# Lecture 6: Population diversity, Niching, Speciation & Co-Evolution

## CS408/CSE5012: Evolutionary Computation and Its Applications

Xin Yao

CSE, SUSTech

27 March 2020

# Review of the Last Lecture

- Permutation-based Optimisation
- Travelling Salesman Problems
- Cutting Stock Problems

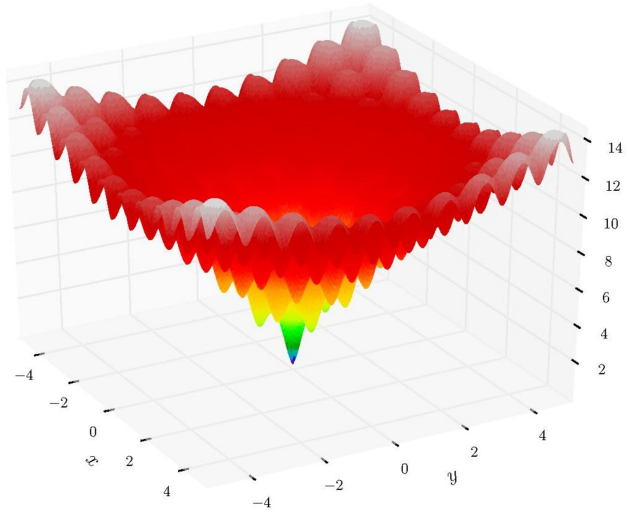
# Outline of This Lecture

Population diversity, Niching, Speciation

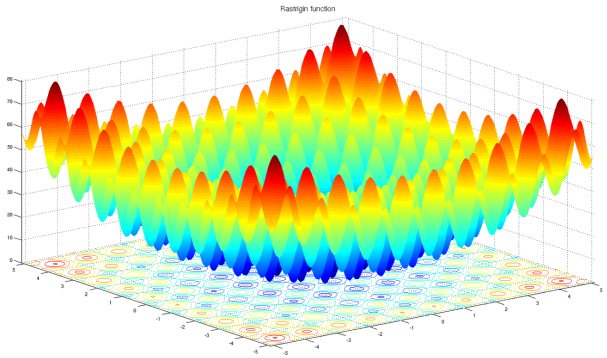
- Why Niching
- Different Niching Techniques
- Fitness Sharing
- Crowding and Speciation

Co-Evolution

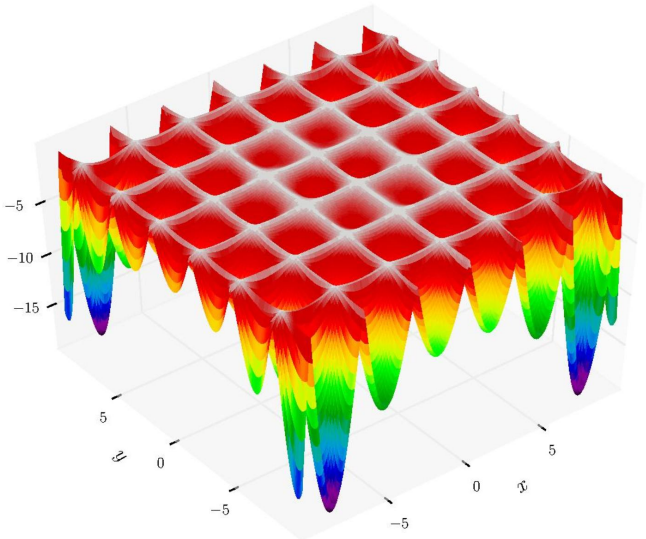
# Multimodality



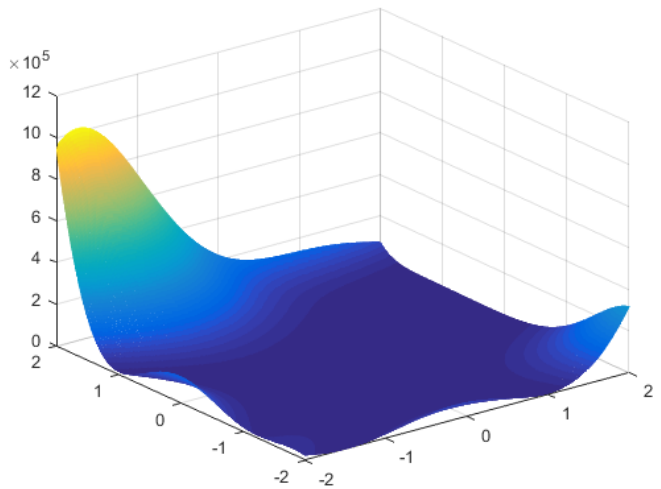

(a) Ackley Function




(b) Rastrigin Function




(c) Easom Function




(d) Goldstein-Price Function

# Exploitation-Exploration Dilemma for EAs

**Exploitation** of learned information by investigating regions containing high fitness solutions discovered.

**Exploration** aiming to uncover new regions with high fitness.

# Niching

**What Is It?** It refers to the formation of groups of individuals in a population. Individuals within a group are similar to each other, while individuals from different groups are very different from each other.

