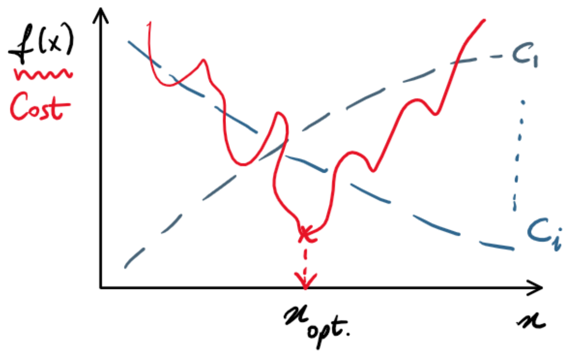
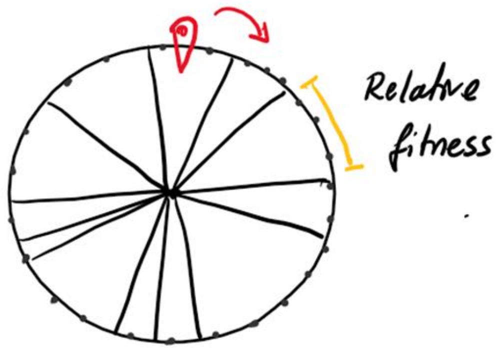
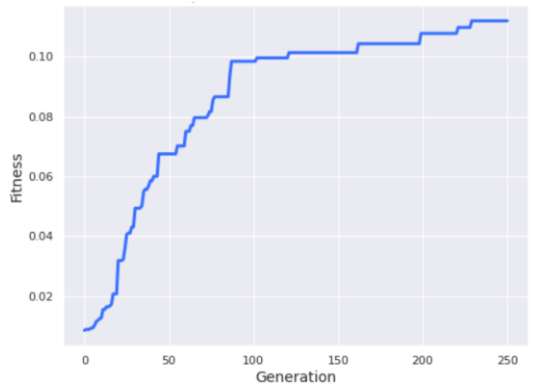
Data Driven Engineering II: Advanced Topics

Genetic algorithms: towards data driven control

Institute of Thermal Turbomachinery
Prof. Dr.-Ing. Hans-Jörg Bauer







DDE: Dynamical Systems

Discovery // Characterization // Simulation

* FACT : Process ⇔ Controller

linked &

Tasks:

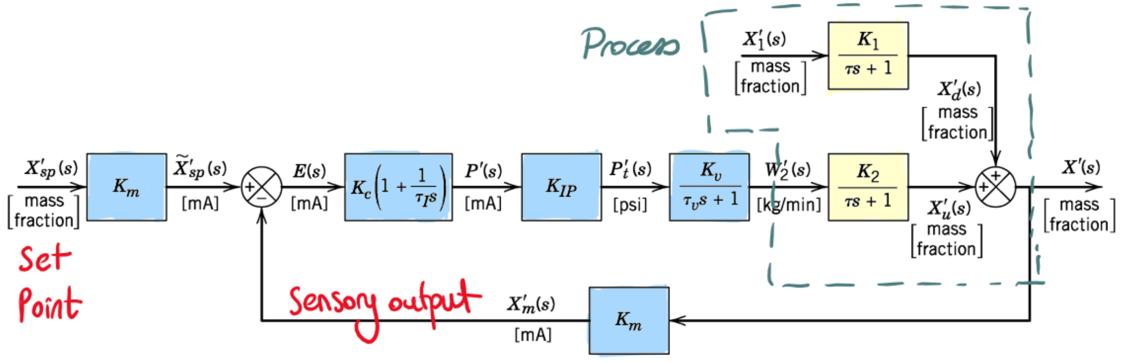
① Create a phy. model I DDE-I

② Create a controller model

③ Coupled optimization

DDE-II

Closed-loop Control System:



DDE ⟺ Optimization

* Optimization landscape
* Evolutionary algorithms
* Genetic algorithms
* Genetic programing

Genetic Algorithms (GAs) have been proven to be highly efficient in data-driven control systems due to several unique attributes. Let's go over some of these reasons:

Global Search: Genetic algorithms perform a global search in the solution space and hence, are less likely to get stuck in local minima compared to certain other optimization methods like gradient descent.

