Search-Based Software Engineering/Testing

Bogdan Marculescu
Bogdan.marculescu@Kristiania.no
Man Zhang

First a quick bit of bureaucracy

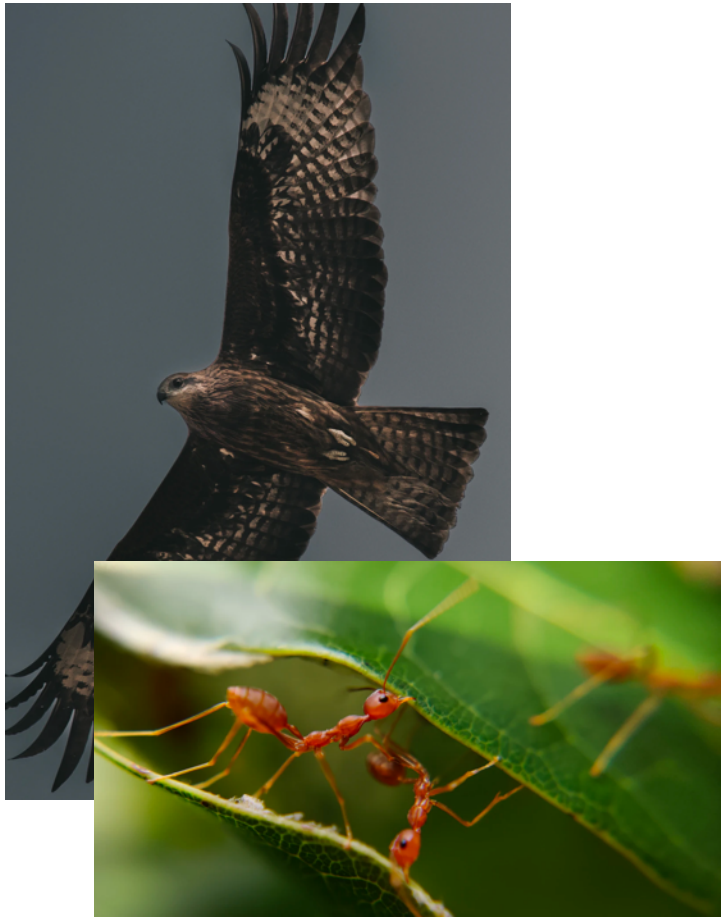
- Quick intro on theory
- Lunch break
- Small workshop on working with Search-Based Techniques

(we'll work together, so no worries on the development side)

Some problems… are different

- Exact solution may be:
  - Impossible
  - Costly to compute
  - Not necessary
- ”Knapsack problem”
- Optimization – “close enough”

Inspiration Outside of Math



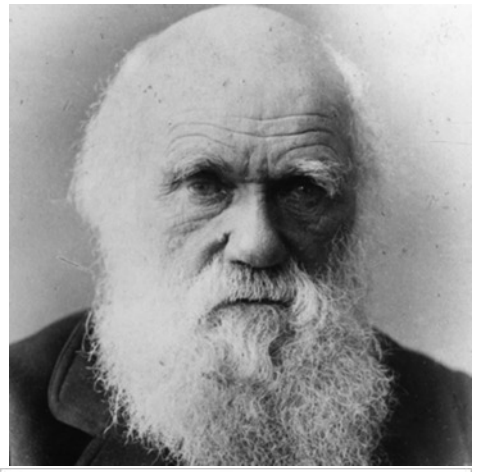
- Nature is good at solving different problems
- E.g.:
  - Carbon-to-Diamond (slow cooling)
  - Ants finding and gathering food
- Working (though perfectible) solutions
- Inspiration for algorithms

