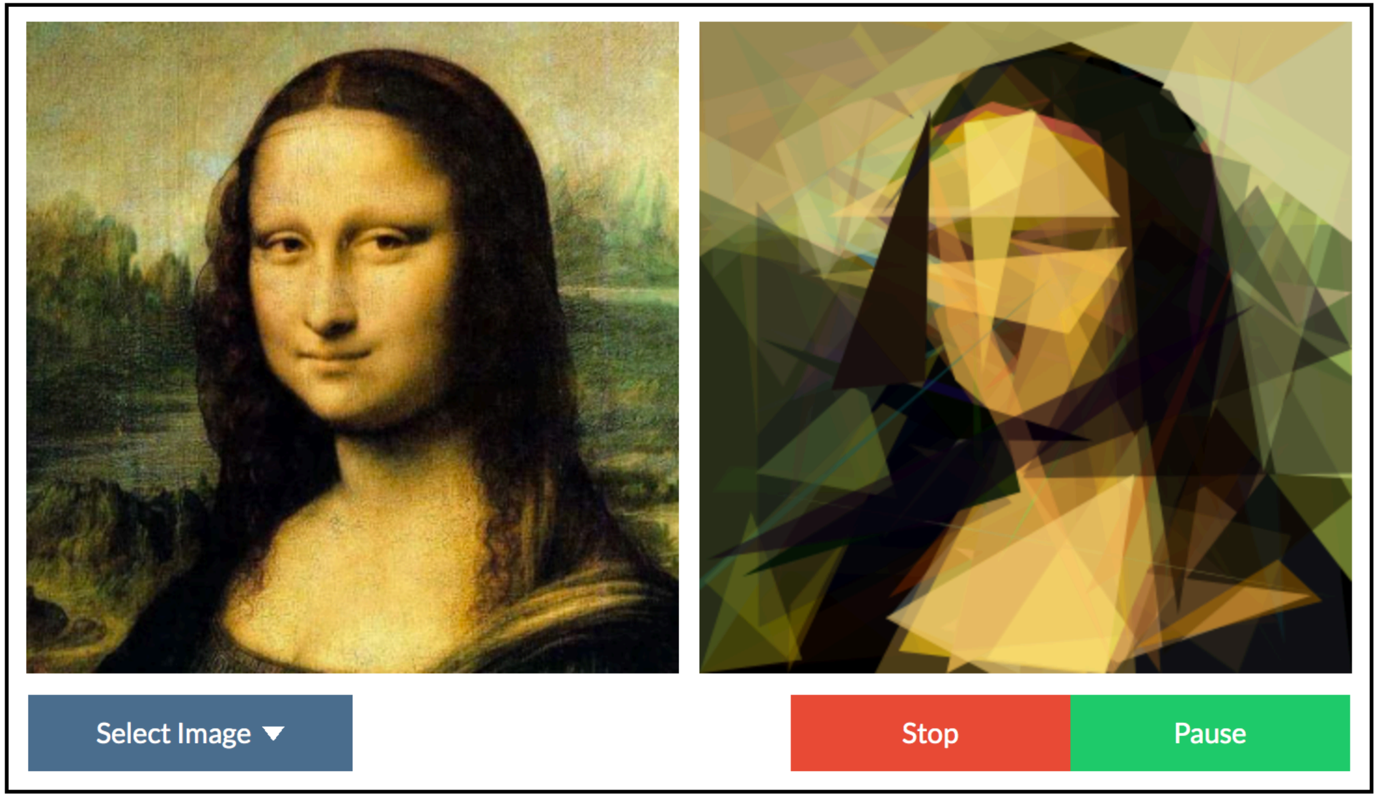


```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        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, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Grow Your Own Picture

Genetic Algorithms & Generative Art



Evolving the Mona Lisa

Welcome! Click start to see genetics in action, as a randomly generated collection of shapes evolve to resemble a given picture.

