Lecture 12: Theoretical Analysis of Evolutionary Algorithms

CSE5012: Evolutionary Computation and Its Applications

Xin Yao

CSE, SUSTech

22 May 2023

Review of the Last Lecture

- **Evolutionary Learning**
  - **Michigan Approach and Pitt Approach**
  - **Evolving Rule-based Systems**
  - **Evolving Neural Networks**

Outline of This Lecture

Some Fundamental Issues in Analysing an EA as an Optimisation Algorithm

1. **Does it converge to the global optimum?**
2. **How long does it take to converge?**
3. **How does the computation time scale with the problem size (i.e., computational time complexity)?**

Convergence and Convergence Rate of EAs

1. **Evolutionary algorithms have often been used in global optimisation.**
   - **Can they find the global optimum as time $\longrightarrow \infty$?**
   - **Under what conditions?**
2. **The canonical genetic algorithm (i.e., the simple GA) does not converge to the global optimum.**
3. **Most EAs with elitism do converge.**
4. **Some EAs without elitism also converge.**
5. **How fast does it converge?**
   - **Only local convergence rates have been calculated for certainly simple functions.**

Ideas Behind Proving Convergence

1. **Model an evolutionary algorithm as a stochastic process, e.g., a Markov chain, where a population at generation t would be modelled as state $\xi(t)$. We want to prove that when $t \longrightarrow \infty$, $\xi(t)$ contains at least one global optimum.**
2. **Establish weak ergodicity of the Markov chain, i.e., showing that *there is a positive probability of visiting any state in the entire search space*.**
3. **Establish strong ergodicity, i.e., *the Markov chain converges to a stable distribution*.**
4. **Verify that such a stable distribution is the optimal distribution, i.e., *the probability of converging to non-optimal states is zero*.**

Remark: There are of course other methods that one can use to prove convergence.

Outline of This Lecture

How to Define Computation Time

- **The computation time of an EA is a stochastic variable.**
- **We consider the first hitting time (FHT), i.e., the first time that the EA has found the global optimum:**

$$\tau = \min\{t; d(\xi_t) = 0\},$$

where

- $\xi_0 = (\mathbf{x}_1, \cdots, \mathbf{x}_\lambda)$ **indicates the initial population of** λ **individuals,**
- $\mathbf{x}^*$ **denotes the optimal solution,**
- t **is the generation number,**
- $d(\xi_t)$ **is the distance between the current population** ξ_t **and the optimal solution, which is**

$$d(\xi_t) = \min\{d(\mathbf{x_i}) : \mathbf{x_i} \in \xi_t\}.$$

Computational Time Complexity

- **What is it? Why is it important?**
- **Results on EA's computational complexity are not many.**
- **Complexity results are usually for a particular EA on a family of problem instances.**
- **Complexity results on EAs are average case results.**
- **Complexity analysis of EAs is very different from classical complexity analysis of combinatorial optimisation problems.**
  1. **Complexity analysis of EAs emphasises the relationship between the EA and the problem instances, which classical analysis is on the problem only. That's why we often use the term of EA-hardness, instead of just hardness.**
  2. **Complexity results on EAs are average case results, which is different from the worst case and the average case analyses in the classical sense.**

Some Research Questions

- **Given a problem instance and an EA, how long does it take for the EA to find the global optimal solution? Specifically,**
  1. **Under what conditions the EA is guaranteed to find the optimal solution in no more than polynomial average time? (An "efficient" EA)**
  2. **Under what conditions the EA is guaranteed to find the optimal solution in at least exponential average time? (An "inefficient" EA)**
- **Key point: The analysis here emphasises the relationship between problems and EAs, so that we can understand what problems are easy/hard for what EAs.**
- **Ultimate Goal: When to use which EA for a given a problem.**

EAs Under Consideration

1. **Initialisation: generate, either randomly or heuristically, an initial population of λ individuals, $\xi_0 = (\mathbf{x}_1, \cdots, \mathbf{x}_\lambda)$, and let $t \leftarrow 0$, where $\lambda > 0$ is an integer. For any population ξ_t, define $f(\xi_t) = \max\{f(\mathbf{x}_i); \mathbf{x}_i \in \xi_t\}$.**