**Why Do We Need It?** Niching is useful in a number of cases.

- It helps to encourage and maintain population diversity, and thus explore a search space better.
- It helps to optimise multiple objectives simultaneously.
- It helps to learn an ensemble of machine learning systems that cooperate.
- It helps to simulate complex and adaptive systems, e.g., artificial ecological systems.

# Different Niching Techniques

Major categories of niching techniques are:

- *Sharing* [1], also known as fitness sharing,
- *Crowding*, and
- *Speciation*.

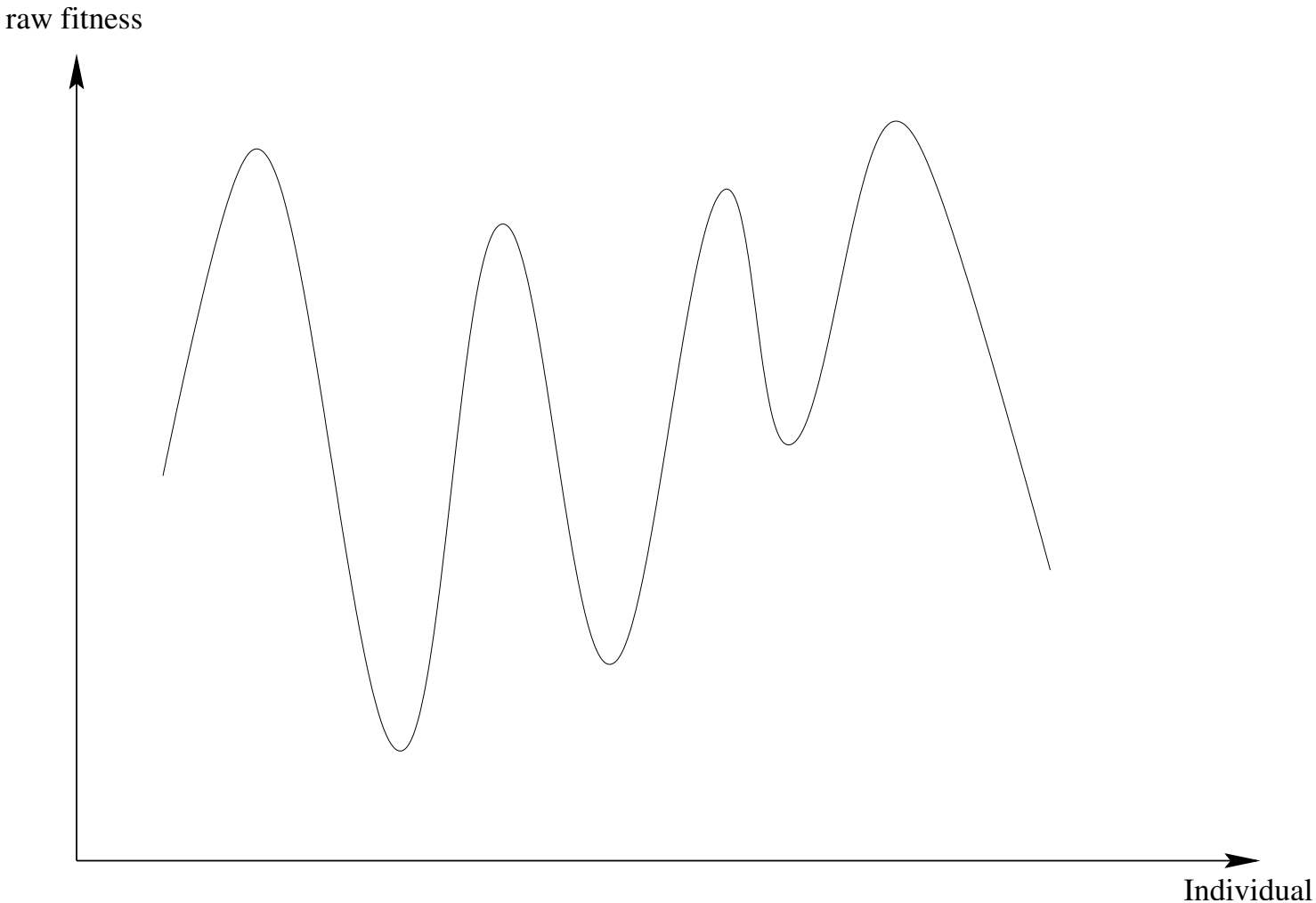
Other niching techniques include *sequential niching* and *parallel hill-climbing*.

