Student: Yifei Pei ID: 1611648 Title: Report 2 Course: Software Architecture

Lecturer: Katrina Falkner Deadline: 16 October 2013

Biologically-inspired software architectures

Introduction

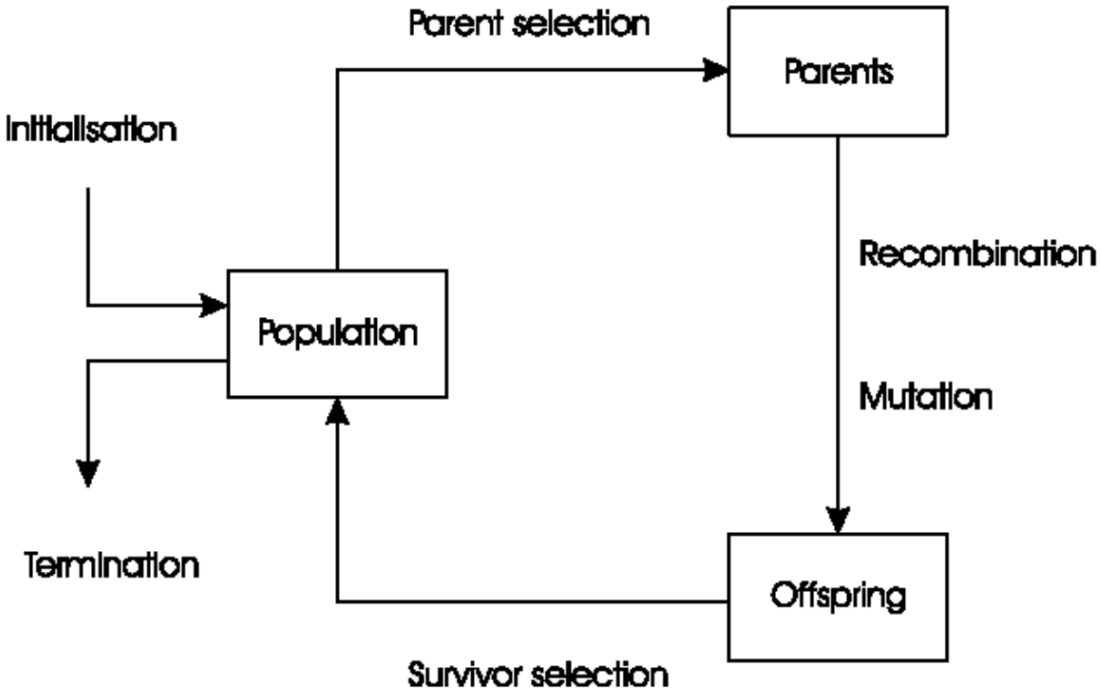
As the name implies, biologically-inspired software architecture refers to the software architecture for Biologically-inspired Computing (Bio-inspired Computing). Academically, Bio-inspired Computing is a broadly defined subject in computer science, which loosely knits together subfields related to the topics of biology. Although this field is tightly connected with findings and theories of biology, its emphasis is actually how the biological concepts can help improve the implementation of computer science.[1] The major domain in computer science that applies the Bio-inspired Computing is Artificial Intelligence, as many of its pursuits can be linked to machine learning. The way to differ Bio-inspired Computing with traditional AI is that it uses evolutionary approaches to improve the algorithm itself rather than just creating an AI algorithm to achieve the goals set by requirements.[2] There are many different subsets of Bio-inspired Computing, including genetic algorithms, biodegradability prediction, cellular automata, emergent systems, neural networks, artificial life, artificial immune systems, rendering (computer graphics), Lindenmayer systems, communication networks and protocols, membrane computers, excitable media, and sensor networks.[3] Each of them can imply certain types of theories and patterns in software architecture. Due to the word limit and the knowledge structure of this report, the main topic that this report will discuss is the very basic level of Bio-inspired Computing - Evolutionary Computation. The purpose of this report is just to give an introduction of the generic software architectures for Evolutionary Computation.

Evolutionary Computation

Evolutionary Computation gets use of the biological theory, Darwinian Evolution, to simulate an environment by using algorithms to help achieve the optimised goals of computer science problems. By doing Evolutionary Computation, it applies a metaphor of three elements in evolutionary theory, where the “environment” in biology is the “problem” in computer science; the “individual” in biology is the “candidate solution” for the “problem”; and the “fitness”, which represents the chance for survival and reproduction, is the “quality”, chance for seeding new solutions. To solve the problem, the algorithm need to generate solutions to simulate “individuals” in nature, and the number of solutions should reach the number of “population”, a generation for example, in nature. After a “generation” of solutions have been produced, the algorithm filter the solutions by some designated conditions based on the problem to simulate the “selection” in nature. After the “selection”, the algorithm will be able to produce next “generation” of solutions and do the “selection” again. After sufficient generations’ selection, the solutions will be optimised to solve the problem, because the “survivals fit the environment best in nature”.[4] Generally, the problems executed by Evolutionary Computation are NP and NP-hard problems which have quite large extent of complexity, so the evolutionary simulation can simplify the problem to “generating the survivals to fit the environment best”. There are two rules of Darwinian Evolution in biology support the application of Evolutionary Computation: 1. Survival of fittest. 2. Diversity drives change.[5] Theoretically, if the algorithm simulate the evolution in nature correctly, optimised solutions will finally be produced after sufficient generations’ selection. The Darwinian Evolution theory and its derivations are the only link to biology in Evolutionary Computation. The implementation of Evolutionary Computation needs the relevant knowledge of computer science to manipulate.

