# Lecture 11: Evolutionary Learning

## CSE5012: Evolutionary Computation and Its Applications

**Xin Yao**

**CSE, SUSTech**

**15 May 2023**

## Review of the Last Lecture

- **Genetic Programming (GP)**
  - **Tree Representation**
  - **Preparatory Steps**
  - **Operators on Trees**
  - **E.g., Evolving Boolean N-multiplexer Functions using GP**
  - **Multi-Objective GP**

## Outline of This Lecture

## Introduction

1. **Evolution is an extremely general and powerful tool for designing learning machines.**
2. **Artificial evolution has been used in evolving**
   - **rule-based systems**
   - **artificial neural networks**
   - **fuzzy logic systems**
   - **hidden Markov models**
   - **gene regulatory networks**
   - **finite state machines**
   - **many other learning machines.**
3. **The key issues here include representation and fitness evaluation.**

## Outline of This Lecture

## Two Major Approaches to Evolutionary Learning

**Depending on what an individual represents,**

**Michigan Approach:** **Each individual represents a single rule (or just a part of a system). The whole population represents the complete (learning) system.**

**Pitt Approach:** **Each individual represents a *complete* system.**

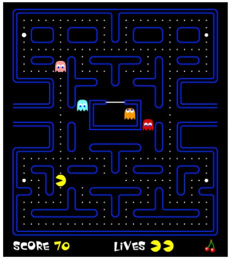
Question

In the lecture of Week 8: Co-evolution, I have asked you about the potential issues of each approach. Do you have an answer now?

## Outline of This Lecture

## Example of Rule-based Systems

### A rule-based agent for playing Pacman



IF $(ghost_y > pacman_y)$ and $(ghost_x == pacman_x)$ THEN forward

IF $(ghost_y < pacman_y)$ and $(ghost_x == pacman_x)$ THEN back

IF $(ghost_y == pacman_y)$ and $(ghost_x < pacman_x)$ THEN left

IF $(ghost_y == pacman_y)$ and $(ghost_x > pacman_x)$ THEN right

## Evolving Rule-based Systems

**Key issues to be resolved:**

1. **Encoding rules and the entire system.**
2. **Variation operators, e.g., recombination, mutation, etc.**
3. **Incorporation of domain knowledge if available.**
4. **Fitness evaluation.**

## Encoding Rules

**There are many different methods. Here is the simplest one. Given a set of reactive rules as follows:**

IF $c_{11}$, $c_{12}$, ..., $c_{1n}$ THEN $a_1$;
IF $c_{21}$, $c_{22}$, ..., $c_{2n}$ THEN $a_2$;
......
IF $c_{m1}$, $c_{m2}$, ..., $c_{mn}$ THEN $a_m$.

**We can encode them in a single individual (chromosome) as follows:**

$$c_{11}, c_{12}, \ldots, c_{1n}, a_1, c_{21}, c_{22}, \ldots, c_{2n}, a_2, \cdots, c_{m1}, c_{m2}, \ldots, c_{mn}, a_m$$

## Discussions

- **Although the previous example assumed the fixed-length representation, we can use variable chromosome length if necessary. In the fixed-length representation, we can allow dummy rules with empty conditions and actions.**
- **The order of listing rules in the chromosome is not essential. So the chromosome should be a set-based, rather than an order-based representation.**
- **Representation is closely linked to variation operators.**

## Recombination

**There can be many different types. Here are two examples:**

**Two parent recombination:** **Select randomly two subsets of rules from two parents and swap them.**

**Global discrete recombination:** **The rule in the offspring is selected uniformly at random from all individuals in the population.**

## Mutation

**Here are two examples:**

**Between rules:** **Swap randomly subsets of conditions of two rules (within the same individual).**

**Within rules:** **Change at random one or more conditions or action (within the same rule) to another value.**

## Incorporating Domain Knowledge

**In many applications, it is important to use available domain knowledge. There are many ways in which domain knowledge can be incorporated into evolutionary learning, especially in representation and variation operators. For example:**

1. **Domain knowledge can be used to design specialisation operators and generalisation operators.**
   - **Specialisation operators are mutation operators that add more conditions to an existing rule(s), while**
   - **generalisation operators will do the opposite.**

2. **Domain knowledge can also be used to bias recombination if we could assign some kind of "local fitness" to each individuals.**

## Current Practice in Pitt Approach

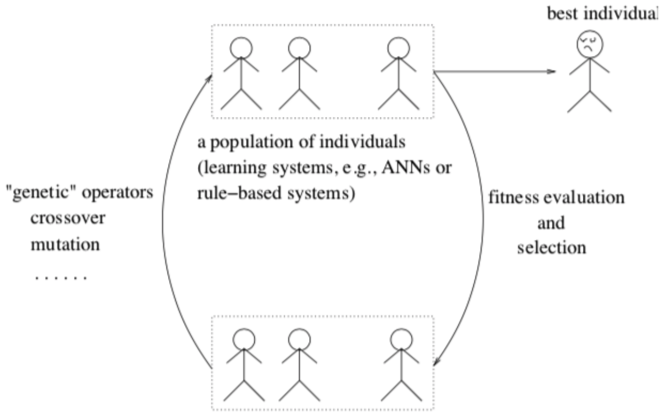


Figure 1: A general framework for Pitt style evolutionary learning.

