

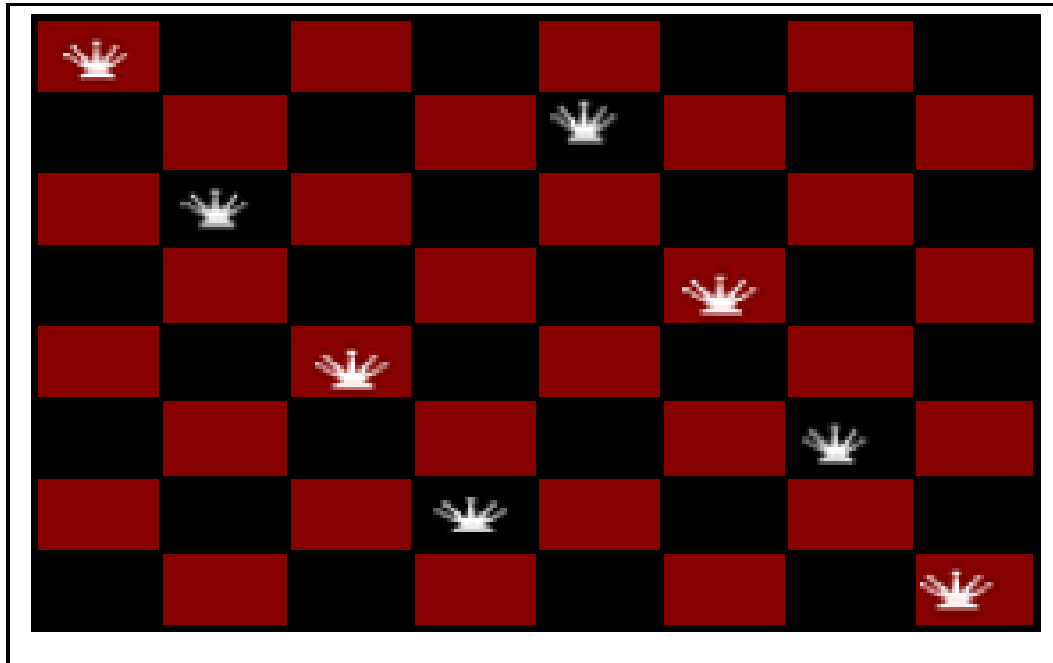
# Genetic Algorithms

Derek Hao Hu

Sep 18, 2007

# How to encode into a state?

## 8-queens State Representation



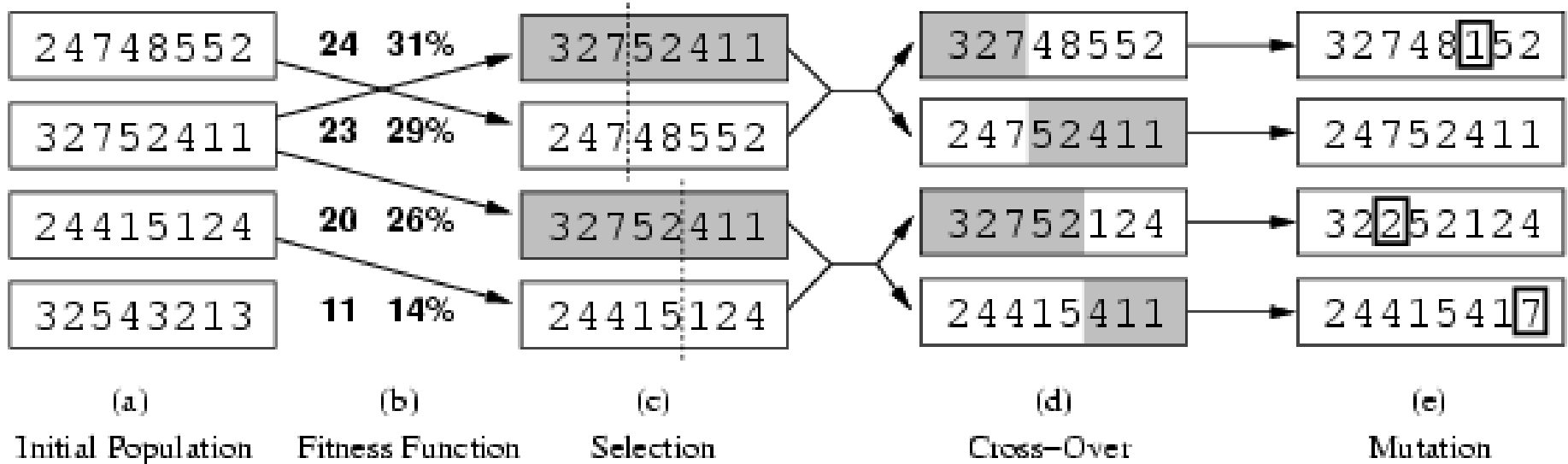
option 1: 86427531

option 2: 111 101 011 001 110 100 010 000

# How to do a genetic algorithm?

- 1. Define the state encoding
  - Done
- 2. define how to obtain the successor states
  - Genetic nature?
    - Selection, mutation, cross over, deletion, etc
- 3. define what is a good state
  - Fitness functions = heuristics to be maxed
- 4. define how to pick parents
  - Probabilities proportional to fitness values
- 5. termination:

# GA: An Example



Fitness function: number of non-attacking pairs of queens (min = 0, max = 28)

The key point to have a good genetic algorithm is to

- [1] have a good fitness function
- [2] have a good state representation

# Features and motivations

- The approach of GA simply mimics the process of evolution in biology.
- It is usually presented using a biological vocabulary vs computer science:
  - fitness, populations, individuals, genes, crossover, mutations, etc.
- It can still be viewed in the “local search” perspective.

# Motivation (cont.)

- It is inspired by biological processes that produce genetic changes in populations of individuals.
- Genetic algorithms (GAs) usually have the following characteristics:
  - A Darwinian notion of fitness: the most fit individuals have the best chance of survival and reproduction.
  - “Crossover” operators
    - Parents are selected.
    - Parents pass their genetic material to their children.
  - Mutation: individuals are subject to random changes in their genetic material.

# Genetic Algorithm: A pseudocode

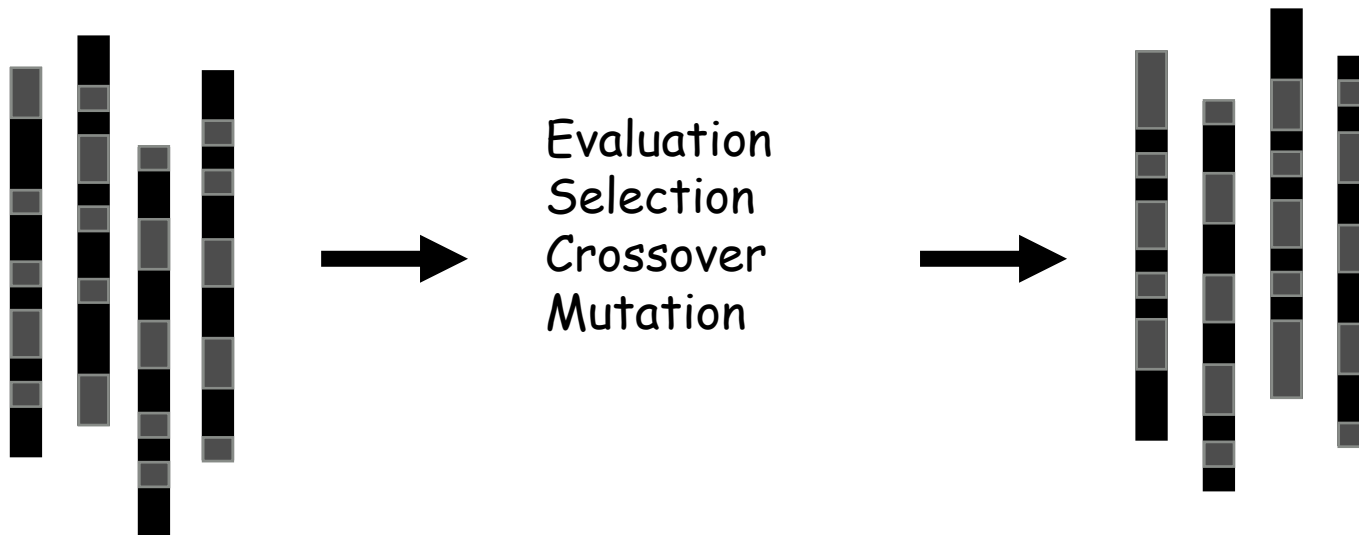
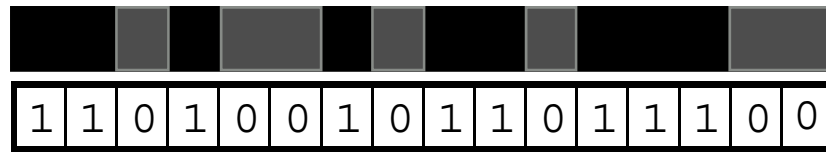
- Choose initial population (usually random)
- Repeat (until terminated)
  - Evaluate each individual's fitness
  - Check for termination criteria
    - Number of generations?
    - Amount of time elapsed?
    - Minimum fitness threshold satisfied?
    - Fitness has reached a plateau?
  - Select pairs to mate from best-ranked individuals
  - Apply crossover operator
  - Apply mutation operator