# (Explicit) Fitness Sharing: Introduction

- Fitness sharing transforms the raw fitness of an individual into the shared one (usually lower).
- The idea is that there is only limited and fixed amount of "resources" (i.e., fitness value) available at each niche. Individuals occupying the same niche will have to share the resources.

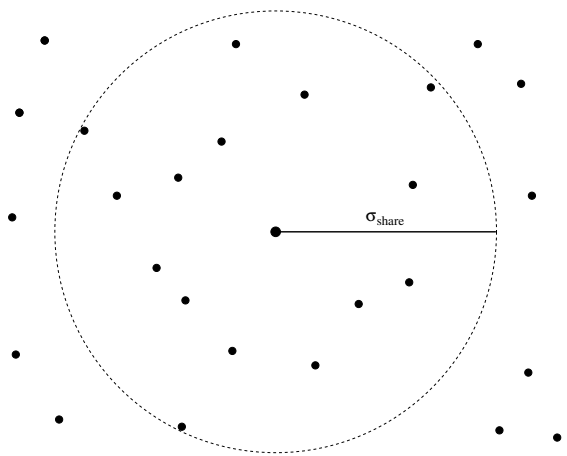
# Fitness Sharing: An Example

Sharing can be explained by a simple example of locating multiple peaks on a multimodal function in a single run.



# Sharing Radius

- Sharing radius, $\sigma_{share}$, defines the niche size. Individuals within this radius will be regarded as being similar to each other and thus need to share fitness.
- The similarity between two individuals is defined by the distance between them.
  - For example, the similarity between two binary strings can be defined by their Hamming distance. Thus, the "distance metric" used here is Hamming distance.



# Sharing Function and Shared Fitness

- The sharing function can be defined as

$$sh(d_{i,j}) = \begin{cases} 1 - \left(\frac{d_{i,j}}{\sigma_{share}}\right)^{\alpha}, & \text{if } d_{i,j} < \sigma_{share}, \\ 0, & \text{otherwise,} \end{cases}$$

where

  - $d_{i,j}$ is the distance between individuals $i$ and $j$.
  - $\alpha$ determines the shape of sharing function.
    - The function is linear when $\alpha = 1$.
    - When increasing $\alpha$, $f_{share}$ reduces more rapidly with distance.
  - $\sigma_{share}$ is share radius, it decides
    - how many niches can be maintained in a population and
    - the granularity (粒度) with which different niches can be discriminated.

- The shared fitness of individual $i$ can be defined as

$$f_{share}(i) = \frac{f_{raw}(i)}{\sum_{j=1}^{\mu} sh(d_{i,j})},$$

where $\mu$ is the population size.

# Fitness Sharing: Discussions

1. Sharing can be done at genotypic or phenotypic levels. For example,
   - Genotypic level: Hamming distance, i.e., the number of positions at which the corresponding chromosomes are different, $d_H(\mathbf{x}_1, \mathbf{x}_2) = \sum_{k=1}^{d} \mathbf{I}(x_1(k) \neq x_2(k))$, where $x_1(k)$ refers to the $k$-th coordinate of $\mathbf{x}_1$. E.g., $d_H(1001, 0110) = 4$.
   - Phenotypic level: Euclidean distance, $d_H(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{k=1}^{d} (x_1(k) - x_2(k))^2}$. E.g., $d_E(1001, 0111) = 2$.

   The key is how to define the "distance" (i.e., similarity).