## Population

- **Michigan systems: a single rule-set which represents the problem solution.**
- **Pitt systems: a collection of multiple competing rule-sets.**

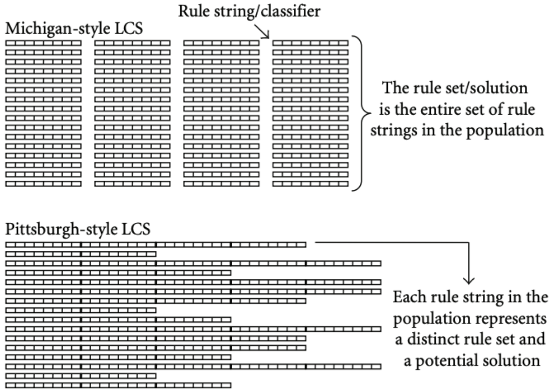

Michigan-style LCS
Rule string/classifier
The rule set/solution is the entire set of rule strings in the population
Pittsburgh-style LCS
Each rule string in the population represents a distinct rule set and a potential solution

Figure 2: Figure 4 of [2]: Michigan versus Pitt-style systems.

## Fitness Evaluation

1. **Based on the training error.**
   $\rightarrow$ **Weakness: May overfit noisy training data.**

2. **Based on the training error and complexity (regularisation), i.e.,**

$$\frac{1}{fitness} \propto error + \alpha * complexity$$

   $\rightarrow$ **Weakness: Difficult to set $\alpha$ and difficult to choose $complexity$.**

3. **Based on a separate validation set.**
   $\rightarrow$ **Weakness: Need more training data and may be time-consuming.**

## Learning Classifier Systems (LCS)

- "***LCSs employ two biological metaphors; evolution and learning...learning guides the evolutionary component to move toward a better set of rules.***"
  **– Dam, H. H., Abbass, H. A., Lokan, C., & Yao, X. (2008). Neural-based learning classifier systems. *IEEE Transactions on Knowledge and Data Engineering*, 20(1), 26-39.**

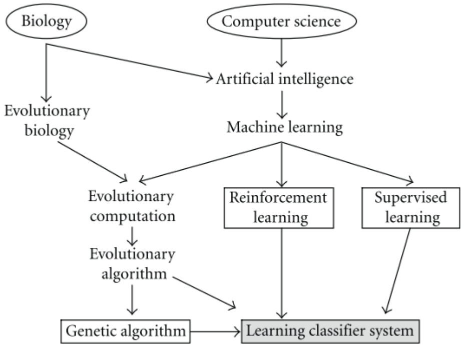


Figure 3: Figure 1 of [2]: Field tree-foundations of the LCS community.

## Learning Classifier Systems (LCS) [1]

- **When dealing with a complex system, evolving a population of rules which collectively model the system is favourable[2].**
- **The tripartite LCS structure:**
  - **Rule discovery system:** Genetic Algorithm (GA).
  - **Credit assignment system: update each rule's utility (it's *strength*) using the feedback obtained from the interacts with the environment (the source of input data for LCS). E.g., temporal differential updates, monte carlo updates.**
  - **Production system: find the rules which apply or match the current state of the problem.**
- **Fitness: classification accuracy.**
- **LCS $=$ policy learning $+$ generalisation.**

**Tim Kovacs. *Strength or accuracy: credit assignment in learning classifier systems*. Springer Science & Business Media, 2012**

## Example: A Minimal Classifier System (MCS)

- **Input data: 4-digit binary number, encoded from an instance in the environment.**
- **A classifier is made up of**
  1. **a condition: a string of characters from the ternary alphabet $0$, $1$ and $\#$,**
  2. **an action: a binary string,**
  3. **an associated fitness parameter $\{F\}$: indicate how good a given classifier is.**
- **Covering: "If none of the rules in the population match the input, a covering operator generates a rule with a matching condition and a random action" [2].**

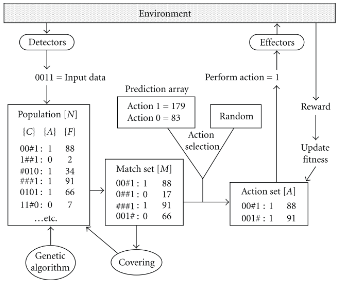


Figure 4: Figure 3 of [2]: MCS algorithm-an example iteration.

