# Artificial Intelligence Foundation – JC3001

## Lecture 9: Search III: Local Search II

**Prof. Aladdin Ayesh** (aladdin.ayesh@abdn.ac.uk)

**Dr. Binod Bhattarai** (binod.bhattarai@abdn.ac.uk)

**Dr. Gideon Ogunniye,** (g.ogunniye@abdn.ac.uk)

September 2025



Material adapted from:
Russell and Norvig (AIMA Book): Chapter 4 (4.1–4.4)
Dana Nau (University of Maryland)

UNIVERSITY OF ABERDEEN

# Course Progression

- Part 1: Introduction
  1. Introduction to AI ✓
  2. Agents ✓
- Part 2: Problem-solving
  1. Search 1: Uninformed Search ✓
  2. Search 2: Heuristic Search ✓
  3. **Search 3: Local Search**
  4. Search 4: Adversarial Search
- Part 3: Reasoning and Uncertainty
  1. Reasoning 1: Constraint Satisfaction
  2. Reasoning 2: Logic and Inference
  3. Probabilistic Reasoning 1: BNs
  4. Probabilistic Reasoning 2: HMMs
- Part 4: Planning
  1. Planning 1: Intro and Formalism
  2. Planning 2: Algos and Heuristics
  3. Planning 3: Hierarchical Planning
  4. Planning 4: Stochastic Planning
- Part 5: Learning
  1. Learning 1: Intro to ML
  2. Learning 2: Regression
  3. Learning 3: Neural Networks
  4. Learning 4: Reinforcement Learning
- Part 6: Conclusion
  1. Ethical Issues in AI
  2. Conclusions and Discussion

# Objectives

- Single-state problems ✓
- Local Search and Optimization
  - Hill Climbing ✓
  - Simulated Annealing ✓
  - Beam Search
  - Genetic Algorithms


# Outline

1: **function** Beam-Search($problem$, $k$)
2: start with $k$ random generated states
3: **loop**
4: generate all successors for all $k$ states
5: **if** any of them is a solution **then** return it
6: **else** selected $k$ best successors

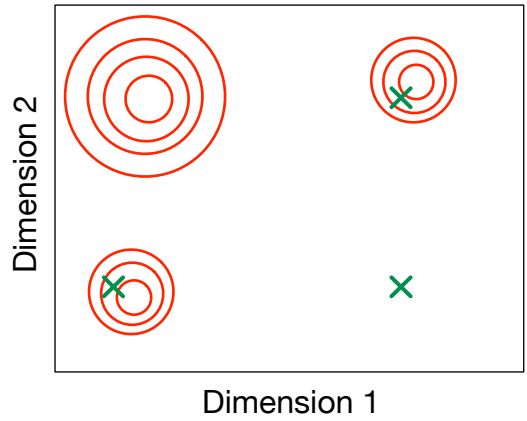
Questions to consider in analysing Beam search algorithm:

- What happens if there is no solution found?
- What happens if all the states have the same score?

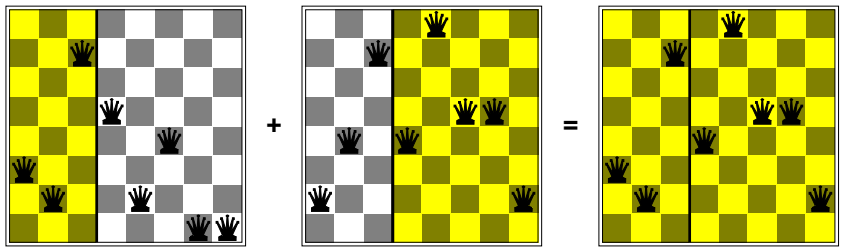
We may find variations on the Beam search algorithm, but be mindful of the implication of the changes made to the algorithm.

- Not the same as $k$ parallel searches
  - Searches that find good states will recruit other searches to join them
  - Problem: often all $k$ states end up on same local hill
- Stochastic beam search:
  - Choose $k$ successors randomly, biased towards good ones
  - Close analogy to natural selection

- Population of individuals
- Mutation — local search $N(x)$
- cross over — population holds information
- generations — iterations of improvement



- Genetic algorithms
  - stochastic local beam search
  - generate successors from pairs of states
- Each state should be a string of characters — substrings should be meaningful components
- Example: n-queens problem
  $i^{th}$ character = row where $i^{th}$ queen is located



- Gene - characters in the string representing the state
- Chromosome - blocks of genes in the string in a state
- Population - neighbours in the search
- Selection, crossover, mutation

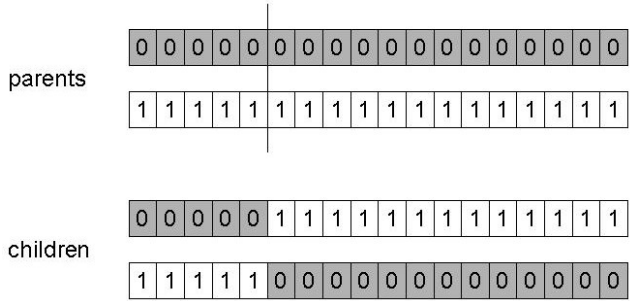
UNIVERSITY OF ABERDEEN

1: **function** Genetic-Algorithm($population$, $Fitness$)
2: **loop**
3: $newpopulation \leftarrow$ empty set
4: **for** $i = 1$ to $|population|$ **do**
5: choose $x$ and $y$ randomly* from $population$ using $Fitness$
6: $child \leftarrow$ Reproduce$(x, y)$
7: **if** small random probability **then** $child \leftarrow$ Mutate$(child)$
8: add child to $newpopulation$
9: $population \leftarrow newpopulation$
10: **if** some individual is fit enough or enough time has elapsed **then**
11: **return** $\arg\max$ {Fitness$(x)$ | $x \in population$}
12: **function** Reproduce(x, y)
13: $c \leftarrow$ random integer `range`$(|x|)$
14: **return** $x[:c] + y[c:]$