2. Sharing radius, $\sigma_{share}$, can be difficult to set. *Why?*

3. Population size is particularly important in sharing. *Why?*

4. A population may not be able to locate all peaks. It may lose peaks located during evolution. *Why?*

5. Fitness sharing needs extra computation time.

6. Fitness sharing as described above may not work well.

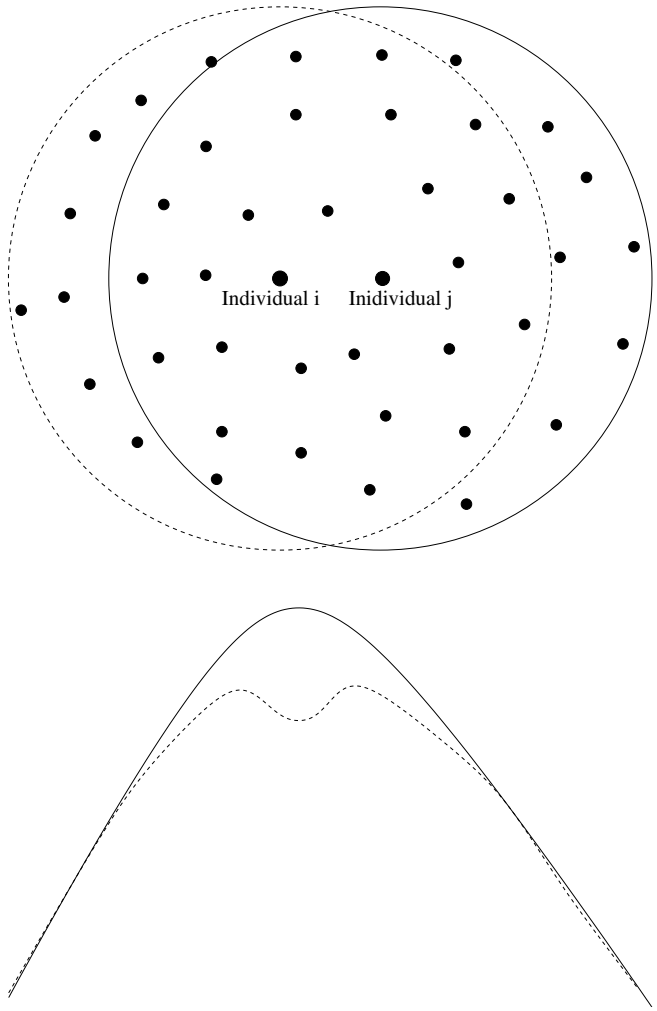
# Fitness Scaling in Sharing

To make fitness sharing work, raw fitness scaling is often needed as follows:

$$f_{share}(i) = \frac{(f_{raw}(i))^{\beta}}{\sum_{j=1}^{\mu} sh(d_{i,j})},$$

where $\beta > 1$ is a scaling factor.

# Why Do We Need It?




$\beta$ needs to be sufficiently large. *Why?*

# A Dilemma

With a low scaling factor: Individuals won't converge to the real optima because they are not attractive.

With a high scaling factor: "Super individuals" in initial populations may dominate the population quickly. The evolution may not be able to locate all peaks.

Solutions?

- Large population.
- Soft selection.
- Anneal $\beta$, i.e., staring from $\beta = 1$ and increasing it gradually.

# Implicit Fitness Sharing: Background

- The idea comes from the immune system: antibody which best matches an invading antigen receives the payoff for that antigen.
- Similar situation appears in games: a strategy that scores the best against a test case receives payoff.
- Implicit fitness sharing has often been used in machine learning.

# Implicit Fitness Sharing: Algorithm

Assume we are dealing with a machine learning problem.

For each *case* $i$ to be solved, do the following $C$ times:

1. Select a sample of $\sigma$ individuals from the population.
2. Find the individual in the sample that achieves the best performance for solving *test case* $i$.
3. This best individual (and this one only) receives the payoff. Ties are broken evenly, i.e., payoff will be shared evenly among all best individuals if they have the same best performance.

# Implicit Fitness Sharing: Discussion

1. It has been shown that implicit fitness sharing and (explicit) fitness sharing have the same theoretical basis.

2. A larger $C$ often leads to better sharing performance, but more time-consuming.

# Comparison Between the Two Sharing Methods

1. Implicit fitness sharing covers optima more comprehensively even when those optima have small basins of attraction, *if the population size is large enough for a species to form at each optimum*.

2. Explicit fitness sharing can find the optima with larger basins of attraction and ignore the ones with smaller bases, *if the population size is not large enough to cover all optima*.

*Why?*

# Niching and Speciation

1. We will use the two terms interchangeably although some people distinguish between them.

2. Niching is concerned more with locating peaks (locating basins of attraction), while speciation is more focused on converging to the actual peaks.

# What Is Crowding

- Crowding techniques insert new individuals into the population by replacing similar individuals.
- Crowding techniques strive to maintain the preexisting diversity of a population.
- Crowding techniques do not modify fitness.