## GA for Searching

**One generation:**

1. **Evaluate the fitness of all individuals (i.e., rules) in the current population.**
2. **Select parent rules from the population using fitness proportionate selection.**
3. **Crossover and/or mutate parents to form offspring.**
4. **Select next population from the current population $+$ offspring proportionally to fitness.**

## Learning

- **Supervised learning and reinforcement learning.**
- **Exploitation-exploration dilemma:**
  - **identify classifiers that help obtaining future rewards**
  - **or encourage searching for better rules.**

## A Generic LCS

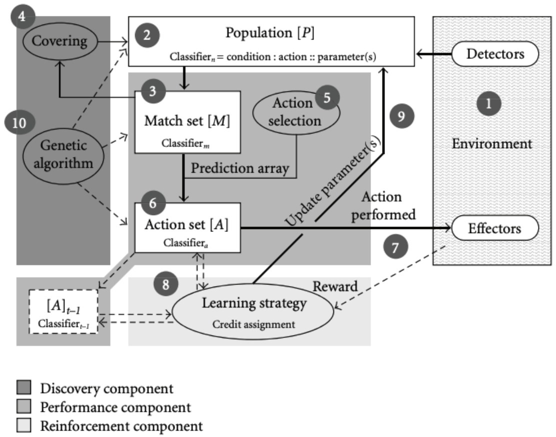


Figure 5: Figure 2 of [2]: A Generic LCS-the values 1-10 indicate the typical steps included in a single learning iteration of the system Thick lines indicate the flow of information, thin lines indicate a mechanism being activated, and dashed lines indicate either steps that do not occur every iteration, or mechanisms that might occur at different locals.

## Outline of This Lecture

## Evolving Neural Networks

- Although all biological NNs are evolved, most artificial NNs (ANNs) are trained through gradient descent algorithms.
- Most ANN learning algorithms concern only with connection weights (including biases). The architecture is manually designed.
- There are efforts made towards optimising ANN architectures using constructive and pruning algorithms. However, these are greedy algorithms and can be trapped in a poor local optimum.
- Design of a near optimal ANN architecture can be formulated as a search problem in the architecture space. The optimality of architectures forms a surface/landscape. Evolutionary algorithms are better candidates for searching on this surface.
- We can evolve NN weights and architectures simultaneously.

$\Longrightarrow$ **Neural Architecture Search (NAS) using EC.**

## Encoding ANN Architectures

**Direct Encoding:** **All the details, i.e., every connection and node of an architecture are specified by the chromosome.**

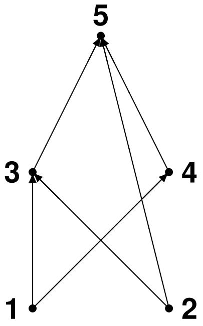
**Indirect Encoding:** **Only the most important parameters of an architecture, such as the number of hidden layers and hidden nodes in each layer are encoded. Other details about the architecture are left to the training process to decide.**

1. **Parametric representation.**
2. **Developmental rule representation.**
3. **Others.**

## The Direct Encoding Scheme

- **In the direct encoding scheme, each connection in an architecture is directly specified by its binary representation.**
- **An $N \times N$ matrix $C = (c_{ij})_{N\times N}$ can represent an architecture with $N$ nodes, where $c_{ij}$ indicates presence or absence of the connection from node $i$ to node $j$.**
- **Each such matrix has a direct one-to-one mapping to the corresponding architecture. The binary string representing an architecture is just the concatenation of rows (or columns) of the matrix.**

## Example 1: Feed-forward Networks



(a)

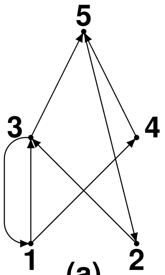
$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b)

`0110 101 01 1`

(c)

## Example 2: Recurrent Networks



(a)

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

(b)

00110 00100 10001 00001 01000

(c)

## A Typical Cycle of Evolving ANN Architectures

1. **Decode each individual in the current generation into an architecture.**
2. **Train each ANN with the decoded architecture starting from different sets of random initial connection weights.**
3. **Calculate the fitness of each individual.**
4. **Reproduce a number of individuals for each individual in the current generation based on its fitness.**
5. **Apply variation operators to the individuals generated above and obtain the next generation.**

## Fitness Evaluation

1. **Based on training error.**
2. **Based on validation error.**
3. **Based on training/validation error and network complexity.**
   3.1 **based on the number of connections.**
   3.2 **based on some information criterion, such as AIC, MDL or MML.**

## Discussions

1. **The direct encoding scheme is simple and straightforward to implement.**
2. **It is suitable for detailed design of architectures.**
3. **It may facilitate rapid generation and optimisation of tightly pruned interesting designs that no one has hit upon so far.**
4. **It does not scale well for very large architectures.**

## The Indirect Encoding Scheme

1. **Parametric Representation.**
2. **Developmental Rule Representation.**
3. **Other Representations.**

## Parametric Representation

**Describe an architecture using a set of parameters, such as the number of hidden layers, the number of hidden nodes in each layer, the number of connections between two layers, etc., without specifying each node-to-node connection.**