2. **Generation: generate a new intermediate population by crossover and mutation (or any other operators for generating offspring), and denote it as $\xi_{t+1/2}$.**

3. **Selection: select and reproduce λ individuals from the combined population of $\xi_{t+1/2}$ and ξ_t, and obtain another new intermediate population ξ_{t+s}.**

4. **If $f(\xi_{t+s}) = f_{\max}$, then terminate the algorithm; otherwise let $\xi_{t+1} = \xi_{t+s}$ and $t \leftarrow t + 1$, and go to step 2.**

Theorem: Polynomial Average Computation Times I

- **If $\{d(\xi_t); t = 0, 1, 2, \cdots\}$ satisfies the following two conditions,**
  1. **there exists a polynomial of problem size n, $h_0(n) > 0$, such that $d(\xi_t) \leq h_0(n)$ for any population ξ_t, and**
  2. **for any $t \geq 0$, if $d(\xi_t) > 0$, then there exists a polynomial of problem size n, $h_1(n) > 0$, such that**

$$E[d(\xi_t) - d(\xi_{t+1}) \mid d(\xi_t) > 0] \geq \frac{1}{h_1(n)},$$

then starting from any initial population ξ_0 with $d(\xi_0) > 0$,

$$E[\tau \mid d(\xi_0) > 0] \leq h(n),$$

where $h(n)$ is a polynomial of problem size n.

Theorem: Polynomial Average Computation Times II

- **What do the two conditions tell us?**
  1. **The first condition implies that all populations occurred during the evolutionary search process are reasonably close to the optimum, i.e., their distances to the optimum is upper bounded by a polynomial in problem size.**
  2. **The second condition implies that, on average, the EA always drifts towards the optimum with at least some reasonable distance, i.e., the drifts are lower bounded by $\frac{1}{h_1(n)}$, where $h_1(n) > 0$ is a polynomial.**
- **In one sentence:**

  The stochastic process defined by the EA can reach the optimum efficiently (in polynomial time) if populations generated during the search are never too far away from the optimum and the drift towards the optimum is not too small.

Theorem: Exponential Average Computation Time I

1. **For any population ξ_t with $d_b < d(\xi_t) < d_a$, where $d_b \geq 0$ and $d_a > 0$,**

$$E[e^{-(d(\xi_{t+1})-d(\xi_t))} \mid d_b < d(\xi_t) < d_a] \leq \rho < 1,$$

   where $\rho > 0$ is a constant.

2. **For any population ξ_t with $d(\xi_t) \geq d_a$, $d_a > 0$,**

$$E[e^{-(d(\xi_{t+1})-d_a)} \mid d(\xi_t) \geq d_a] \leq D,$$

   where $D \geq 1$ is a constant.

If $d(\xi_0) \geq d_a$, $D \geq 1$ and $\rho < 1$, then there exist some $\delta_1 > 0$ and $\delta_2 > 0$ such that

$$E[\tau \mid d(\xi_0) \geq d_a] \geq \delta_1 e^{\delta_2(d_a - d_b)}.$$

Theorem: Exponential Average Computation Time II

- **What do the two conditions tell us?**
  - **The first condition indicates that (d_b, d_a) is a very difficult interval to search. When this condition is satisfied, $d(\xi_{t+1}) > d(\xi_t)$. In other words, the offspring population is on average drifting away from the optimum, rather than getting closer to it.**
  - **The second condition indicates that a population in the interval $[d_a, +\infty)$ will not, on average, drift towards the optimum too much because it is always quite far away from the optimum, i.e., $d(\xi_{t+1}) \geq d_a - \ln D$.**
- **The theorem says we should not use such an EA for the problem because it's very inefficient.**

Some Remarks

1. **It has been shown that an EA will take at least an exponential amount of time (in the problem size) to find the optimal solution to an NP-hard problem.**
2. **It can also be shown that an EA may take at least an exponential amount of time (in the problem size) to find the optimal solution to a polynomial time problem.**
3. **However, certain EA takes no more than a polynomial time (in the problem size) to find the optimal solution to a polynomial time problem.**
4. **An EA may take substantial less time (from exponential to polynomial) in finding an approximate solution to a problem instance class.**
5. **Population can make a substantial difference to time complexity, i.e., from population size 1 to a size that is greater than 1.**

Problem Classification: Why Bother?