To choose $x$, usually use relative fitness: for each $x \in population$, $Pr[choose\ x] = \dfrac{Fitness(x)}{\sum_z Fitness(z)}$

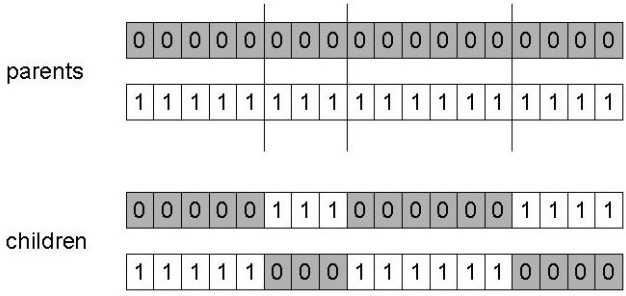
To choose $y$, usually use relative fitness or a uniform distribution

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails (typically with $0.6 < P_C < 0.9$)

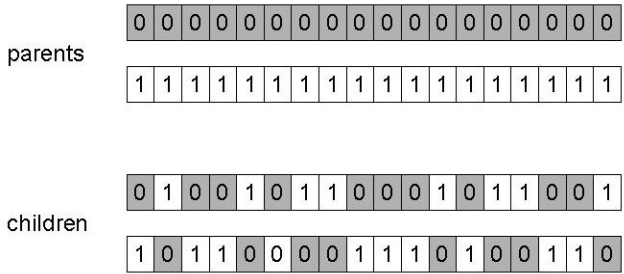


# n-point Crossover

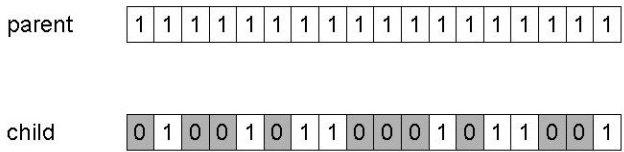
- Choose $n$ random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1 point (still some positional bias)

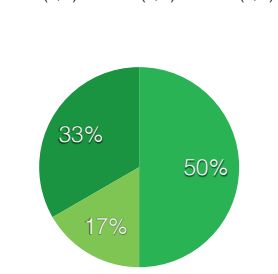


- Assign ‘heads‘ to one parent, ‘tails‘ to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position



- Alter each gene independently with a probability $P_m$
- $P_m$ is called the mutation rate
- Typically between 1/pop_size and 1/chromosome_length






- Main idea: better individuals get higher chance
- Chances proportional to fitness
- Implementation: roulette wheel technique
  - Assign to each individual a part of the roulette wheel
  - Spin the wheel $n$ times to select $n$ individuals

Example

- Fitness(A) = 3
- Fitness(B) = 1
- Fitness(C) = 2

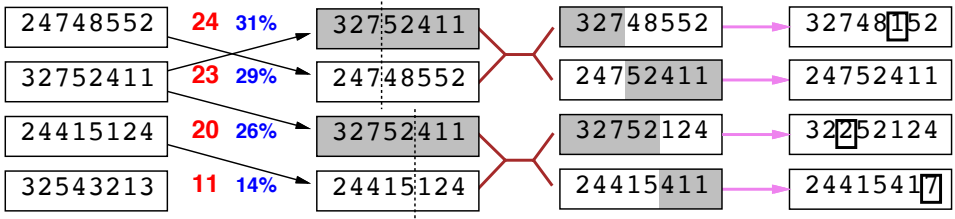
# Crossover VS. mutation?

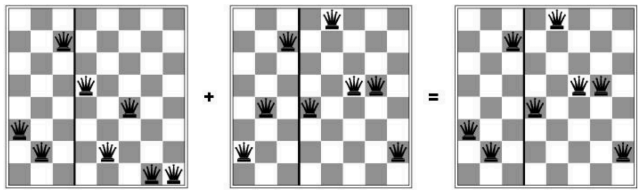
- Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem
- Exploitation: Optimising within a promising area, i.e. using information
- There is co-operation **and** competition between them
  - Crossover is explorative, it makes a big jump to an area somewhere “in between“ two (parent) areas
  - Mutation is exploitative, it creates random small diversions, thereby staying near (in the area of) the parent

# Crossover VS. mutation?

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population
- To hit the optimum you often need a ‘lucky‘ mutation

# Genetic Algorithm Example



- Suppose we want to site three airports in Romania:
  - 6-D state space defined by $(x_1, y_1), (x_2, y_2), (x_3, y_3)$
  - objective function $f(x_1, y_1, x_2, y_2, x_3, y_3)$
    sum of squared distances from each city to nearest airport
- Discretization methods turn continuous space into discrete space
  ,e.g., empirical gradient considers $\pm\delta$ change in each coordinate
- Gradient methods compute $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$
  to increase/reduce $f$, e.g., by $x \leftarrow x + \alpha \nabla f(x)$

- Local Search Algorithms
  - Hill Climbing
  - Simulated Annealing
  - Local beam search
  - Genetic Algorithms

Any Questions.