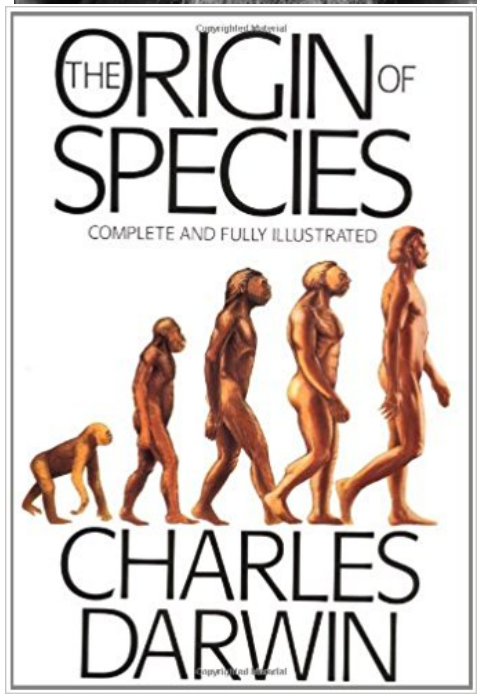
Theory Evolution

- Charles Darwin, *"The Origin of Species",* 1859

- Theory describing how different species *evolved* from unicellular organisms

- Evolutionary Computation, Genetic Algorithms, Search-Based Techniques





(1+1) EA

- 1 individual that produces 1 offspring
- Evolution is driven by mutations in the chromosome of the offspring
- Kill the offspring if worse than parent, otherwise kill parent after birth
  - Yep, evolution can be cruel…

In practice

Individuals – are the (approximate) solutions

- E.g. loading of the knapsack

Chromosome – representation of that solution

- E.g. binary array of objects (in knapsack or not)

Mutation – small changes that lead to *similar* individuals

- E.g. taking one object out leads to similar loading

Fitness – some measurement of solution quality

Mutation Example

1	1	0	0	0	1	0	1	0	0

Mutation: flip 1 bit

Non-zero probability - any solution

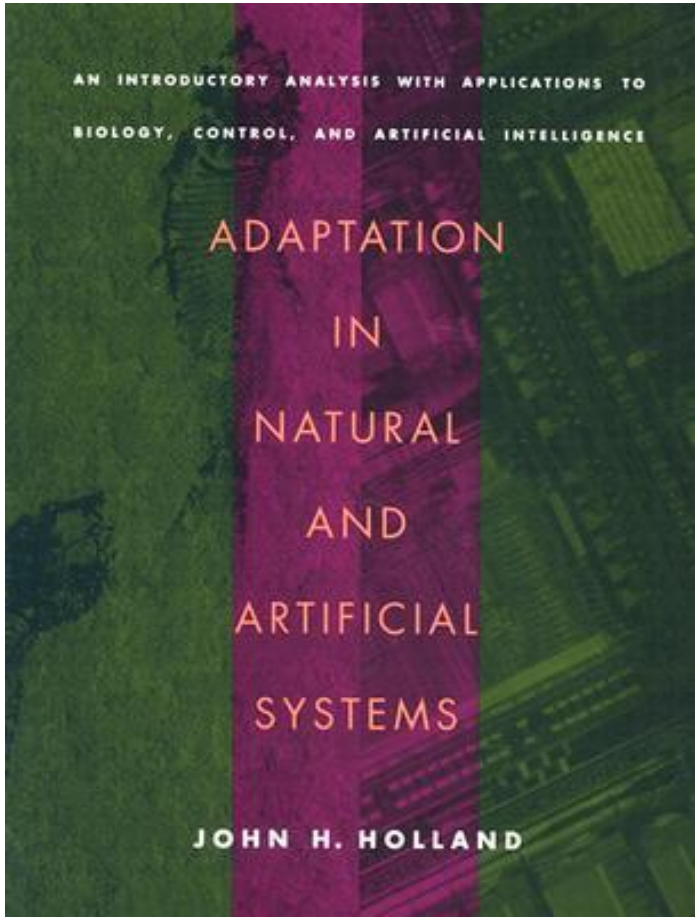
- ie, all bits can be flipped - low probability (1/N)^N

Reason: small modifications on average, but allow larger jumps

Fitness and Fitness functions

- Is this solution “good”?
- How good is it?
  - Compared to others?
  - Compared to previous version?
  - Compared to goal?
  - Measurable
  - Allow small changes
- Can become increasingly complex
- There may be more than one “good” solution