Parallelism: Genetic algorithms search from a population of points rather than a single point. Thus, they intrinsically provide a form of parallelism and can sample the search space more efficiently.

Handling Non-linear, Non-convex Problems: Genetic algorithms do not make any assumptions about the problem being solved and can handle highly complex, non-linear, and non-convex problems, which are often found in data-driven control systems.

No Derivative Information Needed: Unlike other optimization methods, GAs don't require derivative information. This makes them suitable for problems where the derivative is difficult to compute, or for discrete optimization problems.

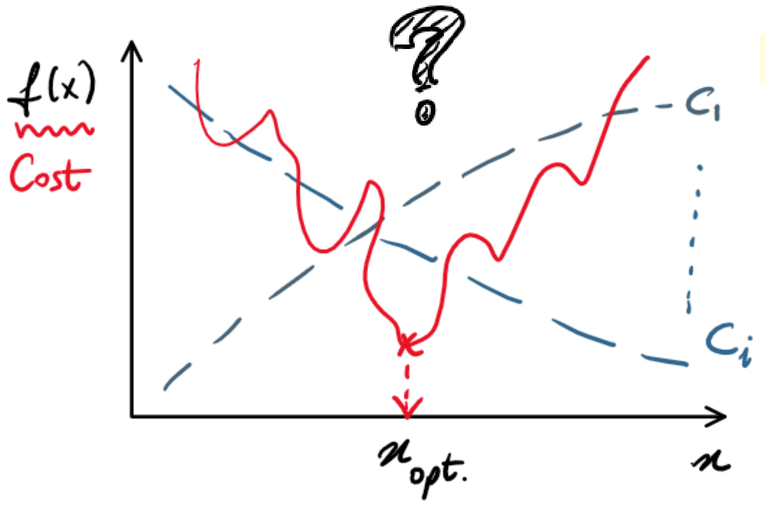
Robustness: Genetic algorithms are relatively insensitive to the specific details of the underlying problem and can therefore provide good solutions to a broad range of problem types.

Adaptability: Genetic algorithms can be easily combined with other AI and machine learning techniques, such as neural networks or fuzzy logic, to create more robust and efficient systems.

Feature Selection: Genetic algorithms can be useful for the purpose of feature selection in machine learning, which is essential for creating efficient data-driven control systems.

Remember, while GAs have these advantages, they're not always the best tool for every task. GAs tend to be computationally expensive and may not be suitable for problems where a quick solution is necessary or when computational resources are limited. They also might not perform as well on problems where the optimal solution requires a specific, nuanced understanding of the problem's characteristics. In those cases, other optimization techniques may perform better.

