A GENERAL FRAMEWORK FOR EVOLUTIONARY ALGORITHMS

Kenneth De Jong
George Mason University, Fairfax, Virginia USA

Keywords: evolutionary computation, evolutionary algorithms, genetic algorithms, evolution strategies, evolutionary programming, genetic programming.

Contents

Summary

Since the early 1990s the field of Evolutionary Computation has experienced a tremendous growth of interest resulting in many new ideas, new algorithms and new applications. This growth is both a blessing and a curse. It serves as strong evidence of the usefulness of these techniques, but makes it difficult to see "the big picture" regarding how the many instances of evolutionary algorithms relate to each other, and how to choose among them for a particular application. The purpose of this chapter is to provide a general framework for evolutionary algorithms that helps clarify these relationships and supports informed choices.

1. Introduction

It is probably best to begin by clarifying what we mean by the term "evolutionary algorithm". Although the term evolution is often used in a very general sense to describe systems that are changing slowly and incrementally over time (e.g., an evolving legal system), it also is used in a very specific sense to describe Darwinian-like systems that evolve over time via the interacting processes of reproductive variation and selection. It is this latter sense of the word that is the focus here: algorithms that exhibit Darwinian-like evolutionary change via variation and selection.

That still leaves two possible interpretations to the term "evolutionary algorithm", relating to the goal or intended use of the algorithm. Scientists studying biological evolutionary systems often gain insight into the complex behavior of these systems by constructing computational models capable of simulating the observed evolutionary behavior at some level of fidelity. While important and interesting, that is not the focus here. Rather, the goal is to design new and powerful algorithms for solving difficult

computer science and engineering problems (e.g., search and optimization problems). Hence, the name of the field: Evolutionary Computation.

So, the first step in developing a general framework for evolutionary algorithms is to understand that they are part of a broader class of "nature-inspired" algorithms: algorithms developed not from mathematics or engineering first principles, but rather by understanding and abstracting the information processing mechanisms of systems found in nature. An early example of this is "simulated annealing". More recent examples are "ant colony" and "particle swarm" optimization techniques.

A second important step in developing a general framework for evolutionary algorithms is to understand that they are also part of a broader class of algorithms called "meta-heuristics". Heuristic algorithms incorporate problem-specific information in order to achieve efficiency improvements (e.g., assuming linearity). Meta-heuristic algorithms prove a framework for creating heuristic algorithms capable of exploiting problem-specific information. For example, the notion of a "steepest descent" algorithm is a meta-heuristic algorithm that is instantiated in many different topology- and problem-specific ways.

This, then, is the theme for the remainder of the chapter: to understand in more detail the sense in which evolutionary algorithms are nature-inspired meta-heuristics.

2. Simple Evolutionary Algorithms

In general, evolutionary algorithms are complex, non-linear systems capable of exhibiting a wide range of frequently unexpected and difficult to analyze behavior. The strategy in this chapter is to deal with this issue by starting simple and gradually adding complexity. In the author's mind, the simplest evolutionary algorithms are those that maintain a single fixed-size population of individuals that each represent possible solutions to the problem of interest, where the quality of the solution they represent is their "fitness", and for which there are simple abstract notions of reproductive variation and natural selection. This can be made a bit more precise by restating it in algorithmic pseudo-code:

> Randomly generate an initial population of size *M*.
> Evaluate the fitness of each of these individuals.
>
> Do until some stopping criterion is met:
> Select some parents to produce some offspring.
> Evaluate the fitness of the offspring.
> Reduce the population size to *M* by selecting some individuals to die.
> End Do
>
> Return as the problem solution, the individual with the highest observed fitness.

This algorithmic template is our first example of a surprisingly powerful evolutionary meta-heuristic, namely, one can create highly parallel adaptive search algorithms by simulating at a high level of abstraction the evolutionary dynamics of a natural system.

However, to apply this to a specific class of problems, many additional details need to be specified. For example,

- How an individual in the population represents a problem solution.
- How big the population should be.
- How parents are selected.
- How offspring are produced.
- How many offspring are produced.
- Who dies.