- **When we analyse an algorithm, we want to know which problem instance classes are more amenable to this algorithm and which are not. Different instance classes of a problem pose different challenges to different algorithms.**
- **In evolutionary computation, we are particularly interested in problem characteristics that make the problem hard or easy for a given algorithm. A problem instance class may be very hard for one algorithm, but easy for another.**
- **Analysing the relationship between problem characteristics and algorithmic features will shed light on the essential question of when to use which algorithm in solving a difficult problem instance class.**

Problem Classification: EA-hard vs EA-easy

- **Given an EA, we can divide all optimisation problems into two classes based on the mean number of generations (i.e., the mean first hitting time) needed to solve the problems.**[1]

Easy Class **Starting from any initial population, the mean number of generations needed by the EA to solve the problem, i.e., $E[\tau|\xi_0]$, is at most polynomial in the problem size.**

Hard Class **Starting from some initial population, the mean number of generations needed by the EA to solve the problem, i.e., $E[\tau|\xi_0]$, is at least exponential in the problem size.**

[1] Jun He and Xin Yao. "A study of drift analysis for estimating computation time of evolutionary algorithms". In: *Natural Computing* 3.1 (2004), pp. 21–35

So, What Are Easy/Hard Problems?

- **Given an EA, a problem belongs to the EA-easy Class *if and only if* there exists a distance function $d(\xi_t)$, such that for any population ξ_t with $d(\xi_t) > 0$,**
  1. **$d(\xi_t) \leq g_1(n)$, where $g_1(n)$ is polynomial in the problem size n, and**
  2. **$E[d(\xi_t) - d(\xi_{t+1})|\xi_t] \geq c_{low}$, where $c_{low} > 0$ is a constant.**

- **Given an EA, a problem belongs to the EA-hard Class *if and only if* there exists a distance function $d(\xi_t)$, such that**
  1. **for some population ξ_{t_1}, $d(\xi_{t_1}) \geq g_2(n)$, where $g_2(n)$ is exponential in the problem size n, and**
  2. **for any population ξ_t with $d(\xi_t) > 0$, $E[d(\xi_t) - d(\xi_{t+1})|\xi_t] \leq c_{up}$, where $c_{up} > 0$ is a constant.**

How to Analyse Time Complexity of EAs I

- **Consider the population sequence $\{\xi_t; t = 0, 1, \cdots, n\}$ on a probability space, the general approach to analysing EAs using drift analysis can be summarised as follows:**
  1. **Define a distance function $d(\xi)$ to measure the distance between a population ξ and the optimal solution.**
  2. **Estimate the one-step mean drift of ξ_t towards the optimal solution:**

$$E[d(\xi_t) - d(\xi_{t+1})].$$

  3. **Derive the computation time using the concept of**

$$Time = \frac{InitialDistance}{OneStepDrift}.$$

Jun He and Xin Yao. "Drift analysis and average time complexity of evolutionary algorithms". In: *Artificial intelligence* 127.1 (2001), pp. 57–85

An Example of Drift Analysis

After Friday night dinner party, ...

- **Xin Yao walks from the restaurant to his house.**
  - **The restaurant is n meters away from his house.**
  - **He always moves forward 1 meter in each step.**
- **Question: how many steps does he need to reach his house?**

Simple and Correct Answer

n **steps!**

Making Simple Things Complicated

- **Define a distance function to measure the distance between a point x and the house,**

$$d(x) = x, \quad x \in \{0, \cdots, n\}.$$

- **Estimate the "speed" (drift) (ξ_t is his position at t-step):**

$$d(\xi_t) - d(\xi_{t+1}) = \begin{cases} 0, & \xi_t = 0, \\ 1, & \xi_t \in \{1 \cdots, n\}. \end{cases}$$

- **Then the first hitting (i.e., computation) time τ to the house:**

$$\tau = \frac{distance}{drift} = n.$$

After a Few Drinks

- **The impact of the few drinks:**
  - **From the restaurant, he always leaves for the house;**
  - **In between the restaurant and house,**
    - **he moves 1 meter forward in each step with probability 0.6,**
    - **he moves 1 meter backward in each step with probability 0.4.**
- **Question: how many steps does he need to go to his house?**