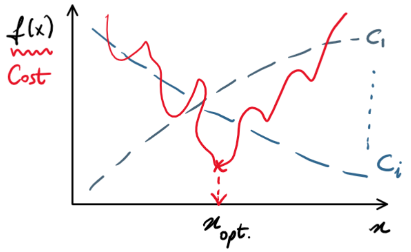
Optimization Problem

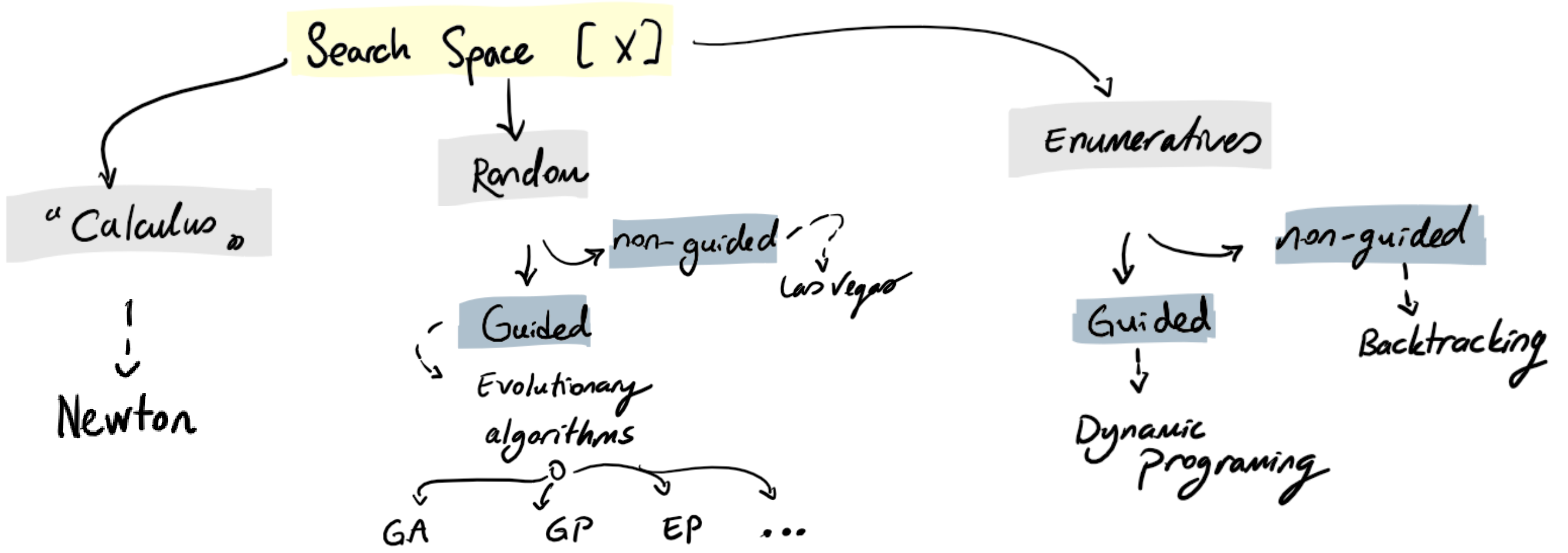


⇨ Competing factors

⇨ Cost function

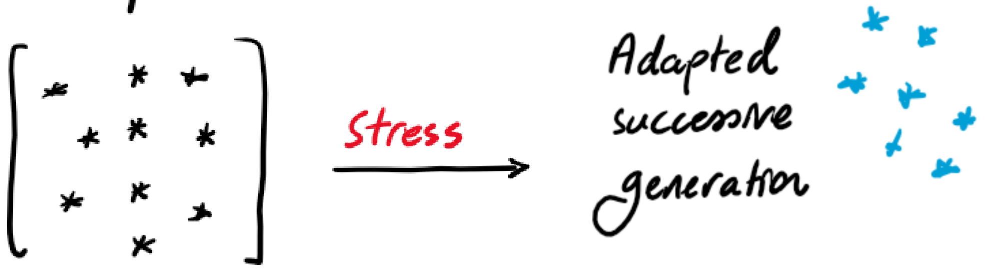
- ☐ Single/multiple variable
- ☐ Static / dynamic
- ☐ Discrete / continuous
- ☐ Constrained / uc
- ☐ Function / Data


f(x)
Cost
C1
Ci
x opt.
x


Search Space [X]
" Calculus "
Newton
Random
non-guided
Las Vegas
Guided
Evolutionary algorithms
GA
GP
EP
...
Enumeratives
non-guided
Backtracking
Guided
Dynamic Programing

Evolutionary Algorithms

* Population-based



Eg Rechenberg 1965, 1973
Airfoil optimization
"Evolutionsstrategie"

* Genetic algorithm: 60s, 70s

- John Holland "Adaptation in Natural and Artificial Systems"

Genetic Algorithms :

⇨ no rigorous definition on GA

$$\begin{tikzcd} & \text{population of chromosomes} \\ \bullet \arrow[ur, bend left] \arrow[r, bend right] \arrow[d, bend right] & \text{selection via a fitness} \\ \text{offsprings} \arrow[r] & \text{crossover \& mutations} \end{tikzcd}$$

The 2006 NASA ST5 spacecraft antenna.
The shape was found by EA

Genetic Algorithms :

⇨ **Selection :** a measure of fitness ⇨ stress over population

↳ not all parents have offsprings (hyperparameter)

↳ individuals not fitting ⇒ eliminated (hp.)

$\downarrow$

Crossover : allow to exchange info among members (hp.)

Mutations : Change a single element in an individual (hp.)