# Deterministic Crowding

$P(0) \longleftarrow initialise();$
FOR $t \longleftarrow 1$ TO $g$ DO
    $P(t) \longleftarrow shuffle(P(t-1));$
    FOR $i \longleftarrow 0$ TO $\mu/2 - 1$ DO
        $p_1 \longleftarrow a_{2i+1}(t);$
        $p_2 \longleftarrow a_{2i+2}(t);$
        $\{c_1, c_2\} \longleftarrow recombine(p_1, p_2);$
        $c_1' \longleftarrow mutate(c_1);$
        $c_2' \longleftarrow mutate(c_2);$
        IF $[d(p_1, c_1') + d(p_2, c_2')] \leq [d(p_1, c_2') + d(p_2, c_1')]$ THEN
            IF $f(c_1') > f(p_1)$ THEN $a_{2i+1}(t) \longleftarrow c_1'$ FI;
            IF $f(c_2') > f(p_2)$ THEN $a_{2i+2}(t) \longleftarrow c_2'$ FI;
        ELSE
            IF $f(c_2') > f(p_1)$ THEN $a_{2i+1}(t) \longleftarrow c_2'$ FI;
            IF $f(c_1') > f(p_2)$ THEN $a_{2i+2}(t) \longleftarrow c_1'$ FI;

# Discussions

- Capable of niching, i.e., locating and maintaining peaks.
- Minimal replacement error.
- Few parameters to tune.
- Fast because of no distance calculations.
- Population size must be large enough.
- Should use full crossover, i.e., crossover rate $= 1.0$.

# Crowding Methods in General

- Unlike sharing methods, crowding methods do not allocate individuals proportional to peak fitness. Instead, the number of individuals congregating around a peak is largely determined by the size of that peak's basin of attraction *under crossover*.
- Similarity can be measured at either genotypic or phenotypic levels.

# Speciation in a Narrow Sense

Speciation in a narrow sense focuses search within a peak.

- A speciation method restricts mating to similar individuals and discourages mating of individuals from different species.
- In order to apply such a speciation method, individuals representing each species must be found first. The speciation method cannot be used independently.
- Niching and speciation are complementary.
- Similarity can be measured at either genotypic or phenotypic levels.

# Mating Restriction: Use Tags

Each individual consists of a tag and a functional string.

| # 1 # 0 | 10010 | 1010 ... ... | ... ... | ... ... | 101 |
|---|---|---|---|---|---|
| template | tag | functional string | | | |

- Tags participate in crossover and mutation, but not fitness evaluation.
- Templates can also be used.
- This method has been shown to be effective for multi-modal function optimisation.
- Only individuals with the same tag are allowed to mate.

# Mating Restriction: Use Distance

- Define a threshold parameter, $\sigma_{mate}$.
- Two individuals are allowed to mate only when their distance is smaller than $\sigma_{mate}$.
- EAs with niching and mating restriction were found to distribute the population across the peaks better than those with sharing alone.

Mating restriction is always applies during crossover.

# Fitness Sharing by Speciation

- Use tags to identify species (peaks).
- For a given problem, let $k$ be the number of different tags. Let $\{S_0, S_1, \ldots, S_{k-1}\}$ be $k$ species of individuals and $\|\cdot\|$ be the cardinality of the set. Then,

$$f_{share}(i) = \frac{f_{raw}(i)}{\|S_j\|}, \quad i \in S_j, \quad j = 0, 1, \cdots, k-1$$

- Recombination occurs only among individuals with the same tag.
- A tag can be mutated.
- No distance is used here.
- This is actually sharing plus mating restriction.

# Summary So Far (I)

1. Niching techniques enable us to find multiple peaks simultaneously in evolution.
2. Every niching technique has its own "niche." There is no single best niching method for all problems.
3. All fitness sharing techniques transform fitness values.
4. Population size becomes an important parameter when fitness sharing is used.

# Summary So Far (II)

**Fitness Sharing** modifies fitness.

- (explicit) fitness sharing
- implicit fitness sharing
- fitness sharing with mating restriction

**Crowding** is about replacement strategies.

- deterministic crowding

**Speciation** in a narrow sense occurs during recombination. It is all about mating restriction.

- by tags
- by distances

# Outline of This Lecture

Population diversity, Niching, Speciation
- Why Niching
- Different Niching Techniques
- Fitness Sharing
- Crowding and Speciation

Co-Evolution

# What Is Co-Evolution

The fitness of one individual depends on other individuals.