Drift Analysis: Getting Back After a Few Drinks

- **Define the same distance function** $d(x) = x,\ x \in \{0, \cdots, n\}$**.**
- **Estimate his speed (drift) (**ξ_t **is his position at** t**-step):**

$$d(\xi_t) - d(\xi_{t+1}) = \begin{cases} 1, & \textbf{if } \xi_t = n; \\ 1, & \textbf{if } 0 < \xi_t < n, \quad \textbf{with probability } 0.6, \\ -1, & \textbf{if } 0 < \xi_t < n, \quad \textbf{with probability } 0.4. \end{cases}$$

- **Compute the mean speed/drift before arriving at the house:**

$$1 \geq E[d(\xi_t) - d(\xi_{t+1})] \geq 0.6 \times 1 - 0.4 \times 1 = 0.2.$$

- **The mean first hitting time to the house is:**

$$n \leq E[\tau] \leq \frac{n}{0.2} = 5n.$$

Too Many Drinks

- **After drinking too much:**
  - **He always heads towards his house.**
  - **In between the restaurant and house,**
    - **he moves 1 meter forward in each step with probability 0.4,**
    - **he moves 1 meter backward in each step with probability 0.6.**
- **Question: how many steps does he need to arrive at his house?**

Drift Analysis: Journey Towards the House

- **Define a new distance function:**

$$d(x) = \begin{cases} 0, & x = 0, \\ \sum_{i=0}^{x} 1.5^{n-i}, & x \in \{1, \cdots, n\}. \end{cases}$$

- **Speed/drift in between the restaurant and house ($\xi_t = x$ is his position at t-step),**

$$d(\xi_t) - d(\xi_{t+1}) = \begin{cases} 1.5^{n-x}, & \textbf{with probability } 0.4, \\ -1.5^{n-(x+1)}, & \textbf{with probability } 0.6. \end{cases}$$

- **Mean speed/drift before arriving at his house:**

$$1 \geq E[d(\xi_t) - d(\xi_{t+1})] \geq 0.4 \times 1.5^{n-x} - 0.6 \times 1.5^{n-(x+1)} = 0.$$

- **The mean first hitting time τ to his house:** $E[\tau] \geq \sum_{i=0}^{n} 1.5^{n-i}$.

Some Runtime Complexity Results

OneMax	(1+1) EA	$O(n \log n)$
	(1+λ) EA	$O(\lambda n + n \log n)$
	(μ+1) EA	$O(\mu n + n \log n)$
	1-ANT	$O(n^2)$ w.h.p.
	(μ+1) IA	$O(\mu n + n \log n)$
Linear Functions	(1+1) EA	$\Theta(n \log n)$
	cGA	$\Theta(n^{2+\varepsilon})$, $\varepsilon > 0$ const.
Max. Matching	(1+1) EA	$e^{\Omega(n)}$, PRAS
Sorting	(1+1) EA	$\Theta(n^2 \log n)$
SS Shortest Path	(1+1) EA	$O(n^3 \log(nw_{max}))$
	MO (1+1) EA	$O(n^3)$
MST	(1+1) EA	$\Theta(m^2 \log(nw_{max}))$
	(1+λ) EA	$O(n \log(nw_{max}))$
	1-ANT	$O(mn \log(nw_{max}))$
Max. Clique	(1+1) EA	$\Theta(n^5)$
(rand. planar)	(16n+1) RLS	$\Theta(n^{5/3})$
Eulerian Cycle	(1+1) EA	$\Theta(m^2 \log m)$
Partition	(1+1) EA	4/3 approx., competitive avg.
Vertex Cover	(1+1) EA	$e^{\Omega(n)}$, arb. bad approx.
Set Cover	(1+1) EA	$e^{\Omega(n)}$, arb. bad approx.
	SEMO	Pol. $O(\log n)$-approx.
Intersection of	(1+1) EA	$1/p$-approximation in
$p \geq 3$ matroids		$O(\lvert E\rvert^{p+2} \log(\lvert E\rvert w_{\max}))$
UIO/FSM conf.	(1+1) EA	$e^{\Omega(n)}$

Figure 1: From "A Gentle Introduction to the Time Complexity Analysis of Evolutionary Algorithms" by P. S. Oliveto and X. Yao (2012).