**Advantage: shorter representation.**

**Disadvantage: limited search space.**

## Developmental Rule Representation

- **Instead of optimising architectures directly, a set of rules that are used to generate architectures will be optimised.**
- **This method has several advantages, such as more compact representation and better scalability.**
- **It also has its own disadvantages though, e.g., noisy fitness evaluation.**

## Evolving Rules: An Example

- **A modified version of graph generation system was used which includes a set of graph generation rules.**
- **Each rule consists of a left-hand side (LHS) which is a non-terminal symbol and a right-hand side (RHS) which is a $2 \times 2$ matrix with either terminal or non-terminal symbols.**
- **Each rule is represented by five *allele* positions corresponding to five symbols in a rule in a genotype. Each position in a genotype can take the value of an symbol in the range from "A" to "p".**
- **The $16$ rules with "a" to "p" on the left-hand side and $2 \times 2$ matrices with only $1$'s and $0$'s on the right-hand side are pre-defined and do not participate in evolution.**

## Example: Rules

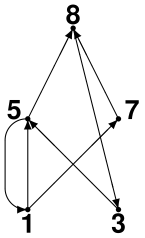
$$S \longrightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad A \longrightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix} \quad B \longrightarrow \begin{pmatrix} i & i \\ i & a \end{pmatrix}$$
$$C \longrightarrow \begin{pmatrix} i & a \\ a & c \end{pmatrix} \quad D \longrightarrow \begin{pmatrix} a & e \\ a & e \end{pmatrix} \quad a \longrightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$
$$c \longrightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad e \longrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad i \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Figure 6: Examples of some developmental rules used to construct a connectivity matrix. $S$ is the initial symbol (or state).

# Example: Actual Development

$$S \qquad \begin{matrix} A & B \\ C & D \end{matrix} \qquad \begin{matrix} a & a & i & i \\ a & a & i & a \\ i & a & a & e \\ a & c & a & e \end{matrix}$$

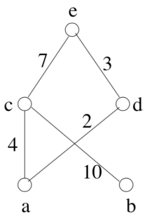
$$\begin{matrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$



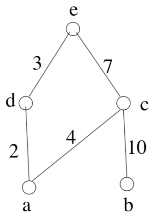
## The Permutation Problem

- **Also known as the competing convention problem.**
- **It is caused by the many to one mapping from genotypes to phenotypes since two different genotypes may be mapped to the same phenotype.**
- **It creates plateau in the search space and makes search inefficient.**

# Example: Permutation Problem



| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0000 | 0000 | 0100 | 0010 | 0000 |
| b | 0000 | 0000 | 1010 | 0000 | 0000 |
| c | 0000 | 0000 | 0000 | 0000 | 0111 |
| d | 0000 | 0000 | 0000 | 0000 | 0011 |
| e | 0000 | 0000 | 0000 | 0000 | 0000 |



| | a | b | d | c | e |
|---|---|---|---|---|---|
| a | 0000 | 0000 | 0010 | 0100 | 0000 |
| b | 0000 | 0000 | 0000 | 1010 | 0000 |
| d | 0000 | 0000 | 0000 | 0000 | 0011 |
| c | 0000 | 0000 | 0000 | 0000 | 0111 |
| e | 0000 | 0000 | 0000 | 0000 | 0000 |

## Problem: Noisy Fitness Evaluation

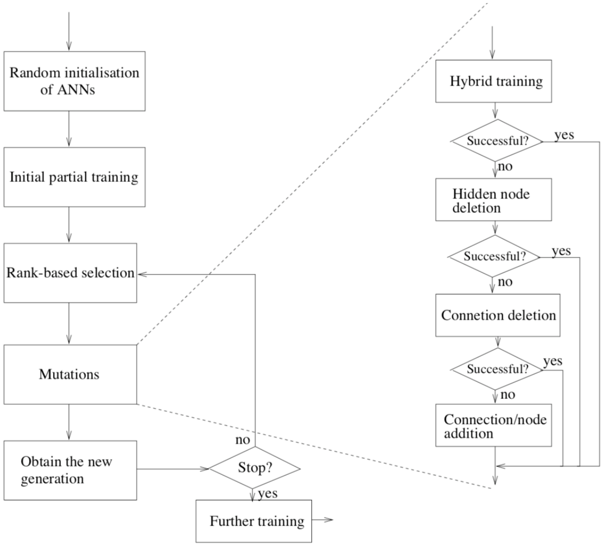
**The fitness evaluation of NN architectures is noisy.**

1. **We want to evaluate architectures without weights (genotypes).**
2. **We actually evaluate ANN with weights (phenotypes) which**
   - **are initialised at random;**
   - **are trained with a particular algorithm.**
3. **The evaluation would be even noisier if the developmental rules are not deterministic in the case of an indirect encoding scheme.**

## Behaviour Disruption by Genetic Operators

- **The behavioural difference between the parent and the offspring may be very large after genetic operations.**
- **The useful information learned previously may be lost because of such a large disruption to the NN behaviour.**