Genetic Algorithms (GAs)



- 1950s: Turing proposed use of evolution in computer programs
- 1970s: John Holland created GAs to address optimization problems
- Simulate evolution of an entire *population* of individuals, which procreate sexually

Population

- Keep track of more individuals
  - More likely to keep and combine “good” traits

- At the end of the search – select the best

Generations

- Select (more on that in a minute) individuals for reproduction
- Create a new generation of the same size
- Kill previous generation
  - Yes, evolution is STILL cruel

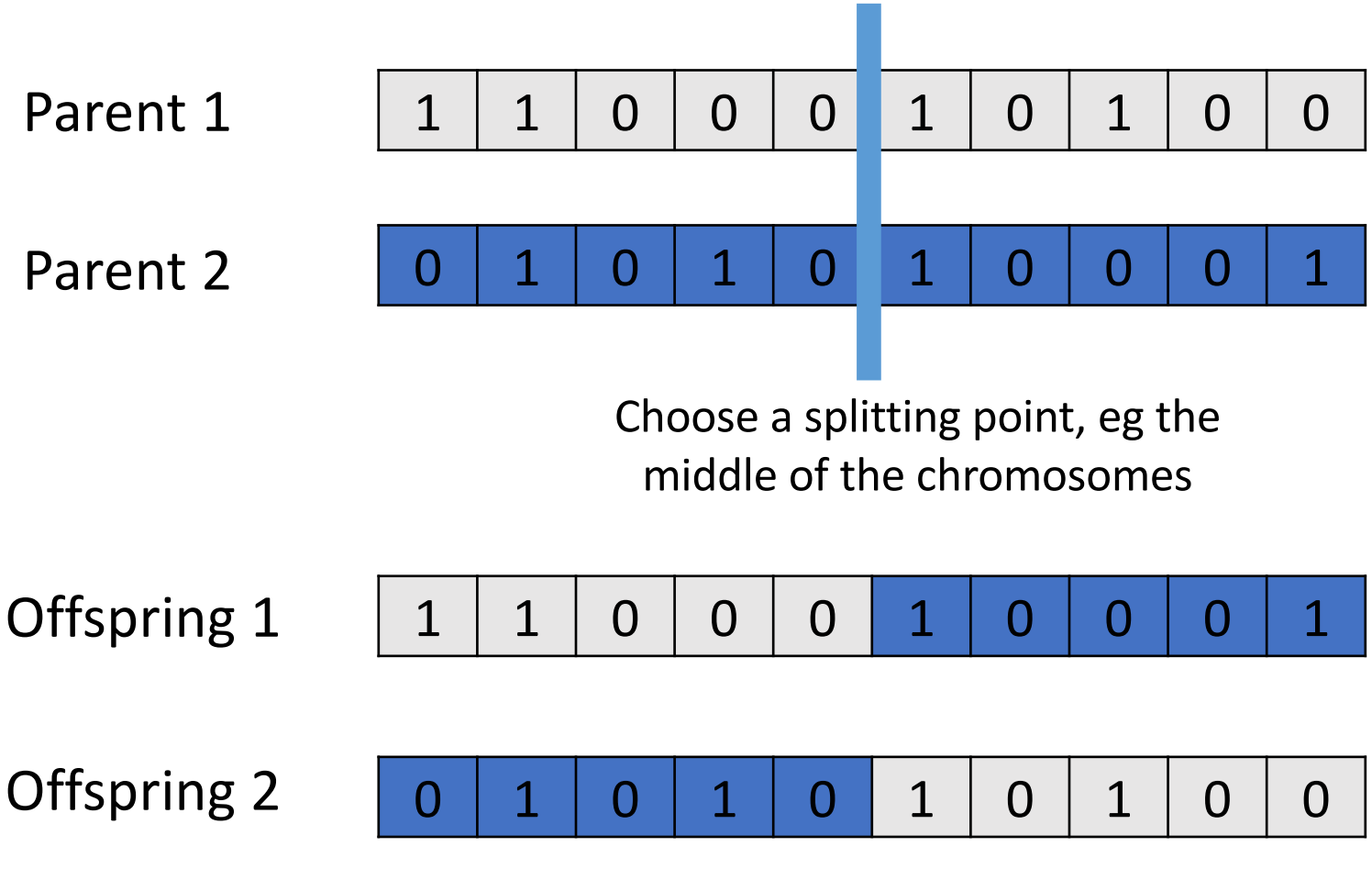
Selection

- The “best” (i.e. fittest) individuals - higher chances of reproduction
- Different strategies for parent selection
- Tournament Selection: sample T individuals *randomly* - choose the best
- All evaluations of “best” – Fitness Function

Sexual reproduction

- Select 2 parents, which give birth to 2 offspring
  - Note, ignoring gender here…
- Offspring will share genetic material with their parents, via the *crossover* operation (aka xover)
- After xover, offspring still have chances of getting mutated

Crossover


Parent 1
1 1 0 0 0 1 0 1 0 0
Parent 2
0 1 0 1 0 1 0 0 0 1
Choose a splitting point, eg the middle of the chromosomes
Offspring 1
1 1 0 0 0 1 0 0 0 1
Offspring 2
0 1 0 1 0 1 0 1 0 0

Crossover issues

- Chromosomes are constrained by domain
  - E.g. maximum weight of knapsack
- Crossover could lead to invalid offspring/individuals
- Options:
  - Discard (“kill”)
  - Repair

Elitism

- We (unlike nature) have a problem to solve, and finite time
- Don’t want to lose our ”best” individuals
- Preferential treatment (copy the “best” of each generation directly)

So, everything is all clear?

- More details to worry about:
  - “search landscape”, “search space”
  - Exploration (see more different solutions) vs
  - Exploitation (improve the best solutions so far)
  - Parameter tuning
  - Multiple (contradictory) objectives
  - Population diversity
  - etc

So, theory is okay. Practice?

Widely used, in a lot of different domains

- E.g.: antenna design
- **Chromosome**: shape of the antenna

- **Mutations**: changes in length, angles

- **Fitness**: improve beamwidth and impedance

So, theory is okay. Practice? – cont.

