# Chapter 6
# Evolutionary Algorithms: Concepts, Designs, and Applications in Bioinformatics

**Ka-Chun Wong**
*City University of Hong Kong, Hong Kong SAR*

## ABSTRACT

*Inspired from nature, evolutionary algorithms have been proven effective and unique in different real world applications. Comparing to traditional algorithms, its parallel search capability and stochastic nature enable it to excel in search performance in a unique way. In this chapter, evolutionary algorithms are reviewed and discussed from concepts and designs to applications in bioinformatics. The history of evolutionary algorithms is first discussed at the beginning. An overview on the state-of-the-art evolutionary algorithm concepts is then provided. Following that, the related design and implementation details are discussed on different aspects: representation, parent selection, reproductive operators, survival selection, and fitness function. At the end of this chapter, real world evolutionary algorithm applications in bioinformatics are reviewed and discussed.*

## INTRODUCTION

Since genetic algorithm was proposed by John Holland (Holland, 1975) in the early 1970s, the study of evolutionary algorithm has emerged as a popular research field (Civicioglu & Besdok, 2013). Researchers from various scientific and engineering disciplines have been digging into this field, exploring the unique power of evolutionary algorithms (Hadka & Reed, 2013). Many applications have been successfully proposed in the past twenty years. For example, mechanical design (Lampinen & Zelinka, 1999), electromagnetic optimization (Rahmat-Samii & Michielssen, 1999), environmental protection (Bertini, De, Moretti, & Pizzuti, 2010), finance (Larkin & Ryan, 2010), musical orchestration (Esling, Carpentier, & Agon, 2010), pipe routing (Furuholmen, Glette, Hovin, & Torresen, 2010), and nuclear reactor core design (Sacco, Henderson, Rios-Coelho, Ali, & Pereira, 2009). In particular, its function

DOI: 10.4018/978-1-5225-0788-8.ch006

optimization capability was highlighted (Goldberg & Richardson, 1987) because of its high adaptability to different function landscapes, to which we cannot apply traditional optimization techniques (Wong, Leung, & Wong, 2009).

## BACKGROUND

Evolutionary algorithms draw inspiration from nature. An evolutionary algorithm starts with a randomly initialized population. The population then evolves across several generations. In each generation, fit individuals are selected to become parent individuals. They cross-over with each other to generate new individuals, which are subsequently called offspring individuals. Randomly selected offspring individuals then undergo certain mutations. After that, the algorithm selects the optimal individuals for survival to the next generation according to the survival selection scheme designed in advance. For instance, if the algorithm is overlapping (De Jong, 2006), then both parent and offspring populations will participate in the survival selection. Otherwise, only the offspring population will participate in the survival selection. The selected individuals then survive to the next generation. Such a procedure is repeated again and again until a certain termination condition is met (Wong, Leung, & Wong, 2010). Figure 1 outlines a typical evolutionary algorithm.

In this book chapter, we follow the unified approach proposed by De Jong (De Jong, 2006). The design of evolutionary algorithm can be divided into several components: representation, parent selection, crossover operators, mutation operators, survival selection, and termination condition. Details can be found in the following sections.

*Figure 1. Major components of a typical evolutionary algorithm*

**Algorithm 1 A Typical Evolutionary Algorithm**

```
Choose suitable representation methods;

P(t): Parent Population at time t
O(t): Offspring Population at time t

t ← 0;
Initialize P(t);
while not termination condition do
    temp = Parent Selection from P(t);
    O(t+1) = Crossover in temp;
    O(t+1) = Mutate O(t+1);
    if overlapping then
        P(t+1) = Survival Selection from O(t+1) ∪ P(t) ;
    else
        P(t+1) = Survival Selection from O(t+1) ;
    end if
    t ← t+1;
end while

Good individuals can then be found in P(t);
```