## An Evolutionary System — EPNet



## Mutation

1. **Five mutation operators are used in our system: partial training, node deletion, connection deletion, connection addition, and node addition.**

2. **They are applied sequentially. Only one of them will be used in each generation.**

3. **Partial training is the only mutation operator used to change NN's connection weights. It uses a hybrid algorithm to train the NN for a *fixed* number of epochs. Such training does not guarantee NN's convergence. Hence the training is *partial*.**

## Connection Deletion

**Connections are probabilistically selected for deletion according to**

$$test(w_{ij}) = \frac{\sum_{t=1}^{T} \xi_{ij}^{t}}{\sqrt{\sum_{t=1}^{T} (\xi_{ij}^{t} - \overline{\xi}_{ij})}} \tag{1}$$

**where**

- $\xi_{ij}^{t} = w_{ij} + \Delta w_{ij}^{t}(w)$**,**
- $\overline{\xi}_{ij}$ **denotes the average over the set** $\xi_{ij}^{t}$**,** $t = 1, \ldots, T$**,**
- $\Delta w_{ij}(w) = -\eta[\partial L_t / \partial w_{ij}]$ **is the weight update,**
- $L$**(**$L = \sum_{t=1}^{T} \sum_{i=1}^{n} |Y_i(t) - Z_i(t)|$**) is a linear error function with respect to example** $t$ **and weight** $w_{ji}$**.**

## Connection and Node Addition

1. **Connections are probabilistically added according to Eq.(1). They are selected from all connections with zero weight.**

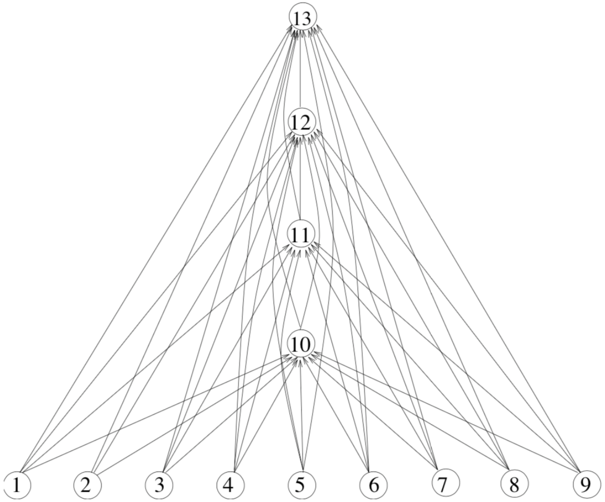
2. **Hidden nodes are added through node splitting. Two nodes obtained by splitting an existing node $i$ have the same connections as the existing node. The weights of these new nodes have the following values:**

$$w_{ij}^1 = w_{ij}^2 = w_{ij},\ i \geq j,$$

$$w_{ij}^1 = (1+\alpha)w_{ij}, w_{ij}^2 = -\alpha w_{ij},\ i < j$$

**where $\mathrm{w}$, $\mathrm{w}^1$ and $\mathrm{w}^2$ are the weight vectors, and $\alpha$ is a mutation parameter.**

## The 9-Parity Problem — Architectures



# The 9-Parity Problem — Weights and Biases

| | T | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 10 | -12.35 | -12.19 | 12.38 | -12.19 | -12.20 | 12.23 | -12.19 |
| 11 | -4.12 | 6.04 | 0 | 6.04 | 6.04 | -6.34 | 6.04 |
| 12 | 7.05 | 6.78 | -7.13 | 6.78 | 6.79 | -6.96 | 6.79 |
| 13 | 0 | 7.89 | -7.76 | 7.89 | 7.89 | -8.04 | 7.89 |

| | | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|
| 10 | | 12.23 | -12.12 | -12.20 | 0 | 0 | 0 |
| 11 | | -6.34 | -26.16 | 6.05 | 0 | 0 | 0 |
| 12 | | -6.96 | -9.59 | 6.79 | 26.70 | -16.45 | 0 |
| 13 | | -8.04 | -10.71 | 7.89 | 30.71 | -18.99 | -17.02 |

## Outline of This Lecture

## Optimal Control Problems with Switching Costs I

- **Switching costs: the cost (time, energy, money) required to switch from one state/product/service to another.**
- **Notation:**
  - $t_i$ **is the initial time and** $t_f$ **is the final time.**
  - $u(t)$ **is the control function.**
  - **The variation of** $u(t)$ **is** $\overset{t_f}{\underset{t_i}{\bigvee}} = \underset{\mathcal{P}}{sup} \sum_{i=0}^{n-1} ||u(t_{i+1}) - u(t_i)||$ **where** $\mathcal{P}$ **ranges over all partitions** $t_i = t_0 < t_1 < \cdots < t_n = t_f$**.**
  - **For a vector function** $u \subset R^m$**, the total variation is,**

$$\overset{t_f}{\underset{t_i}{\bigvee}} u = \sum_{i=0}^{m} \overset{t_f}{\underset{t_i}{\bigvee}} u^i(t).$$

  - $u(t)$ **resembles a step function, the variation of** $u(t)$ **is the size of each step, or switch, in the control function.**

## Optimal Control Problems with Switching Costs II

- **The general control system can be written in vector form as:**

$$x'(t) = f(x(t), u(t)), \text{ in } R^n \tag{2}$$

  - **with initial condition** $x(t_i) = x_0$.
  - $f(x(t), u(t))$ **is a given vector function, makes Eq.** (2) **a system of ordinary differential equations involving an unknown control function** $u$ **and an unknown state function** $x$.