Algorithm of GA :

1. Initialize population
2. Get current fitness (+ filtering)
3. Create offsprings ⟷ cross over
4. Mutations ∅
5. "Survival of the fittest"
   ↳ update the population

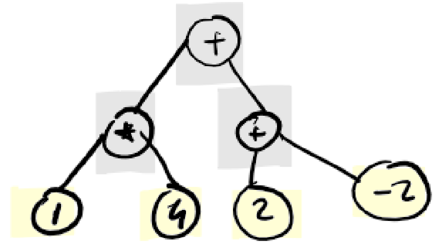
Challanges :

- ☐ Define a fitness function
- ☐ Problem representation
- ☐ Early convergence
- ☐ Calling fitness func. many times
- ☐ Hyperparameter tuning

Gene Encoding

? how to represent individual genes

- ☐ Binary encoding (1101000110 10)
- ☐ Octal encoding [0,7] → 157124613
- ☐ Hexadecimal enc. [0-9, A-F] → 9CE7
- ☐ Value encoding [1, 16 5.24 2.41 2.54]
  [N N S W E]
- ☐ Tree encoding



Breeding

(i) Select parents (ii) Create offsprings (iii) update population

Selection

~ higher the fitness; higher the chance ~

* affects the conversion rate

(a) proportionate-based ~ relative fitness

(b) ordinal-based ~ rank within population

In the context of Genetic Algorithms (GAs), selection refers to the process of choosing individuals (solutions) from the current population to form the next generation. Different selection strategies can be used to determine how these individuals are selected. Let's discuss proportionate selection and ordinal-based selection:

1. Proportionate Selection: This is a selection process in which each individual in the population is assigned a fitness value, and the probability of an individual being selected is proportional to its fitness value. In other words, fitter individuals have a higher chance of being selected for reproduction. This method reflects the principle of "survival of the fittest." However, one downside is that highly fit individuals can dominate the selection process, which can potentially lead to a lack of diversity and premature convergence to suboptimal solutions.

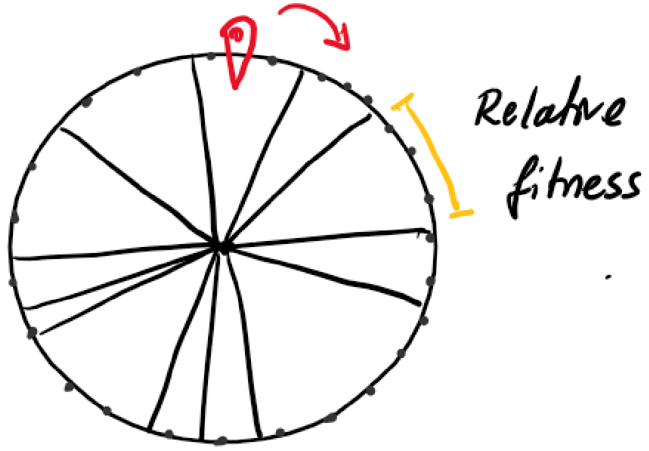
2. Ordinal-based Selection (or Rank Selection): In this approach, individuals are selected based on their rank rather than their fitness value. The population is first sorted according to fitness, and then each individual is assigned a rank. The selection probability of each individual is then determined based on its rank. This can help maintain diversity in the population as it reduces the chance that highly fit individuals will dominate the selection process. However, it may also slow down the convergence process because it does not distinguish significantly different fitness values among individuals.

In summary, both selection strategies have their advantages and disadvantages, and the choice between them depends on the specific requirements of the problem at hand. Proportionate selection can converge faster but risks losing diversity, while ordinal-based selection maintains diversity but can be slower to converge.

Selection

~ higher the fitness ; higher the chance ~

Roulette - Wheel



'Rotate' N times for N parents

⇩

≪ Parent Pool ≫

* Not a strong selection
* Noisy

Selection

~ higher the fitness ; higher the chance ~

Rank Selection

Roulette ⇒ biased

↳ less chance for potential solutions

* Rank population $1 \longrightarrow N$ (best)

* Slow convergence
* Diversity preserved

[Ranked]