Each such decision impacts the overall behavior of the instantiated evolutionary algorithm. Even the earliest examples of evolutionary algorithms exhibited a wide variety in these choices, resulting in several "canonical" subclasses such as "evolution strategies" [Schwefel, 1995], "evolutionary programming" [Fogel, 1995], and "genetic algorithms" [Holland, 1975].

Today, we have a wide array of evolutionary algorithms and their applications. That provides us with the opportunity to take a step back and lay out a general framework in order to better understand the range of choices and their impact on problem-solving capabilities. That is the goal for the remainder of this chapter. First, we explore the implications of design decisions listed above for simple evolutionary algorithms, and then use that understanding to explore their areas of application as well as various extensions to simple evolutionary algorithms.

2.1. How Individuals Represent Problem Solutions

There's a saying often quoted by realtors that goes something like this: What are the three most important factors that determine the value of a piece of property? Answer: location, location, location!

A similar saying might be said to apply to evolutionary algorithm design: What are the three most important design decisions that impact the performance and effectiveness of an evolutionary algorithm? Answer: representation, representation, representation!

That's perhaps an overstatement, but true more often than not in that, for most problems, the solution space to be searched by an evolutionary algorithm can be represented in a variety of ways, some of which are more conducive to evolutionary search than others. That implies that the choice of internal representation has two possibly conflicting goals: to chose a representation that is in some sense "natural" to a problem's solution space, and one that is "evolution friendly".

If we analyze representations from a biologically-inspired perspective, the choices fall into two general categories: "genotypic" representations and "phenotypic representations". Genotypic representations encode problem solutions in a manner analogous to the biological genetic code. That is, solutions are represented internally as strings formed from some alphabet. For example, traditional Genetic Algorithms represent solutions as binary strings. Reproductive variation is achieved by making changes to the binary string representation and the fitness of new strings is determined

by first decoding the string (i.e., mapping it back into the problem solution space), and then evaluating the quality of that solution.

By contrast, phenotypic representations represent problem solutions directly with no intermediate encoding. For example, Evolution Strategies represent the set of real-valued parameters to be optimized as vectors of real numbers. Reproductive variation is achieved by changing the values stored in these vectors and fitness of the new vector of values is obtained directly by invoking the objective function with these values.

To illustrate this design choice, consider how one might apply an evolutionary algorithm to a job-shop scheduling problem. A canonical Genetic Algorithm approach would focus on designing a mapping of the space of possible solutions to/from binary strings, leaving the reproductive variation operators unchanged. A canonical Evolutionary Strategy approach would represent candidate schedules directly and focus on designing new and appropriate reproductive variation operators.

EAs that are designed to manipulate universal encodings have the advantage of increasing significantly the portability of the EA code from one application to the next in the sense that applying such EAs to a new problem class simply requires writing the encoding/decoding procedures. By contrast, EAs that focus on phenotypic representations often have opportunities to introduce problem-specific efficiency-enhancing features, but at a cost of significant redesign and re-implementation of much of the EA code.

2.2. How Offspring are Produced

A key element of the evolutionary search process is the notion of reproductive variation, namely, how existing solutions (parents) are used to produce new solutions (offspring) that "inherit" many of the features of their parents (exploitation) but are different enough to provide some novelty (exploration). To achieve this, an EA practitioner must choose reproductive operators that manipulate the internal representation of solutions in useful and meaningful ways. This, in turn, emphasizes the tight coupling between the choice of representation and the choice of reproductive operators.

From a biological perspective, reproductive variation is achieved in two rather distinct ways: asexual and sexual reproduction. Asexual reproduction involves cloning single parents and then applying a mutation operator to provide some variability. This is the strategy used in traditional Evolution Strategies and Evolutionary Programming approaches. By contrast, sexual reproduction involves combining elements of more than one parent (generally with a small dose of mutation) to produce offspring that inherit some features from each parent. This is the strategy used in traditional Genetic Algorithms via a recombination operator (crossover) that manipulates binary strings.

From a search perspective these two reproductive strategies differ significantly. Asexual reproduction tends to be more of a local search operator producing offspring in a nearby neighborhood of their parents. By contrast, sexual reproduction tends to be more of a global search operator producing offspring that are frequently quite far from their parents.