- The fitness landscape is not fixed, but coupled.
- The same individual may have different fitness in different populations.

Co-Evolution can be regarded as a kind of fitness landscape coupling where adaptive moves by one individual will deform landscapes of others, e.g., in prey-predator problems.

# Different Types of Co-Evolution

Based on the number of populations involved:

Inter-population Co-Evolution has two or more populations.

Intra-population Co-Evolution has only one population.

Based on the relationship among individuals:

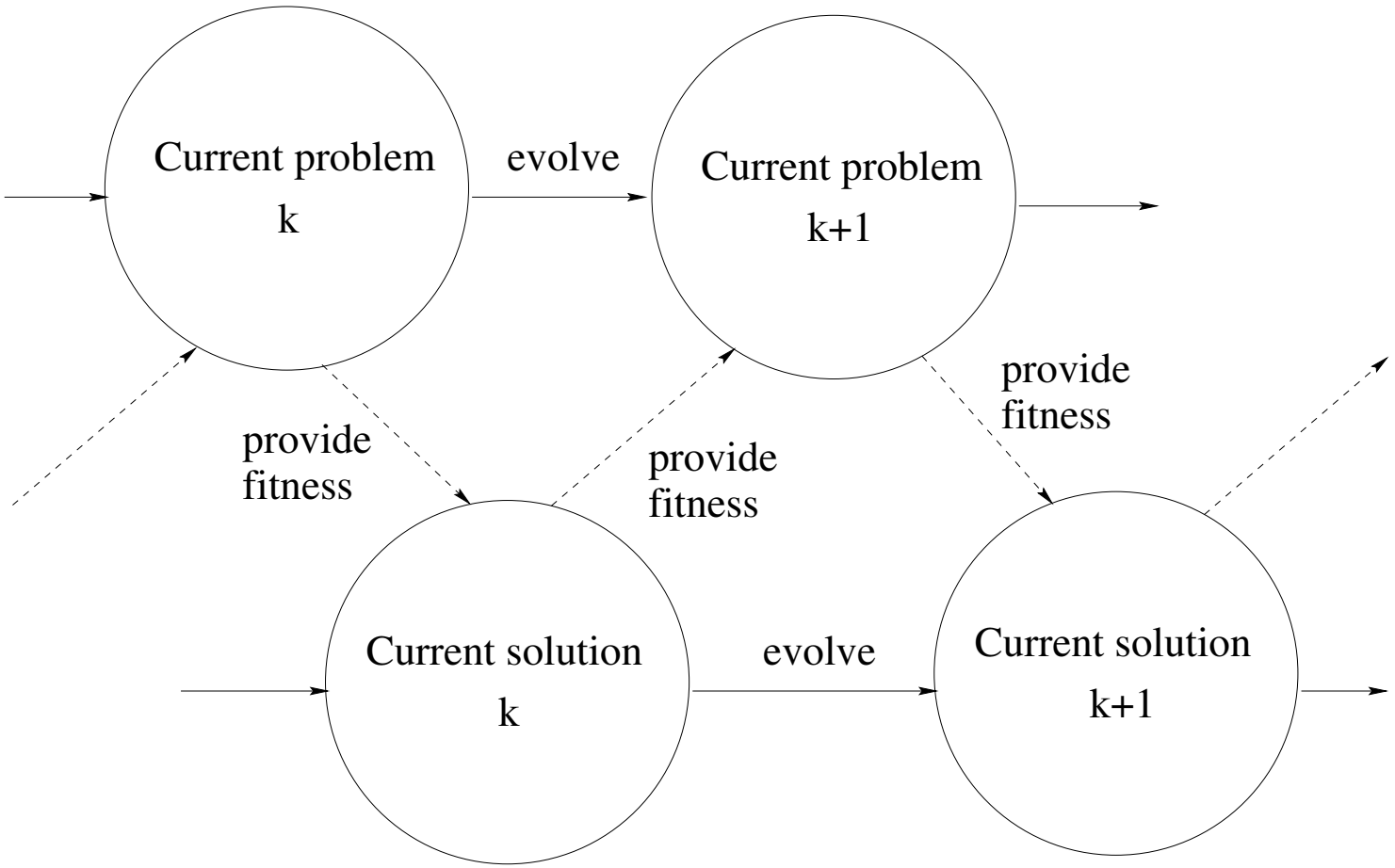
Competitive Co-Evolution: Individuals are competing against each other.

Co-operative Co-Evolution: Individuals co-operate with each other in performing a task.