- **Formulation of a simple optimal control problem:**

$$\min\{g(x(t_f)) + \bigvee_{t_i}^{t_f} Du(t)\}, \tag{3}$$

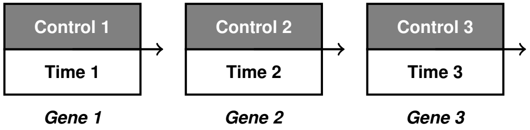
**where**

  - $g(x(t_f))$ **is some function which depends on the final state of the system to be controlled** (2)
  - **and** $D$ **is a matrix of constants (in the general vector problem).**

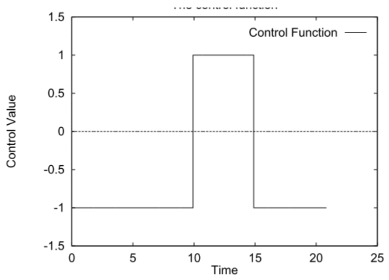
$\rightarrow$ ***A complex and nonconvex optimisation problem*.**

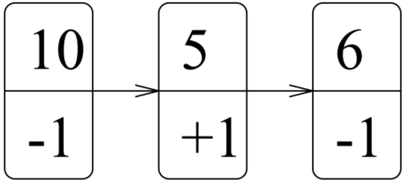
## Chromosome Representation

- **The $[t_i, t_f]$ is discretised into $N$ equal parts.**
- **Each chromosome represents a complete control function.**
- **Each gene represents a switch point, where the control function changes from one allowable state to the state coded on the gene.**
  - **Each gene is a $(time, control)$ pair, where $time$ is an interval.**
  - **The sum of $time$ values in a chromosome equals to $t_f - t_i$.**
    **$\rightarrow$ Each chromosome is of variable length.**
    **$\rightarrow$ length(chromosome) $=$ #switches.**



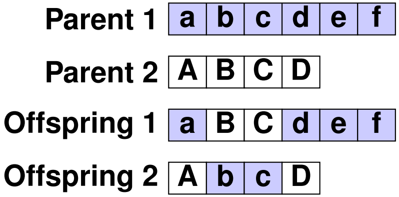
## Chromosome Representation: Example





## Crossover

- **Similar to uniform crossover**
  - **was found to be advantageous to the population when compared with mutation alone**
  - **and also provided better performance than one-point or two-point crossover.**
- **When the two solutions are of different length, the remaining genes go to the candidate who received the gene preceding the extra genes.**



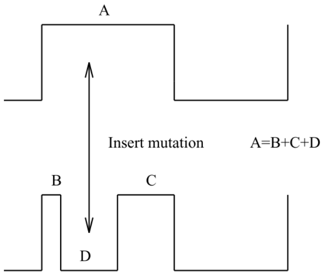
## Blend

- **Linear combinations of two chromosomes: non-uniform arithmetical crossover operator**
  - **Child 1:**
$$(\alpha A + (1-\alpha)a) \quad (\alpha B + (1-\alpha)b) \quad (\alpha C + (1-\alpha)c) \quad \cdots$$
  - **Child 2:**
$$(\alpha a + (1-\alpha)A) \quad (\alpha b + (1-\alpha)B) \quad (\alpha c + (1-\alpha)C) \quad \cdots$$
  - **For chromosomes of different length, the remaining genes are left untouched at the end of the appropriate child.**
  - $\alpha$ **is selected between 0 and 1 with a uniform distribution.**

## Mutation

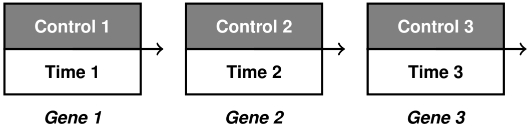
- **Either of the following two:**
  - **Mutate time values: with a probability of $0.005$, a Cauchy random number is generated and added to the existing integer time component, and the decimal part of the number is truncated.**
  - **Mutate control values: with a probability of $0.01$, replace the original state with any of the other allowable control states uniformly at random.**

## Problem Specific Operator



- **Insertion operator:**
  - **Similar to a standard mutation, except that it affects 2 genes.**
  - **New switches in the control function can be tested faster.**
  - **Applied with a probability of $0.005$ in the experiments.**