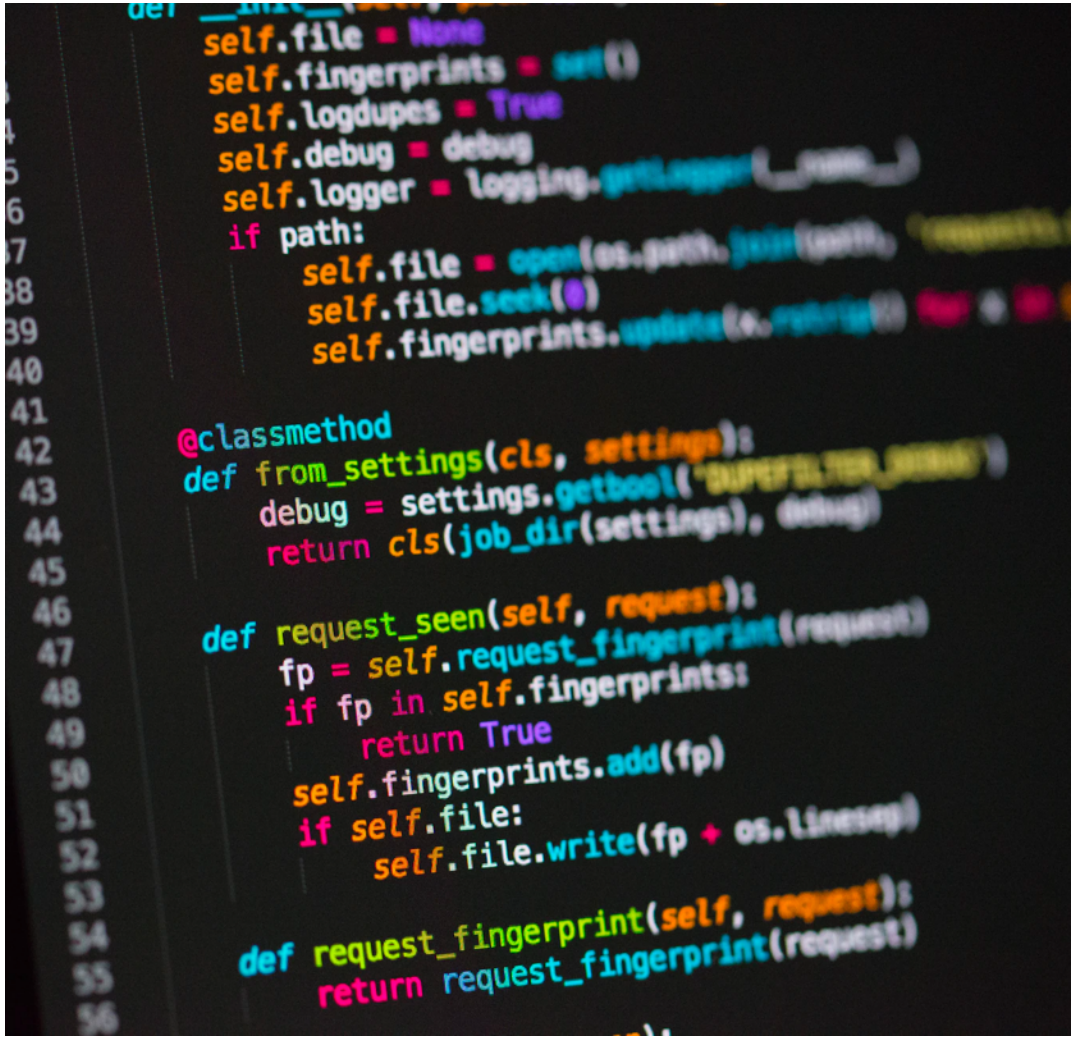
We use it for **software testing:**

- **Individual**: software tests?

- **Chromosome:** Test actions?

- **Mutations**: Sequence of test actions? Input values?

- **Fitness**: Bugs found? “Expected” behaviour?



Software Testing

- Run a system to assess its behaviour
- Ensure it does:
  - Everything that it should do – expected behaviour
  - Nothing that it shouldn’t do
- Use valid AND invalid inputs
- Try to break the system under test