⇓

Potential parents

* random selection
  ↳ pairs
    ↳ tournament

Selection

~ higher the fitness ; higher the chance ~

Boltzmann Selection

$$P = \exp\left[-(f_{max} - f_i)/T\right]$$

$$T = T_0 (1-\alpha)^k$$

$$k = (1 + 100g/G)$$

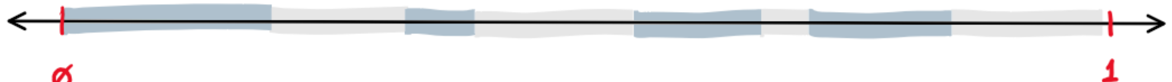
- * $T_0 = [5, 100]$
- * $\alpha = [0, 1]$
- * $g :=$ generation i
- * $G :=$ max. gen. allowed
- * $f_i =$ fitness

} Gradually narrow down search space

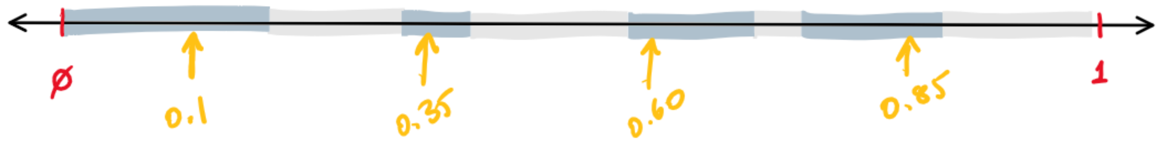
Selection

~ higher the fitness ; higher the chance ~

Stochastic Universal Sampling



* 4 pointers $\Rightarrow$ $x_i = 1/4 = 0.25$
* $R_1 = [0, 25] \Rightarrow 0.10$



Selection

~ higher the fitness ; higher the chance ~

Random Selection

* Randomly select N parents.
* Much more noisy

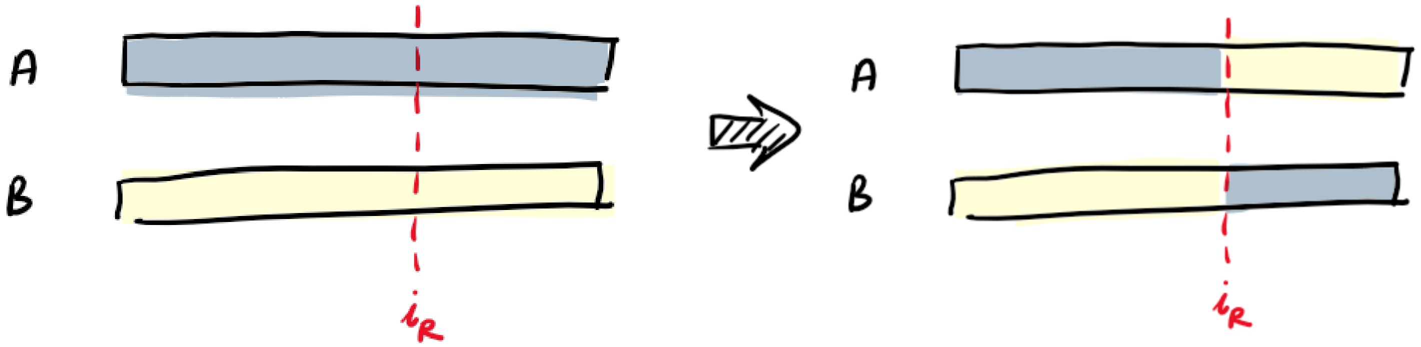
Elitism

* Best E chromosomes are directly passed to parent pool.
* For the rest ; other methods are used.

Crossover <=> Recombination

* 2 Parents ⇨ 1 offspring
* applied to the parent pool.