## Problem Specific Issues

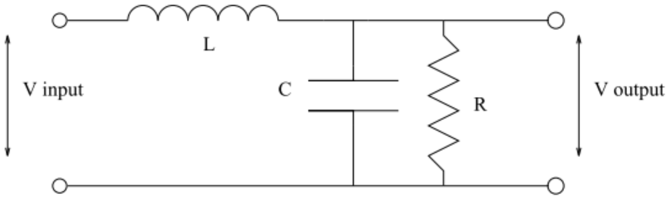


- **After the operations, the sum of $time$ of a new chromosome may not equal to $t_f - t_i$:**
  - **If $> t_f - t_i$, then truncate the chromosome to code for the correct time,**
  - **If $< t_f - t_i$, then extend the time coded by the last gene to code for the correct time.**

## Main Steps

1. **Initialise 100 chromosomes, each with a randomly determined #genes $\in [20, 60]$.**
2. **Solve the control system using the control function obtained from each chromosome in the population. Calculate the cost associated with this control sequence and order the population inversely by this cost.**
3. **Take the ordered population and use a linear ranking scheme with a bias of $1.5$ to select two chromosomes for breeding.**
4. **Apply either crossover or blending to the two chromosomes selected with probabilities $0.4$ and $0.4$.**
5. **Mutated time values with probability $0.005$ or control values with probability $0.01$, or insert a switch with probability $0.005$.**
6. **Repeat from step 3 until 100 children have been formed.**
7. **Select the 100 chromosomes from the new ordered population of children and parents, using a linear ranking scheme.**
8. **Repeat from step 2 until 2000 generations are achieved.**

## Application: Controlling LCR Circuits with a Finite Number of Discrete Voltage Values



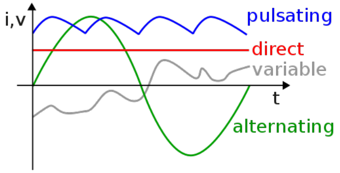


Figure 7: Left: RLC filter (滤波电路). Right: direct current vs. alternating current.

- **Switched amplifier design.**
- **The output stage of such an amplifier is modelled as an RLC circuit, where the $v_{input}$ voltage is the control, and the $v_{output}$ voltage is the output, which we want to track a sine wave.**

## Recall

- **Cresistor (R, 电阻器):**
$$v_R = Ri \tag{4}$$
- **Inductor (L, 电感器):**
$$v_L = L\frac{di}{dt} \tag{5}$$
- **Capacitor (C, 电容器):**
$$v_C = \int_0^t i d\tau \tag{6}$$
- **Kirchhoff Circuit Laws (基尔霍夫定律):**
  - **Kirchhoff's voltage law (KVL):**
$$\sum_i U_i = 0 \tag{7}$$
  - **Kirchhoff's current law (KCL):**
$$\sum_i I_i = 0 \tag{8}$$

## Differential Equation relating $v_{input}$ and $v_{output}$

**Together with $I_L = I_C + I_R$, Eqs. (4), (5), (6) and (7):**

$$\begin{aligned}
v_{input} &= v_L + v_{output} && \textbf{\textit{according to Eq.}}\ (7) \\
&= L\frac{dI_L}{dt} + v_{output} && \textbf{\textit{according to Eq.}}\ (5) \\
&= L\frac{d(I_C + I_R)}{dt} + v_{output} \\
&= L\frac{dI_C}{dt} + L\frac{dI_R}{dt} + v_{output} \\
&= L\frac{d(C\frac{dv_C}{dt})}{dt} + L\frac{d(\frac{v_R}{R})}{dt} + v_{output} && \textbf{\textit{according to Eqs.}}\ (6)\ \textbf{\textit{and}}\ (4) \\
&= LC\frac{d^2v_C}{dt^2} + \frac{L}{R}\frac{dv_R}{dt} + v_{output} && (9)
\end{aligned}$$

**As $v_C = v_R = v_{output}$, Eq. (9) becomes**

$$v_{input} = LC\frac{d^2v_{output}}{dt^2} + \frac{L}{R}\frac{dv_{output}}{dt} + v_{output}. \qquad (10)$$

## Case 1: A Simple Control Problem with Switching Costs

- **Try and track an output voltage of $v_{output} = 0$ with an output allowable control set, $v_{input} \in \{-1, +1\}$. The cost criteria is**

$$\min\{\int_{t_i}^{t_f} v_{output}^2 dt + \epsilon \bigvee_{t_i}^{t_f} v_{input}\},$$

**where**

  - $t_i = 0$, $t_f = 16$, $\epsilon = 0.025$,
  - $L = C = 1$ **and** $R = 2$,
  - **with initial conditions** $v_{output}(0) = 1$, $v'_{output}(0) = 1$.