# GA Algorithm (Description from AIMA)

```
function GENETIC_ALGORITHM( population, FITNESS-FN) return an individual
  input: population, a set of individuals
           FITNESS-FN, a function which determines the quality of the
           individual
  repeat
    new_population  $\leftarrow$  empty set
    loop for i from 1 to SIZE(population) do
      x  $\leftarrow$  RANDOM_SELECTION(population, FITNESS_FN)
      y  $\leftarrow$  RANDOM_SELECTION(population, FITNESS_FN)
      child  $\leftarrow$  REPRODUCE(x,y)
      if (small random probability) then child  $\leftarrow$  MUTATE(child )
      add child to new_population
      population  $\leftarrow$  new_population
  until some individual is fit enough or enough time has elapsed
  return the best individual
```

# String-based GAs

- Holland's original approach, and probably still the most common
- Genome (or chromosome) is a string of bits



# Fitness

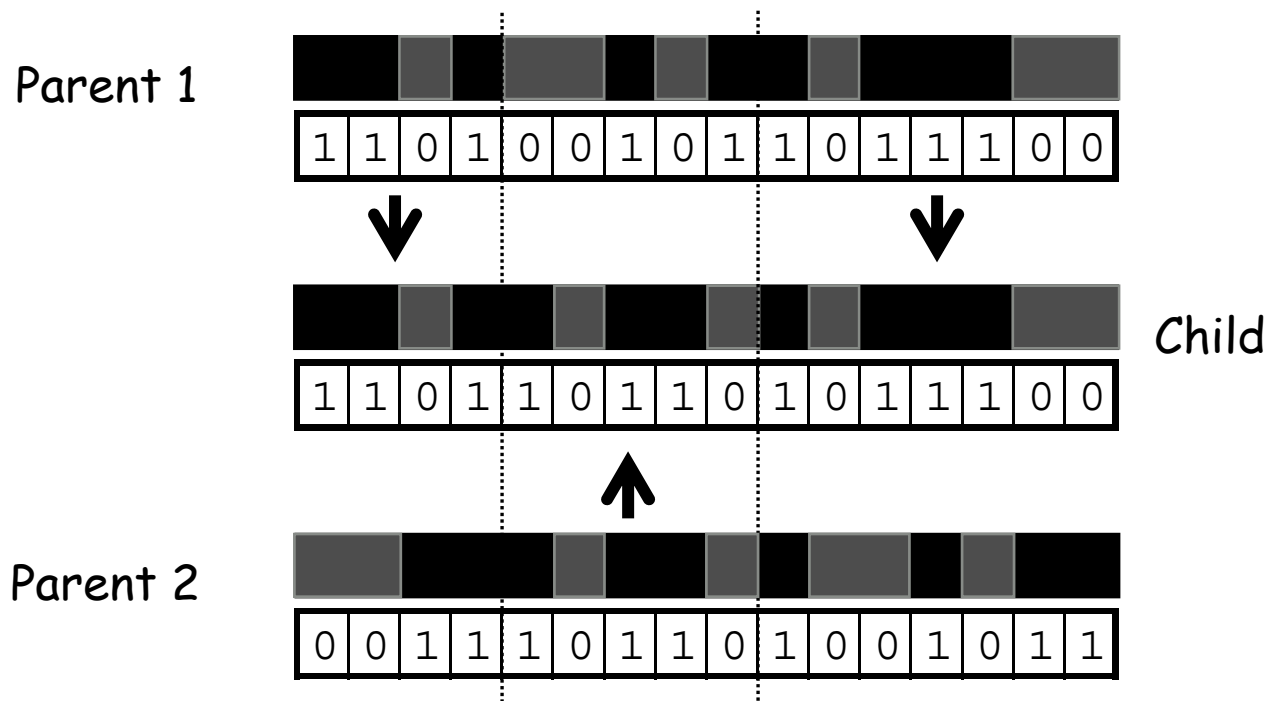
- Fitness is computed for each individual.
- Then individuals are chosen probabilistically for survival and crossover based on some selection rules.
  - Fitness proportionate selection
  - Tournament selection
  - Rank selection
  - Most of the up-to-date selection rules tend to formulate the Darwinian theory “Only the strong survive”.

# However, we should notice...

- Better individuals are preferred
- Best is not always picked
- Worst is not necessarily excluded
- Nothing is guaranteed
- Mixture of greedy exploitation and adventurous exploration
- Similarities to simulated annealing!

