Chapter 6
Parallel Evolutionary Computation in R

Cedric Gondro
The Centre for Genetic Analysis and Applications, University of New England, Australia

Paul Kwan
University of New England, Australia

ABSTRACT

Evolutionary Computation (EC) is a branch of Artificial Intelligence which encompasses heuristic optimization methods loosely based on biological evolutionary processes. These methods are efficient in finding optimal or near-optimal solutions in large, complex non-linear search spaces. While evolutionary algorithms (EAs) are comparatively slow in comparison to deterministic or sampling approaches, they are also inherently parallelizable. As technology shifts towards multicore and cloud computing, this overhead becomes less relevant, provided a parallel framework is used. In this chapter the authors discuss how to implement and run parallel evolutionary algorithms in the popular statistical programming language R. R has become the de facto language for statistical programming and it is widely used in biostatistics and bioinformatics due to the availability of thousands of packages to manipulate and analyze data. It is also extremely easy to parallelize routines within R, which makes it a perfect environment for evolutionary algorithms. EC is a large field of research, and many different algorithms have been proposed. While there is no single silver bullet that can handle all classes of problems, an algorithm that is extremely simple, efficient, and with good generalization properties is Differential Evolution (DE). Herein the authors discuss step-by-step how to implement DE in R and how to parallelize it. They then illustrate with a toy genome-wide association study (GWAS) how to identify candidate regions associated with a quantitative trait of interest.

INTRODUCTION

In recent years R (R Development Core Team 2011) has become *de facto* statistical programming language of choice for statisticians and it is widely used to teach statistic courses at universities. It is also arguably the most widely used environment for analysis of high throughput genomic data and in particular for microarray analyses. R's main strength lies in the literally thousands of packages freely available from repositories such as CRAN or Bioconductor (Gentleman *et al.* 2004) which build on the core platform. Chances are that there already is an *off the shelf* package available for a

DOI: 10.4018/978-1-4666-3604-0.ch006

particular task. At the end of this chapter we briefly summarize the main Evolutionary Computation packages that are available for R.

Since R is a scripted language it is very easy to essentially *assemble* various packages, add some personalized routines and chain-link it all into a full analysis pipeline all the way from raw data to final report. This of course dramatically reduces development and deployment times for complex analyses. The downside is that the development speed and ease comes along with a certain compromise in computational times because R is a scripted language and not a compiled one. But there are some tricks for writing R code which will improve performance, and we will discuss some of these later on. Alternatively, for time critical routines, R can be dynamically linked to compiled code in C or Fortran (and also other languages to various degrees), this opens the possibility of using prior code or developing code specifically tailored for solving a computationally intensive task and then sending the results back into R for further downstream analyses (Gentleman 2009).

Parallel computation has been a buzz word for a few years now, but programs and programming practices have not quite caught up with the technology and there generally is a reasonable amount of work involved in developing a program that runs in parallel. Of course this will be problem specific, but it is relatively easy to parallelize iterative routines in R; and this is especially true for evolutionary algorithms (EAs) which are inherently parallelizable.

R is also platform independent. Scripts will generally run on any operating system. When all these factors are taken together we have a perfect environment for working with complex problems. Herein we assume that the reader is reasonably familiar with R and its syntax. For those who are unfamiliar with it, two excellent texts more focused on the programming aspects of the language are Chambers (2008) and Jones *et al.* (2009). A very brief *Getting Started with R* is provided in Appendix 1 for the interested readers.

A Quick Tour of Evolutionary Algorithms

Evolution can be seen as a dynamic and opportunistic optimization process. Effectively it is a method to search through a vast solution space and find a solution that allows organisms to survive and reproduce in a certain environment. It is dynamic in the sense that solutions (organisms) can change to adapt to environmental changes and it is opportunistic in the sense that solutions are not necessarily globally optimal but rather tend to move to the next available solution that ensures viability, even if in detriment of a more globally optimal solution. Interestingly enough, the high-level rules that govern evolution and account for the great variability of organisms are quite straightforward. Organisms – which can be seen as candidate solutions – evolve through random variation due to mutation, crossover and manipulations on their genetic material; these candidates are subjected to selective pressures which evaluate their *adaptiveness* and determine their capacity of generating descendants, thus propagating better fit genotypes into the future generations. These characteristics are the inspiration of Evolutionary Computation.

Evolutionary algorithms are primarily computational methods designed for optimization of complex problems with large search spaces. These algorithms try to mimic the mechanisms of biological evolution to evolve a solution (Mitchell and Taylor 1999; Fogel 2000a; Fogel 2000b). Even though specific implementations can vary significantly and algorithms are not constrained to using only biological mechanisms, there are three common features which are shared by the different branches of EC (Bäck 2000): population, selection and search operators.

- **Population:** A number (n) of candidate solutions (representations of the problem) compete against each other to remain in the population and generate offspring. Since