- **The following ordinary differential equation problem is derived:**

$$\min\{v_3 + \epsilon \bigvee_{0}^{16} v_{input}\}, \tag{11}$$

**where**

$$\begin{aligned} v'_1 &= v_2, & v_1(0) &= 1 \\ v'_2 &= v_{input} - v_1 - \frac{1}{2}v_2, & v_2(0) &= 1 \\ v'_3 &= v_1^2, & v_3(0) &= 0 \\ v_{output} &= v_1. \end{aligned}$$

## Experimental Results for Case 1

| $N$ | Average | Best | Worst |
|---|---|---|---|
| 2400 | 2.28334 | 2.28294 | 2.28384 |
| 3200 | 2.28468 | 2.27962 | 2.29265 |

Table 1: Results (over 10 trials) from the constant output switched amplifier problem. Dynamic Programming found a solution with a cost of 2.2896.

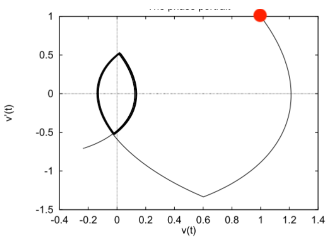


Figure 8: The phase portrait for a typical solution with a discretisation of 2400 for the constant output circuit. Red point refers to the initial conditions $v_{output}(0) = 1$, $v'_{output}(0) = 1$.

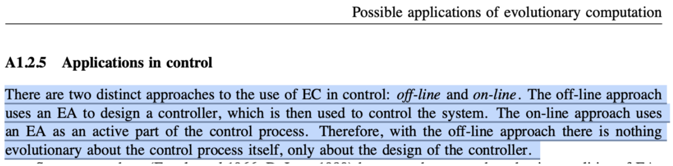
## Case 2: Generating a Sine Wave

- **The output of the amplifier is to be a sine wave given by $sin(\omega t)$ using an input voltage of only $v_{input} \in \{-1, 0, +1\}$ over a time span $T$ of $40$ units.**
- **Parameters:** $\omega = 0.7$**,** $v_0(0) = 0, v_0'(0) = 1$ **and** $N = 1600$**.**

# Experimental Results for Case 2

| $N$ | Average | Best | Worst |
|---|---|---|---|
| 1600 | 0.857015 | 20.84449 | 0.869541 |

Table 2: Results (over 10 trials) for the sine wave generating circuit. Dynamic Programming found a solution with a cost of 0.810619.



Possible applications of evolutionary computation

**A1.2.5 Applications in control**

There are two distinct approaches to the use of EC in control: *off-line* and *on-line*. The off-line approach uses an EA to design a controller, which is then used to control the system. The on-line approach uses an EA as an active part of the control process. Therefore, with the off-line approach there is nothing evolutionary about the control process itself, only about the design of the controller.

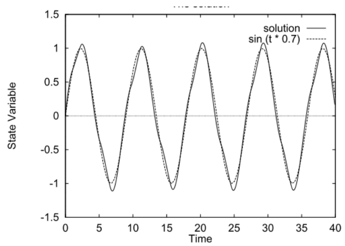


Figure 9: The input (control) and output functions of the best control found for the sine wave generator. Left: the control function. Right: the solution.

## Observations

- **The EA achieved extremely competitive results, and in some cases, better than previously known results.**
- **In many optimal control problems, finding controls which obtain better minima of the costs associated with the systems can often be of great value.**
- **The EA will be particularly useful for those optimal control problems which are difficult to specify mathematically.**

## Outline of This Lecture

## Summary and Remarks

1. **Chromosome representation has an important impact on the success of evolutionary design.**
2. **Transform a problem from one space to another one may bring in certain benefits.**
3. **When the mapping between genotypes and phenotypes is not one-to-one, care must be taken in fitness evaluation.**
4. **Design of a model (a learning system) can be combined with the optimisation of model parameters in order to exploit potential symbiosis.**

## Outline of This Lecture

## Reading List for This Lecture

1. **R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: A complete introduction, review, and roadmap," Journal of Artificial Evolution and Applications, vol. 2009, p. 1, 2009.**
2. **T. Kovacs, "Strength or accuracy: credit assignment in learning classifier systems," Springer Science & Business Media, 2012.**
3. **X. Yao and Md. M. Islam, "Evolving artificial neural network ensembles," IEEE Computational Intelligence Magazine, 3(1):31-42,February 2008.**
4. **X. Yao, "Evolving artificial neural networks,"** ***Proceedings of the IEEE*****, 87(9):1423-1447, September 1999.**
5. **X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks,"** ***IEEE Transactions on Neural Networks*****, 8(3):694-713, May 1997.**

## Reading List for The Next Lecture

1. **T. Bäck, D. B. Fogel, and Z. Michalewicz (eds.), *Handbook of Evolutionary Computation*, IOP Publ. Co. & Oxford University Press, 1997. (Sections B2.1-2.3, B2.7)**
2. **A. E. Eiben and G. Rudolph, "Theory of evolutionary algorithms: A bird eye view," *Theoretical Computer Science*, 229(1-2):3-9, 1999.**
3. **P. S. Oliveto, J. He and X. Yao, "Time Complexity of Evolutionary Algorithms for Combinatorial Optimization: A Decade of Results," *International Journal of Automation and Computing*, 4(3):281-293, July 2007.**
4. **D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, 1(1):67-82, April 1997.**