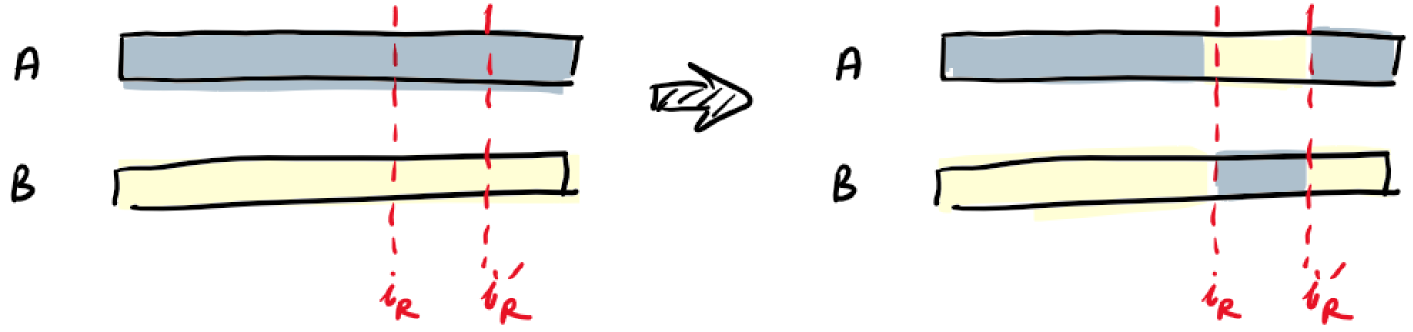
Single point crossover



Crossover <=> Recombination

* 2 Parents ⇨ 1 offspring
* applied to the parent pool.

Two points crossover



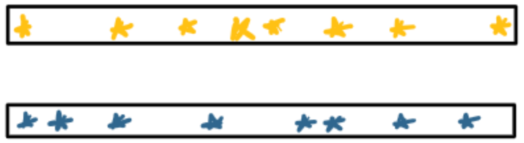
Crossover $\Leftrightarrow$ Recombination

* 2 Parents $\Rightarrow$ 1 offspring   * applied to the parent pool.

Uniform Crossover

✓ Parent I

✓ Parent II



✓ Child I $N/2 + N/2 = N$ genes

✓ $\text{Child}_{II} := 1 - \text{Child}_{I}$

Others

$\Rightarrow$ Three parent crossover

$\Rightarrow$ Shuffle Crossover

$\Rightarrow$ Custom f.

Mutation :

* Typically after breeding

* Remedy to exit from local minimum
  - ↳ extending search space
  - ↳ increasing diversity

Mutation is an important operation in Genetic Algorithms (GAs) that introduces diversity into the population, helping to prevent premature convergence to local optima. It works by randomly modifying parts of an individual in the population. The key idea behind mutation is to allow exploration of the solution space beyond the current population.

The role of mutation in GAs is to maintain diversity in the population and to allow the algorithm to escape local minima. However, it's important to carefully balance the mutation rate. Too much mutation can cause the search process to become random, while too little mutation can cause the algorithm to get stuck in local optima. The ideal mutation rate often depends on the specific problem and may require some trial-and-error to determine.

Mutation :

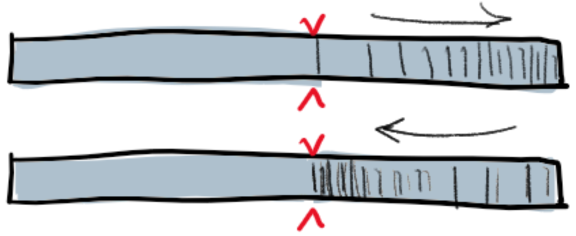
Flipping

Flipping: This is commonly used in binary-coded GAs. In a flip mutation, a bit in the binary representation of an individual is flipped from its current state. If the bit is '1', it is changed to '0' and vice versa. This introduces a new point in the search space and can help to escape local minima.

* Binary; $0 \Rightarrow 1$, $1 \Rightarrow 0$
* Random numbers $\Rightarrow$ # events / indices

Reversing

Reversing (or Inversion Mutation): Another method often used in permutation-based GAs. In this type of mutation, a subset of the sequence is selected and its order is reversed. This can be useful in problems where the relative ordering of values is important.



Interchanging

Interchanging (or Swap Mutation): This is often used in permutation-based GAs, such as the ones used for solving the Travelling Salesman Problem. In swap mutation, two positions are selected randomly in the individual, and the values at these positions are swapped. This also allows exploration of the search space beyond the immediate vicinity of the current individuals.

* Two random genes are exchanged.

! Mutation rate $\Rightarrow$ 100%
GA $\Rightarrow$ Random search

Population Update ⇔ Replacement

* $\underbrace{[\text{Population}] \rightarrow [\text{Parents}] \rightarrow [\text{Offsprings}]}_{\text{Which ones are kept?}}$

Children can replace parents. Kill parents, keep children. We are losing some information.