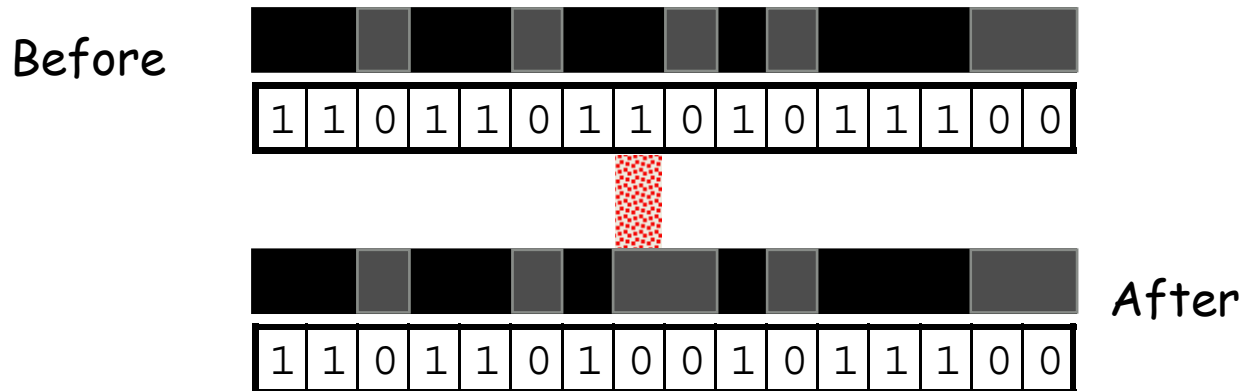
# Crossover

- There are a number of techniques about how to perform this “crossover” operation. However, all involve swapping genes – sequences of bits in the strings.



# Mutations

- Generally, bits are flipped with a small probability.
- This explores parts of the space that crossover might miss, and helps prevent premature convergence.

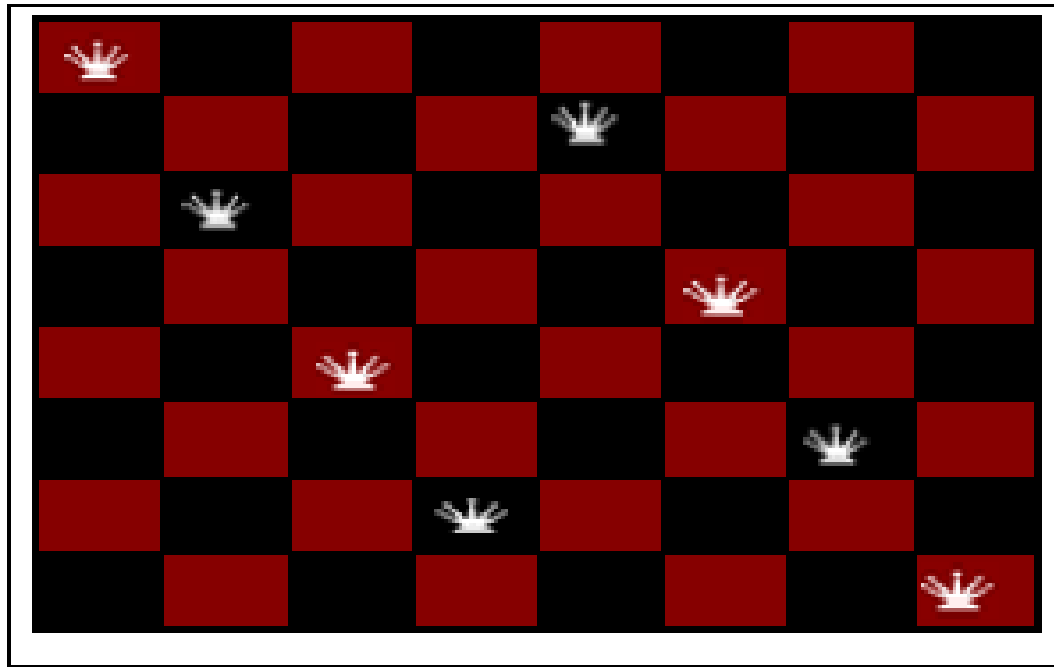


# Think about it?

- What happens if the mutation rate is too low?
- What happens if the mutation rate is too high?
- So a common solution is to
  - Use a high mutation rate when search begins
  - Decrease the mutation rate as the search progresses