Traveling salesperson problems (TSPs) provide a simple but clear illustration of the issues of representation and reproductive variation. The challenge for the traveling salesperson is to minimize the distance he/she travels in visiting each of N cities exactly once and returning home (i.e., a tour). The solution space is easily seen to be all possible permutations of the order in which the N cities are visited, the size of which grows factorially as a function of N .

The most "natural" representation is just an ordered list of cities, a fixed-length vector of size N with each vector element containing a city code. Given this representation, we now need to define reproductive variation operators. Single point mutation operators won't work, nor will standard crossover operators since both produce offspring that are no longer permutations of the cities to be visited. Rather, one needs to choose reproductive operators that provide variation while preserving the permutation property. In the case of mutation, that is not hard to do (e.g., defining mutation as swapping two cities' positions in the tour). It is, however, more difficult to define a useful recombination operator. The evolutionary computation community has explored this issue extensively, concluding in many cases that this is best achieved by adopting a less "natural" representation of TSP tours (see, for example, Whitley (1989).

For effective search in general, one needs a combination of local and global search mechanisms. There is considerable experimental evidence that including both sexual and asexual reproductive operators provides an effective blend of local and global search. As a consequence, many evolutionary algorithms today use a combination of both.

2.3. How Individuals are Selected

In the simple evolutionary algorithm template give earlier, there are two points at which individuals need to be selected: when choosing parents to produce offspring and when choosing which individuals will survive into the next generation. Both of these points are opportunities to use fitness information to bias the search. A natural first thought is to select only the best individuals to reproduce and to survive.

However, that results in a "greedy" EA algorithm. Greedy algorithms are a well-studied class of heuristic search procedures that converge rapidly, but not necessarily to the best solution. For some problem domains, rapid convergence to a nearby local optimum is more than adequate. For others, a slower, more diffuse search is required to avoid getting immediately trapped in a local optimum.

If we adopt the meta-heuristic perspective here, it is important to be able to choose the amount of greediness when applying an evolutionary algorithm to a particular problem. Fortunately, we have a well-studied and well-understood collection of selection mechanisms to choose from:

Bibliography

Branke, J., (2002) *Evolutionary Optimization in Dynamic Environments*, Boston: Kluwer. [An excellent survey of the application of evolutionary algorithms to dynamic optimization problems.]

Coello, C., Veldhuizen, D. and Lamont, G., (2002) *Evolutionary Algorithms for Solving Multi-objective Problems*, Boston: Kluwer. [An excellent survey of the application of evolutionary algorithms to multi-objective optimization problems.]

Deb, K., (2001) *Multi-objective Optimization Using Evolutionary Algorithms*, New York: John Wiley and Sons. [An excellent survey of the application of evolutionary algorithms to multi-objective optimization problems.]

De Jong, K., (2006) *Evolutionary Computation: A Unified Approach*, Cambridge, MA: MIT Press. [An excellent overview of the entire field of evolutionary computation.]

Fogel, L.J. and Owens, A.J. and Walsh, M.J., (1966) *Artificial Intelligence through Simulated Evolution*, New York: John Wiley and Sons. [The first book describing the subclass of evolutionary algorithms called evolutionary programming.]

Fogel, D.B., (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, Piscataway, NJ: IEEE Press. [An updated description of evolutionary programming in the broader context of evolutionary computation.]

Grefenstette, J.J. and Ramsey, C.L. and Schultz, A.C., (1990) Learning Sequential Decision Rules Using Simulation Models and Competition, *Machine Learning* 5(4), 355-381, Boston: Kluwer. [An excellent example of the use of evolutionary algorithms to evolve rule sets.]

Holland, J.H., (1975) *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press. [The first book describing the subclass of evolutionary algorithms called genetic algorithms.]

Kicinger, R., Arciszewski, T. and De Jong, K., (2004) Morphogenesis and Structural Design: Cellular Automata Representations of Steel Structures in Tall Buildings, Proceedings of the Congress on Evolutionary Computation, 411-418, New York: IEEE Press. [An excellent example of the use of cellular automata as a generative representation for evolving complex structures.]