# Co-Evolution in Design

Inter-population Co-Evolution

Many design problems do not have a fixed specification because people change minds and design environments change.



# Co-evolving Programs and Test Cases

1. Given a software specification, one can co-evolve programs along with test cases/data.
2. Furthermore, one can use a similar idea to fix software bugs automatically using Co-Evolution.
3. The key issue is fitness evaluation.

# Intra-population Co-Evolution

- There is only one population involved.
- A famous example: TD-Gammon.
  - A grand master level player.
  - Learns by self-playing (i.e., Co-Evolution)

What make the player so strong: good machine learning algorithm or self-playing?

# A Simple Experiment

Evolve a neural network that plays backgammon.

1. Let the initial NN be $NN_0$, $k = 0$;
2. Generate a mutant challenger of $NN_k$:

$$w'_{ij} = w_{ij} + Gaussian(0, \sigma).$$

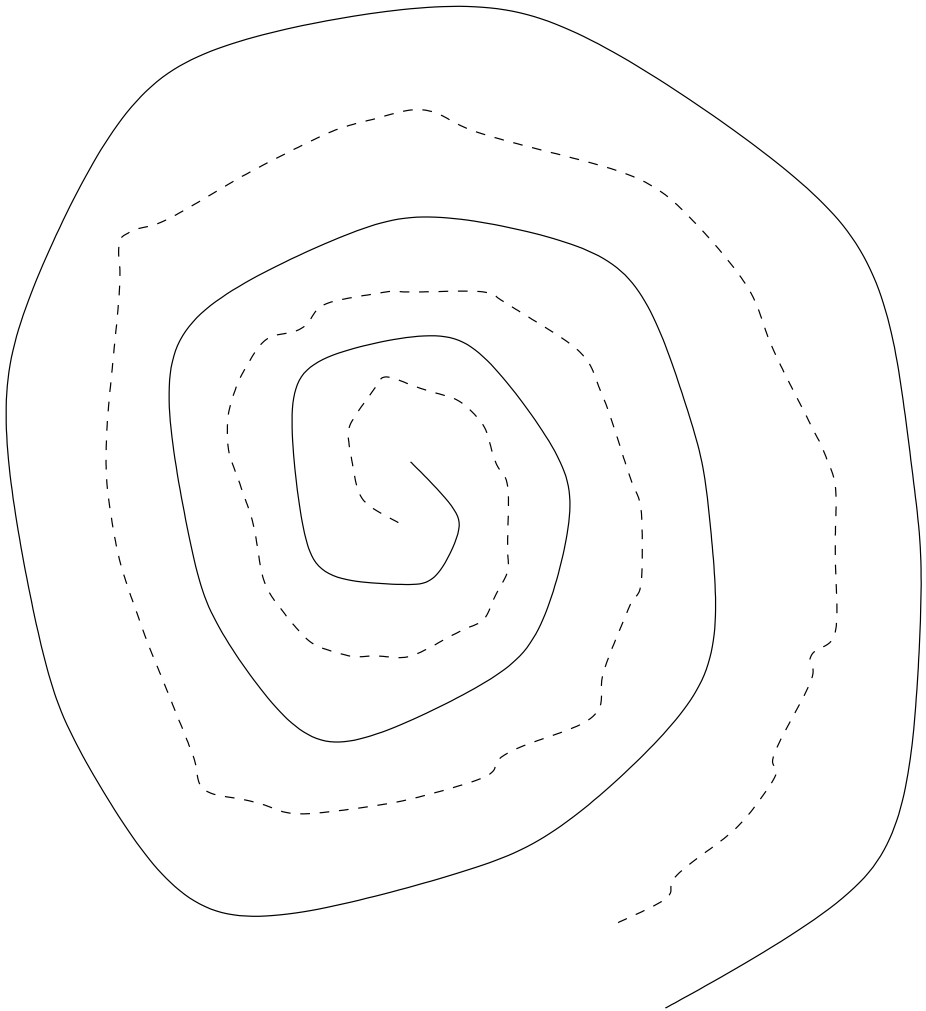
3. If $NN'_k$ is beaten by $NN_k$,
   - then $NN_{k+1} = NN_k$,
   - else $NN_{k+1} = 0.95 \times NN_k + 0.05 \times NN'_k$.
4. $k = k + 1$, go to Step 2 unless the halting criteria are satisfied.

# Comments

- Use 197-20-1 fully-connected feed-forward NN. Initial weights were 0's.
- No training for NNs!
- The EA had population size 1!
- There was only one simple mutation — adding Gaussian noise to weights.
- No recombination at all.