Outline of This Lecture

No Free Lunch Theorems[2]

Theorem

For any pair of algorithms a_1 *and* a_2

$$\sum_f P(d_m^y | f, m, a_1) = \sum_f P(d_m^y | f, m, a_2),$$

where d_m^y *is used to indicate the ordered set of size* m *of the cost values* $y \in Y$ *associated to input values* $x \in X$*,* $f : X \to Y$ *is the function being optimised and* $P(d_m^y | f, m, a)$ *is the conditional probability of obtaining a given sequence of cost values from algorithm* a *run* m *times on function* f*.*

[2] David H Wolpert, William G Macready, et al. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82

No Free Lunch Theorems

Discussions

1. **For any algorithm, its performance gain over a class of problems is offset by its performance loss over a different class of problems.**
   $\rightarrow$ No algorithm is best for every problem.

2. **What are implications?**
   - **"If an algorithm does particularly well on average for one class of problems then it must do worse on average over the remaining problems."**[3]
   - **Generalisation vs. specialisation.**

[3] David H Wolpert, William G Macready, et al. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82

Outline of This Lecture

Motivation

Remark

In a previous lab, you have compared different variation operators and observed that different variation operators led to diverse performance of the designed algorithm. To better understand their impact and to design more effective GA, we want to understand how evolution works from generation to generation.

Schema (plural "Schemata")

- **A hyperplane in the (genome) search space.**
  $\rightarrow$ Partitions of search space.
- **Representation using binary alphabet: 1, 0, $*$ (don't care).**
- **Specifying sets of "similar" chromosomes.**
- **Sometimes called similarity template or pattern.**

Schema	Examples of Schema
*11**110	01100110
	01101110
	...
	11110110
	11111110

Table 1: Example of a schema and possible examples. Not all the possible examples are necessarily shown up in a population.

Two Important Features of Schema

1. **Schema *order*:**
   $o(H) =$ **the number of fixed ($0/1$) positions in H.**
   - **Motivation: Some schemata are more specific than others.**
   - **Example: $*11**110$ is more specific than $*11*****$.**

2. **Schema *defining distance*:**
   $d(H) =$ **the distance between the first and last fixed position.**
   - **Motivation: Some schemata span more of the total length than others.**
   - **Example: $*1*****0$ spans a larger portion of chromosome than $**1**1**$.**

Example

Consider the the schema $H = *11**110$,

1. its order is $o(H) = 5$ as there are five 0 and 1, and
2. its defining distance is $d(H) = 6$, calculated as $length(11**110) - 1$.

Counting the Schemata

- **Consider a chromosome of length l, there are 3^l schemata.**
- **In general, a chromosome with only 0/1 contains 2^l schemata.**

  $\Longrightarrow$ A population of size λ contains between 2^l and $\lambda \cdot 2^l$ schemata, depending on the diversity in the population.

Questions

1. Which variation operations will affect the #schemata in a population?
2. In which way do the variation operations will affect the #schemata?
3. How will the #schemata change after the variation operations?

We will answer the questions while introducing the Schema Theorems in the following slides.

Notations

- **Schema (Schemata) H: binary strings consisting of 1, 0, $*$.**
- **Schema order $o(H)$: the number of fixed (0/1) positions in H.**
- **Schema defining length $d(H)$: the distance between the first and last fixed position.**
- **$m(H,t)$: number of examples in the population at time t which contains schema H.**
- **$f(H)$: average fitness of the individuals containing H at time t.**
- **$\bar{f}$: average fitness of the population at time t.**
- **l: length of an individual string.**
- **p_c: crossover rate.**
- **p_m: mutation rate.**

Variation Operators Affect Schemata I

- **The number of examples of a schema in a population Pop depends on the effects of variation operators.**
- **Crossover and mutation disrupt the schema if they result in a change to one of the fixed positions.**