# A working example – N Queens!

## 8-queens State Representation



option 1: 86427531

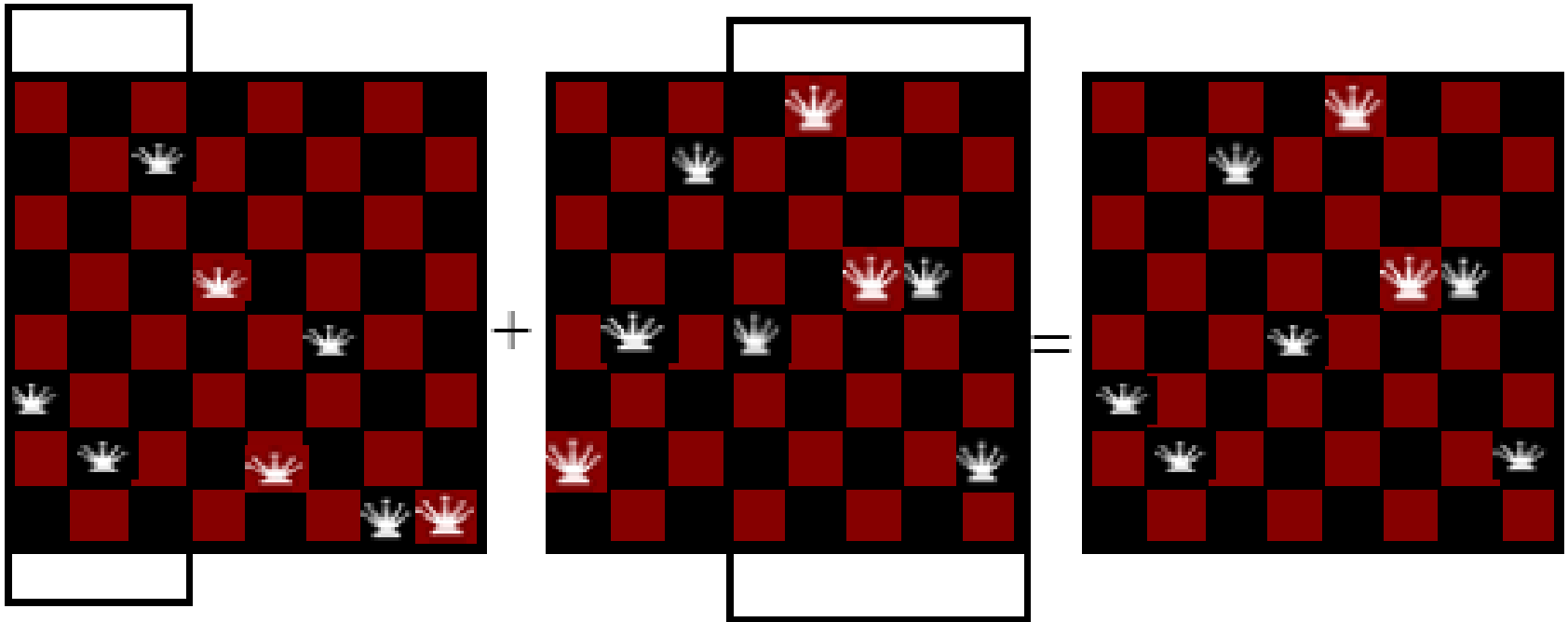
option 2: 111 101 011 001 110 100 010 000

# Fitness Function?

- Just as we had described before, for the n-queens problem we use the number of non-attacking pairs of queens.
- Remember that we should try to use some fitness functions that “have meanings”.

# Crossover Example

## Crossover Example



# Mutation for n-queen

- Choose a queen at a random position and then move it to another random position.
- Or choose two random queens and swap them...
- It works!
  - <http://www.iba.k.u-tokyo.ac.jp/english/userlog.cgi?queenrun>
- N-queens is a constraint satisfaction problem that was widely used as an exercise in algorithm design.
- In the following lecture, we will learn something that seems like “heuristic repair” / “iterative repair” that solves this problem even better.

# Some additional remarks

- Sometimes GA fails...
  - Actually, it does not work for many problems.
  - It might have a tendency to converge towards local optima or even arbitrary points rather than global optimum.
  - It does not know how to sacrifice the short-term fitness for long-term fitness.
  - The likelihood of this occurring depends on the fitness landscape. Certain problems maybe easy to ascent to a global optimum, others maybe easy to find a local optima.
  - Solution:
    - Different fitness function
    - High mutation rate
    - Use some selection rules to maintain diversity

# Genetic Programming

- Biological evolution to find computer programs that perform a user-defined task.
- John R. Koza is a main proponent of GP.
- Programs are represented as trees.
  - Still has the crossover and mutation operators.
- State-of-the-art results:
  - <http://www.genetic-programming.com/humancompetitive.html>