Both Parents

* Children ⇒ Parents

This are some random decisions we have to make.

Random Replacement

* Children ⇒ randomly chosen individuals

Weak Parent Replacement

* Child → weaker parent

Convergence

* Max. generation
* Elapsed time
* Track fitness ○ → Best individual
  → worst individual
  → $\sum f_i$ or $\bar{f_i}$

If the best individual is above this threshold; Stop it.

Algorithm of GA :

1. Initialize population
2. Get current fitness (+filtering)
3. create offsprings ⟷ crossover
4. Mutations
5. "Survival of the fittest"
   ↳ update the population

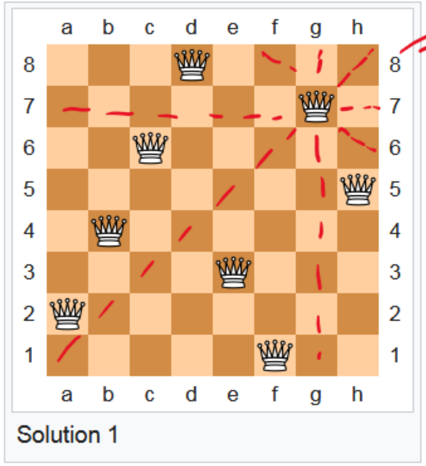
Examples

① Password guessing

ⓘ String matching $\Rightarrow$ Fitness Score

ⓘⓘ Gene pool $\Rightarrow$ abc...z

② "8 Queens"



Solution 1

Fitness score;

4 lines; 32 $\Rightarrow \phi$

Chromosomes;

[: :] $\Rightarrow$ locations

"N" Queen Problem

Examples

③ Solving system of linear eqs.

$$\left.\begin{aligned} 2x + y - 2z &= 3 \\ x - y - z &= 0 \\ x + y + 3z &= 12 \end{aligned}\right\} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = ?$$

* Ch. $\Rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

* Fitness $\Rightarrow \sum f_i \Rightarrow 0$

④ PyGAD $\Rightarrow$ Train a NN via GA

* Regression $\Rightarrow$ Airfoil Noise Problem
* Chromosomes $\Rightarrow$ Weights of NN model
* fitness $\Rightarrow$ MSE / MAE

Training a neural network (NN) via a genetic algorithm (GA) and using a backpropagation algorithm are fundamentally different processes, each with its own unique characteristics, advantages, and disadvantages. Here are the main differences:

1. Learning Approach:

Backpropagation is a gradient-based optimization algorithm. It learns by iteratively adjusting the network's weights in the direction that reduces the error. This direction is found by computing the gradient of the loss function with respect to the weights. It works exceptionally well when we have differentiable loss function and weights.

Genetic Algorithms are evolutionary algorithms that learn via a process of selection, crossover (recombination), and mutation. They treat each set of weights in the network as an individual "genome". These genomes are evolved across generations based on a fitness function. GAs are global search techniques that search through the weight space, and do not require gradient information.

2. Convergence:

Backpropagation tends to converge faster because it uses information about the error surface (i.e., the gradient). However, it can get stuck in local minima if the error surface is not convex.

Genetic Algorithms may converge more slowly because they do not use gradient information. However, they are more capable of escaping local minima because of their global search nature and mutation operation, which allows them to explore new regions in the weight space.

3. Problem Suitability:

Backpropagation is generally better suited to problems where the error surface is smooth and differentiable. It's usually the go-to method for training large, deep neural networks due to its efficiency.

Genetic Algorithms can be more suitable for problems where the error surface is rugged, discontinuous, or non-differentiable. They can also handle discrete and combinatorial optimization problems, which backpropagation cannot.

4. Computationally Expensive:

Backpropagation is generally less computationally expensive than GA, especially for larger networks, as it directly exploits the error information to update the weights.

Genetic Algorithms can be more computationally expensive, as they require maintaining and evaluating a population of solutions, instead of a single solution.

It's worth noting that while these methods are quite different, they can sometimes be used in a complementary manner. For instance, GAs can be used to initialize the weights of a neural network, which is then fine-tuned using backpropagation. This can sometimes provide a good balance between global exploration (from the GA) and local exploitation (from backpropagation).