1. **Selection operators only changes the relative frequency of *pre-existing* examples.**
   - **Reproduction alone does not explore new regions of the search space.**
   - **Fitness proportionate selection: The expected number of examples of H in the Pop after selection is**

$$m'(H,t) = \lambda \cdot p_s(H,t) = \lambda \cdot \left(m(H,t) \cdot \frac{f(H)}{\lambda \cdot \bar{f}} \right) = m(H,t) \cdot \frac{f(H)}{\bar{f}}.$$

Variation Operators Affect Schemata II

2. **Recombination operators can create new examples and destroy current ones.**
   - **Increase/Decrease #examples of a schema.**
   - **One-point crossover (1X): Considering a genotype of length l that contains an example of a schema H, the schema may be disrupted with probability $p_d(H, 1X) = \frac{d(H)}{l-1}$. The probability of surviving crossover is**

$$1 - p_c \frac{d(H)}{l-1}.$$

3. **Mutation operators can create new examples and disrupt current ones.**
   - **Increase/Decrease #examples of a schema.**
   - **Bitwise mutation (BM) with p_m: the probability of surviving mutation is**

$$(1 - p_m)^{o(H)} \approx 1 - o(H) \cdot p_m$$

for small values of p_m.

Schema Theorem

- **The expected number of copies of schema H in the next generation is**

$$m(H, t+1) \quad \geq \quad m(H,t)\frac{f(H)}{\bar{f}}\left(1 - p_c\frac{d(H)}{l-1}\right)(1-p_m)^{o(H)} \tag{1}$$

- **Core message from Equation (1):**

  Short, low-order, above average schemata receive exponentially increasing copies of individuals in subsequent generations.

- **The Schema Theorem is also called the "Fundamental Theorem of Genetic Algorithms".**

Implications of Schema Theorem

- **Short, low-order schema is more likely to be transmitted to the next generation than longer, or higher-order schema of the same mean fitness.**
- **Implicit parallelism: For a GA of population size λ, the number of schemata is proportional to the cube of the population size: $O(\lambda^3)$ estimate.**[4][5]

[4] David E Goldberg. "Optimal initial population size for binary-coded genetic algorithms". In: *Proceeding of the International Joint Conference on Artificial Intelligence*. Vol. 9. 1985, pp. 588–592

[5] David E Goldberg. "Genetic algorithms in search". In: *Optimization, and Machine Learning* (1989) (Pages 40-44)

Building Block Hypothesis

- **Building blocks: Short, low-order schemata of above-average fitness values (high performance).**
- **Building Block Hypothesis: *"A GA seeks near optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the building blocks".***[6]

Question

- What happens if the global optimum is not an example of the low-order schemata that have the highest mean fitness?
  $\rightarrow$ Will answer when introducing "deception".

[6] David E Goldberg. "Genetic algorithms in search". In: *Optimization, and Machine Learning* (1989)

Outline of This Lecture

Fitness Landscapes

- **P. F. Stadler provided a general view of (discrete or continuous) landscapes, consisting of 3 elements:**
  1. **a set X of configurations (solutions to the problem),**
  2. **a neighbourhood, nearness, distance, or accessibility on X, and**
  3. **a fitness function $f : X \mapsto R$**
- **Fitness landscape analysis helps understanding problem structures and designing appropriate algorithms.**

Katherine M Malan and Andries P Engelbrecht. "A survey of techniques for characterising fitness landscapes and some possible ways forward". In: *Information Sciences* 241 (2013), pp. 148–163

Fitness Landscape Characterisation

- **What characteristics of a fitness landscape make the problem hard or easy for an EA?**
- **The analysis of EA-hardness mentioned before addresses this key issue.**
- **Many other methods have also been used to analyse and understand fitness landscapes.**
  1. **Deceptive landscapes, trap functions, ...**
  2. **Number of local optima.**
  3. NK **landscapes.**
  4. **Correlation analysis of fitness landscapes.**

Deceptiveness & Epistasis

- **GAs are powerful for dealing with NP-Complete problems, however, they still have difficulty with some problems (GA-hard problems).**
- **Two features representing difficulty:**
  1. **Deceptiveness.**
  2. **Epistasis indicates the degree of interdependency and nonlinearity between genes in a chromosome.**

  $\rightarrow$ **The principle of causality: Small changes of the genotype can lead to large changes in fitness.**
- **Why important: Understand the weaknesses of GAs and in which cases GAs might not be a good choice.**

GA-deception

- **GA-deceptive problems: The necessary building blocks for successful optimisation are not present. It usually happens to problems with isolated optima.**
- **Deceptive attractors: The local optima that lead the GAs away from the global optimum.**

Epistatic Problems

- **Epistasis indicates the degree of interdependency and nonlinearity between genes in a chromosome.**
- **Deceptive problems cannot contain low epistasis.**
- **Problems with high epistasis are not always deceptive.**

Question

Well, you have used "low" and "high" to describe the level of epistasis. How to measure the level?