Boxes of software testing

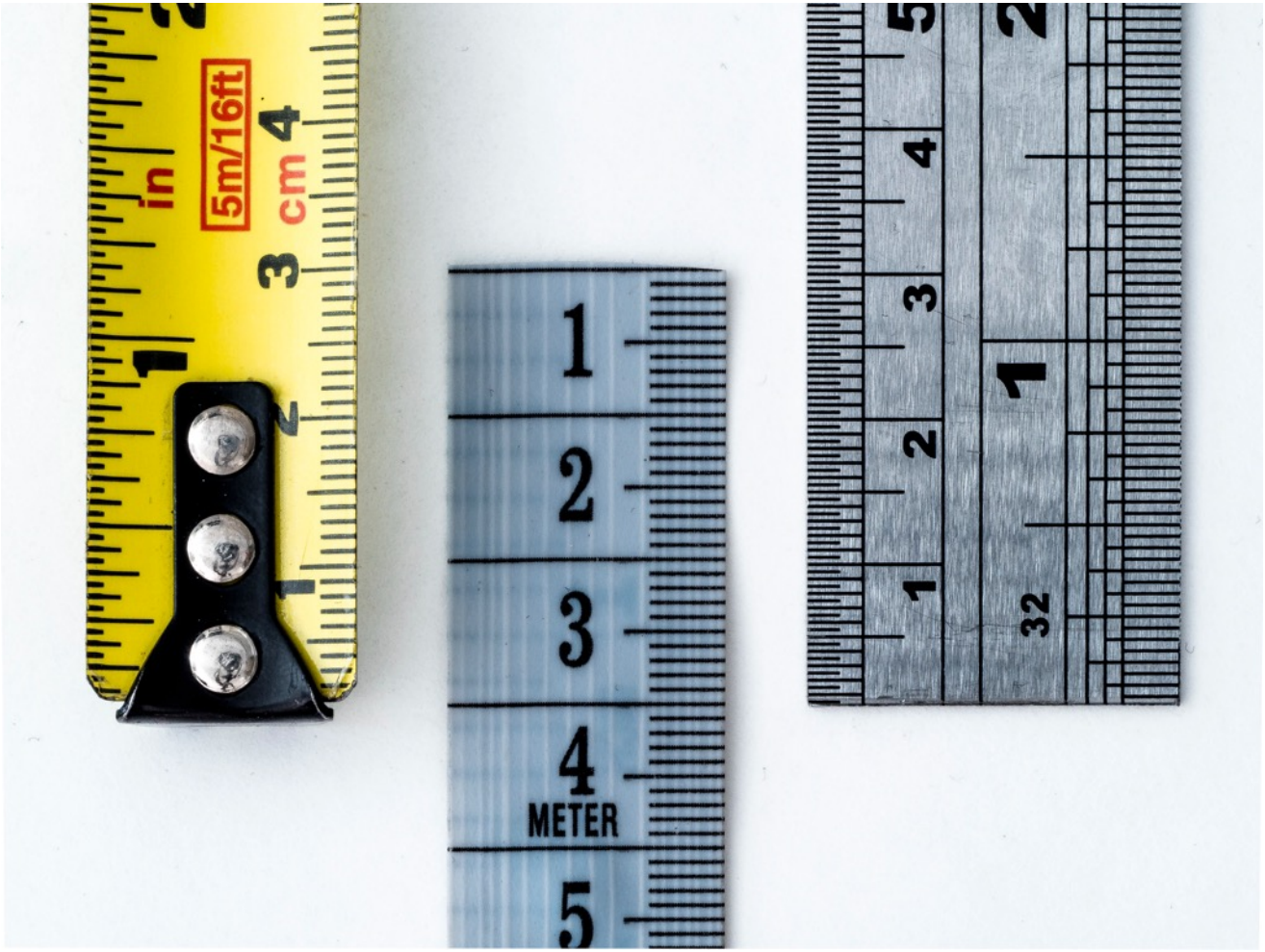
- Black-box testing:
  - Contents unknown
  - Can only observe behaviour
- ”Expected”/”Correct” behaviour only on completion
- Difficult to identify faults (debug)
- Requires domain knowledge to evaluate

Boxes of Software Testing – part 2

- White-box testing
  - Contents known
  - Can trace program execution
  - Can evaluate coverage (code, branch, etc.)
- Partial results
- Easier to debug
- More complex to develop and use

Fitness functions for testing?

- Black-box
  - Crash oracle?
  - ”best”/”better” solutions
- White-box:
  - Lines of code covered (number, overlap with previous)
  - Branch distance

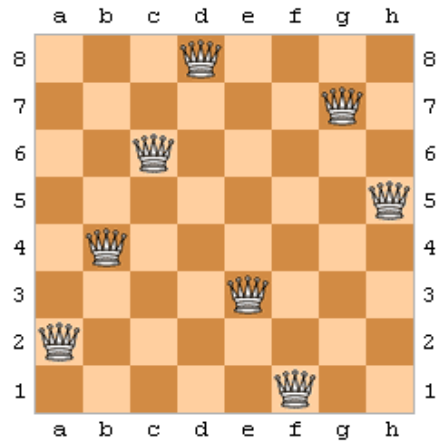


Quick exercise 1 – Knapsack problem

- Let's write a fitness function for the knapsack problem

- Basic restrictions:
  - Binary array of objects
  - Objects have different weights
  - Load up the backpack as much as possible (without exceeding capacity)