Architectures for Evolutionary Computation

The soul of Evolutionary Computation is the architecture for the algorithm. As the problems, the solving mechanism for problems, and the complexity of problems can vary, the key for Evolutionary Computation is to simulate a nature-like environment to solve the problem and optimise the solution. To ensure this mechanism of the algorithm, it needs to work in a relatively fixed architecture. Therefore, there is a common pattern for all the problems that fit this algorithm.



The figure above[6] describes the scheme for Evolutionary Computation and infer the architecture of Evolutionary Computation. First of all, the algorithm require the programmer to pick up a representation, which simulate the “individual” solution for the problem, so the evolutionary simulation can be executed on the representation. The choice of representation depends on the problem type of Evolutionary Computation, which are binary strings for Genetic Algorithms, real-valued vectors for Evolution Strategies, finite state Machines for Evolutionary Programming, LISP trees for Genetic Programming, and even combinations of them for some specific problems. Normally the representation decides the varia-

tion operators for the algorithm, but it has no relevance with selection operators that only use fitness and so are independent of representation. For different types of representations, there is one thing in common, the encoding of the solution. In biological metaphor, the candidate solution is “phenotype” and the encoding of the solution is “genotype”. The solution should be encoded into “chromosome”, which contains “genes” that are in “loci” and a value of “allele”. This encoding can simplify the solution candidate to be some vector that contains a sort of values. In order to find the global optimum, every feasible solution must be represented in genotype space and later the decoding process will give the feasible solutions to the user.

Later, once the representation and genotype encoding are finished, the process of the execution can be started following the architecture described in above figure. The Initialisation process, which randomly generate candidate solutions, needs to ensure even spread and mixture of possible allele values ,and it can include existing solutions, or use problem-specific heuristics, to “seed” the population. The population usually has a fixed size and is a multi-set of genotypes to represent possible solutions. After the generation of population, the parent selection assigns probabilities of individuals acting as parents based on their fitness so the stochastic nature can aid escape from local optima. The high quality individuals have more possibility to become parents than low quality individuals but no individual will be assign zero possibility to optimise the diversity of offsprings. The main process of the architecture is variation operation which generally do two kinds of operations to candidates, recombination and mutation. Recombination deals with more than on arities and the recombination with exactly two arities is called crossover, whereas mutation deals with only one arity, which means the genotype change itself. There are several debates of variation operators: nowadays most Evolutionary Algorithms use both, and choice of particular variation operators is representation dependant. Programmers must design

their variation operations carefully to produce feasible offsprings for the later survivor selection. After the variation operation, the number of candidates will be larger than the population size (parents + offsprings). The survivor selection will assess the fitness of them and keep the number within the fixed size of population. The process will be executed iterative for every generation until the population reaches designated generation or the optimum of the solutions, then the termination will be executed.[7]

Conclusion

The above architecture is the very general pattern for Evolutionary Computation and normally the elements inside it will not be changed dramatically. Due to the word limit, some very important concepts for the architecture were not discussed in this report such as different variables of variation operators and the strategy for survivor selection. This report is also limited in Evolutionary Computation, which is only one subset of Bio-inspired Computing. The report just gives an introductory insight for Biologically inspired software architecture, more details, wider ranges, and deeper conceptions should be given in future.

Reference

[1] Indiana University. (2013) Biologically Inspired Computing, course description by the University. Available from: http://informatics.indiana.edu/rocha/i-bic/

[2] Flake, G, W. (2002) The Computational Beauty of Nature, online edition. Available from: http://mitpress.mit.edu/sites/default/files/titles/content/cbnhtml/home.html

[3] Bongard, J. (2009) Biologically Inspired Computing. AI REDUX, April 2009, pp. 1-4.

[4] Neumann, F. and Witt, C. (2010) Bio-inspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity, Springer.

[5] Michalewicz, Z. and Fogel, D. B. (2004) How to Solve It: Modern Heuristics, Springer.

[6] Eiben, A. E. and Smith, J. E. (2003) Introduction to Evolutionary Computing, Springer.

[7] Rothlauf, F. (2011) Design of Modern Heuristics - Principles and Applications, Springer.