**Performance:** $40\%$ winning against PUBEVAL after 100,000 generations. PUBEVAL is a strong programme trained by experts.

# Separating Interwined Two Spirals



Task: Given 194 training points, learn to separate two spirals.

# The Two Spirals Problem

1. Very tough problem because of its highly nonlinear decision boundaries, especially for MLPs and decision trees.
2. Competitive Co-Evolution based on covering really helps in this case.
   - If no Co-Evolution is used, fitness of an NN is usually determined by the classification accuracy.
   - In Co-Evolution, fitness is evaluated based on pair-wise competition. It depends only on the number of training cases correctly classified by this individual but not by its opponent.

# Iterated Prisoner's Dilemma Games

Non-zero sum, non-cooperative games.

| | | Player B: C | Player B: D |
|---|---|---|---|
| Player A | C | 3 / 3 | 5 / 0 |
| | D | 0 / 5 | 1 / 1 |

How can an EA learn to play the game optimally starting from a population of random strategies?

# Evolutionary Approach to IPD Games

**Use Co-Evolution:** The fitness of an individual is determined by the payoff it obtains by playing against all other individuals in the population.

**Representation of strategies:** Each strategy can be encoded as a binary string easily. One could also use NNs or payoff matrices.

**Evolutionary Algorithm:** Initialised by random strategies.

**Results:** Cooperation (the optimal strategy in this case) can be evolved easily.

# When and Why Co-Evolution

1. We don't know the fitness function.
2. There are too many cases to test in order to obtain a fitness value. Co-Evolution can be used to focus search on the difficult part.
3. The fitness is inherently changing in time.
4. Increase and maintain diversity.
5. Improved robustness and fault-tolerance.

# More Examples of Co-Evolution

- Computer-aided learning
- Robot morphology and controller
- Character recognition
- Checker, chess, etc.
- many others.

# Summary of Co-Evolution

1. Co-Evolution is everywhere. It can occur in a single or multiple populations. It can be used in many areas.
2. An individual's fitness is not fixed in Co-Evolution.
3. Co-Evolution is not well studied yet in evolutionary computation.
4. Artificial Co-Evolution and biological evolution.

# Reading List for This Lecture


1. T. Bäck, D. B. Fogel, and Z. Michalewicz (eds.), Handbook of Evolutionary Computation, IOP Publ. Co. & Oxford University Press, 1997. Section C6.1 and Section C6.2.
2. P. Darwen and X. Yao, "A dilemma for fitness sharing with a scaling function," *Proc. of 1995 IEEE Conference on Evolutionary Computation (ICEC'95)*, Dec. 1995, Perth, Australia, IEEE Press, pp.166–171.
3. P. Darwen and X. Yao, "Every niching method has its niche: fitness sharing and implicit sharing compared," *Lecture Notes in Computer Science, Vol.1141, Proc. of Parallel Problem Solving from Nature (PPSN) IV*, Springer-Verlag, Berlin, pp.398-407, 1996.
4. Pollack, J, Blair, A. and Land, M. (1996). "Coevolution of A Backgammon Player." *Proceedings Artificial Life V*, C. Langton, (Ed), MIT Press.
5. Juillé H. and Pollack, J. B. (1996). "Co-evolving Intertwined Spirals." *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, San Diego, CA, February 29 - March 2, 1996, MIT Press, pp. 461-468.
6. A. Arcuri and X. Yao, "Coevolving programs and unit tests from their specification," *Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'2007)*, Atlanta, Georgia, USA, pp.397-400, ACM Press, New York, NY, USA, November 2007.

# Reading List for Next Lecture


1. Deb, K., Pratap. A, Agarwal, S., and Meyarivan, T. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transaction on Evolutionary Computation*, 6(2), 181-197.
2. H. Wang and X. Yao, "Objective Reduction Based on Nonlinear Correlation Information Entropy", *Soft Computing, June 2016*, Volume 20, Issue 6, pp 2393–2407.
3. Y. Yuan, H. Xu, B. Wang and X. Yao, "A New Dominance Relation Based Evolutionary Algorithm for Many-Objective Optimization", *IEEE Transactions on Evolutionary Computation*, 20(1):16-37, February 2016.
4. K. Praditwong and X. Yao, "A New Multi-objective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm", *Proc. of the 2006 International Conference on Computational Intelligence and Security (CIS'2006)*. IEEE Press, Volume 1, pp.286-291.