HOW IT WORKS

This page uses a genetic algorithm to model a population of individuals, each

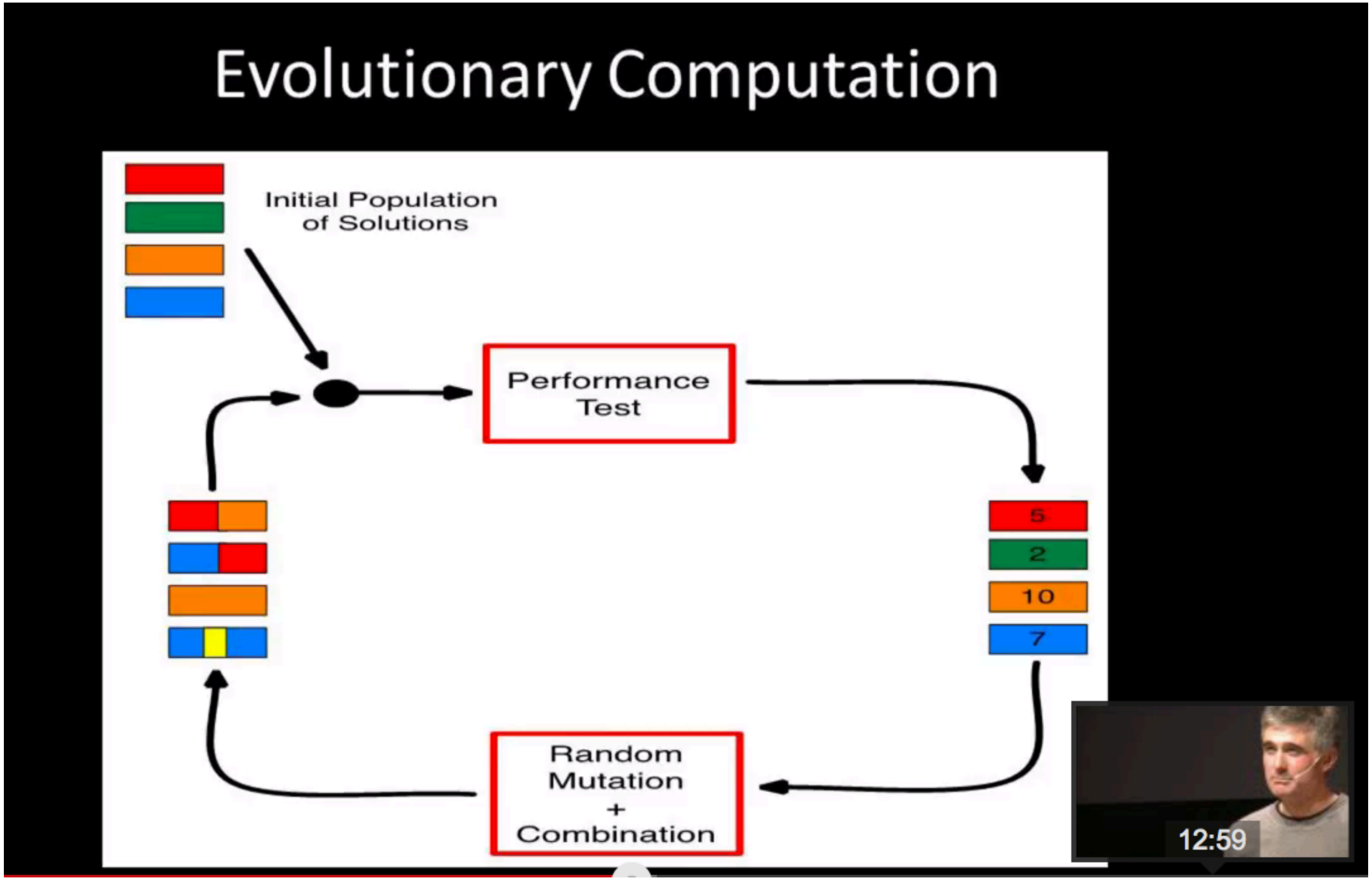
Configuration | Analytics

GENETICS

Population Size 50

http://chriscummins.cc/s/genetics/

Evolutionary Computation



Keith Downing at TEDxTrondheim