Quick exercise 2 - mutation

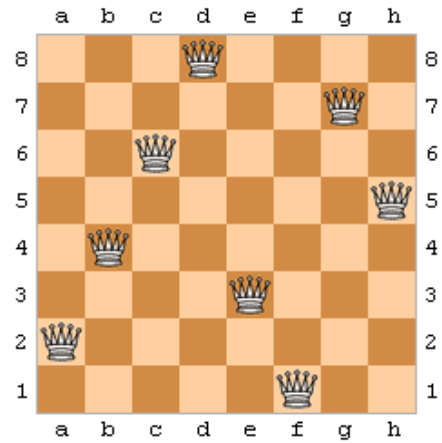


- On a chess board
  - As many queens as possible
  - Without threatening each other
- Representation:?
- Fitness: N queens on the board
  - Minimize the number that are threatened

Quick exercise 2 - mutation

- On a chess board
  - As many queens as possible
  - Without threatening each other

- Representation: matrix N*N of bits
  - 1 for a queen in that position, 0 otherwise



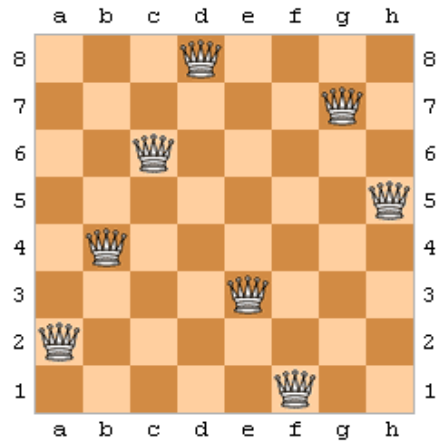
00010000
00000010
00100000
00000001
01000000
00001000
10000000
00000100

Quick exercise 2 - mutation

- On a chess board
  - As many queens as possible
  - Without threatening each other

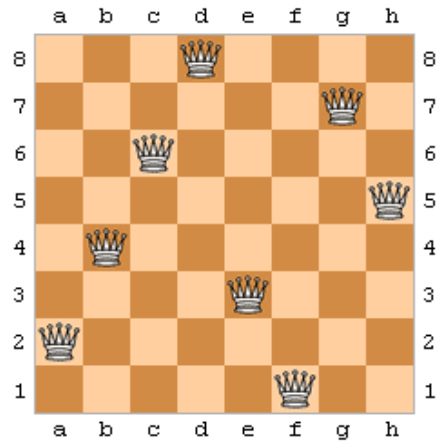
- Representation: [f,a,e,b,h,c,g,d]
  - Eg, queen in row 1 is in column *“f”,* row 2 is in column *“a”*, etc

- Mutation operator:



[f,a,e,b,h,c,g,d]

Quick exercise 2 - mutation



[f,a,e,b,h,c,g,d]

- On a chess board
  - As many queens as possible
  - Without threatening each other

- Representation: [f,a,e,b,h,c,g,d]
  - Eg, queen in row 1 is in column *“f”,* row 2 is in column *“a”*, etc

- Mutation operator: swap two elements
  - Eg, swap “f” with “c”, [c,a,e,b,h,f,g,d]

For more code on these exercises:

With thanks to Prof. Andrea Arcuri, whose algorithms course examples can be found below:

- https://github.com/arcuri82/algorithms/tree/master/solutions/src/main/java/org/pg4200/sol11

- https://github.com/arcuri82/algorithms/tree/master/lessons/src/main/java/org/pg4200/les11/ea

Exercise 3 – EvoMaster and NewsAPI

- News API
  - Simple API
  - Allows users to send in and receive news
  - Each news item has:
    - An id (auto-generated)
    - An author
    - A text
    - A country
- Let’s try to test it

References

- Note! This is just scratching the surface on a pretty wide (and dare I say, interesting) topic.

- A couple of references to get you started:

- Mark Harman, Phil McMinn, Jerffeson Teixeira Souza, and Shin Yoo. Search-Based Software Engineering: Techniques, Taxonomy, Tutorial. Empirical Software Engineering and Verification, Lecture Notes in Computer Science, vol. 7007, pp. 1–59, 2011
  - https://mcminn.io/publications/ic2.html

- Harman, Mark, Yue Jia, and Yuanyuan Zhang. "Achievements, open problems and challenges for search based software testing." *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2015.

Questions and discussion