NK Landscape

- **Introduced by Stuart Kauffman in 1987.**
- **NK landscapes: stochastically generated fitness landscapes on bit strings with N genes and K epistatic interactions between genes: $f : \{0,1\}^N \mapsto \mathrm{R}$, $\mathbf{x} \in \{0,1\}^N$.**
- **The NK fitness of a chromosome x is defined as**

$$fitness_{NK}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_i(x_i; x_1^i, \ldots, x_K^i),$$

where $f_i(x_i; x_1^i, \ldots, x_K^i)$ is a partial fitness and refer to the fitness contribution from each gene i. It depends on its own allele x_i and the alleles of K neighbour genes $x_1^i, \ldots, x_K^i$.

Discussion on NK Landscapes

- **K epistatic interactions: the contribution of other genes to a specific gene in terms of the overall fitness.**
- **K can be set to determine the degree of "ruggedness".**
  $\rightarrow$ "tunably rugged" fitness landscape.

Question

What will happen if increasing or decreasing the value of K?

Relevant Analysis Results

- **Simplest case:** $K = 0$
  - **Each gene is independent of all the others.**
  - **Except for rare "ties", there is only a single global optimum.**
  - **The expected number of steps for a local hill climbing to hit the optimum is $N/2$.**
- **Fully connected NK model:** $K = N - 1$
  - **Completely random fitness landscape.**
  - **Such random landscapes have, on average, $2N/(N + 1)$ local optima.**
  - **Complexity catastrophe: The average fitness value of local optima falls to almost the mean fitness of the whole landscape.**

Fitness Correlation Analysis

1. **Fitness correlation coefficient: measuring the correlation between parent and offspring fitness where the offspring is generated by an operator.**

2. **Fitness distance correlation: measuring the correlation between the fitness of search points and the distances to the global optimum.**

Tools and Techniques Used for EC Theoretical Studies

1. **Stochastic processes, especially Markov chains.**
2. **Statistical analysis, e.g., drift analysis techniques.**
3. **Counting, algebra, ...**
4. **Dynamical systems.**
5. **Orthogonal functions analysis, like Fourier, Walsh, Haar, ...**
6. **Statistical physics.**

Outline of This Lecture

Summary

1. **Theory answers fundamental questions.**
2. **Theory supports the further development of a field.**
3. **Theoretical ideas could be understood without mathematical notations, but we do sacrifice the rigour of results.**
4. **We only touched upon a very small part of EC theory. More comprehensive coverage can be found from:**

   Frank Neumann, Carsten Witt (2010):
   Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity. *Natural Computing Series*, Springer, ISBN 978-3-642-16543-6.

References for This Lecture

[1] David E Goldberg. "Genetic algorithms in search". In: *Optimization, and Machine Learning* (1989).

[2] David E Goldberg. "Optimal initial population size for binary-coded genetic algorithms". In: *Proceeding of the International Joint Conference on Artificial Intelligence*. Vol. 9. 1985, pp. 588–592.

[3] Jun He and Xin Yao. "A study of drift analysis for estimating computation time of evolutionary algorithms". In: *Natural Computing* 3.1 (2004), pp. 21–35.

[4] Jun He and Xin Yao. "Drift analysis and average time complexity of evolutionary algorithms". In: *Artificial intelligence* 127.1 (2001), pp. 57–85.

[5] Katherine M Malan and Andries P Engelbrecht. "A survey of techniques for characterising fitness landscapes and some possible ways forward". In: *Information Sciences* 241 (2013), pp. 148–163.

[6] David H Wolpert, William G Macready, et al. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

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. (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.**