Koppen, M., Wolpert, D.H. and Macready, W.G., (2001) Remarks on a recent paper on the "no free lunch" theorems, *IEEE Transactions on Evolutionary Computation* 5(4), 295-296, New York: IEEE Press. [A good entry point into the "no free lunch" literature.]

Koza, J.R., (1992) *Genetic Programming*, Cambridge, MA: MIT Press. [The first book describing the use of evolutionary algorithms to evolve Lisp programs.]

Lindenmayer, A., (1968) Mathematical Models for Cellular Interaction in Development, *Journal of Theoretical Biology* 18, 280-315. [The original description of what we call today "L-systems".]

Morrison, R., (2004) *Designing Evolutionary Algorithms for Dynamic Environments*, Berlin: Springer-Verlag}. [An excellent description of the application of evolutionary algorithms to dynamic optimization problems.]

Potter, M. and De Jong, K., (2000) Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents, *Evolutionary Computation* 8(1), 1-29, Cambridge, MA: MIT Press. [A good entry point into the literature on co-evolutionary algorithms.]

Praditwong, K. and Yao, X., (2007) A New Multi-objective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm, *Computational Intelligence and Security*, 95-104, Berlin: Springer-Verlag. [A good entry point into the literature on recent developments regarding the use of evolutionary algorithms to solve multi-objective optimization problems.]

Sarma, J., (1998) *An Analysis of Decentralized and Spatially Distributed Genetic Algorithms*, Ph. D. thesis, George Mason University. [The first systematic study of decentralized and spatially distributed evolutionary algorithms.]

Schwefel, H.-P., (1995) *Evolution and Optimum Seeking*, New York: John Wiley and Sons. [The first book in English describing the subclass of evolutionary algorithms called evolution strategies.]

Skolicki, Z. and De Jong, K., (2004) Improving Evolutionary Algorithms with Multi-representation Island Models, Proceedings of PPSN VIII, 420-429, Berlin: Springer-Verlag. [A good entry point into the literature on "island model" evolutionary algorithms.]

Spears, W.M., (1994) Simple Subpopulation Schemes, Proceedings of the Third Conference on Evolutionary Programming, 297-307, World Scientific Publishing. [A good entry point into the literature on the implementation and use of subpopulations in evolutionary algorithms.]

Whitley, D., (1989) Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator, Proceedings of the Third International Conference on Genetic Algorithms, 133-140, San Mateo, CA: Morgan Kaufmann. [An excellent example of the use of evolutionary algorithms as heuristics for NP-hard problems.]

Wolfram, S., (1994) *Cellular Automata and Complexity*, Reading, MA: Addison-Wesley. [The classic reference for the use of cellular automata as generative representations of complex systems.]

Zhan, S., Miller, J.F. and Tyrrell, A.M., (2008) An evolutionary system using development and artificial Genetic Regulatory Networks, Proceeding of the IEEE Congress on Evolutionary Computation, 815-822. [A good entry point into the literature on the use of genetic regulatory network and developmental biology ideas for enhanced evolutionary algorithms.]

Biographical Sketch

Dr. De Jong is a Professor of Computer Science, the Associate Director of the Krasnow Institute, and the director of the Evolutionary Computation Lab, all located on the Fairfax campus of George Mason University. Dr. De Jong's research interests include evolutionary computation, adaptive systems and machine learning. He is an active member of the Evolutionary Computation research community with a variety of papers, Ph.D. students, and presentations in this area. He is also responsible for many of the workshops and conferences on Evolutionary Algorithms. He is the founding editor-in-chief of the journal Evolutionary Computation (MIT Press), and a member of the board of ACM SIGEVO. He is the recipient of an IEEE Pioneer award in the field of Evolutionary Computation and a lifetime achievement award from the Evolutionary Programming Society.

He is the director of a number of research projects involving the use of various kinds of evolutionary algorithms for solving difficult computational problems in science and engineering, including: the modeling of HIV virus evolution, the use of complex adaptive systems models for understanding and identifying computer network intrusions, and the use of evolutionary algorithms to evolve the behaviors of robotic agents.