ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/220801072

Comparison between Genetic Algorithms and Particle Swarm Optimization.

Conference Paper *in* Lecture Notes in Computer Science · March 1998

DOI: 10.1007/BFb0040812 · Source: DBLP

CITATIONS
1,177

READS
2,243

2 authors, including:

Yuhui Shi
Southern University of Science and Technology
190 PUBLICATIONS **26,683** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Evolutionary Computation and Big Data View project

Brain Storm Optimization Algorithms View project

All content following this page was uploaded by Yuhui Shi on 10 February 2015.

The user has requested enhancement of the downloaded file.

Comparison between Genetic Algorithms and Particle Swarm Optimization

Russell C. Eberhart and Yuhui Shi
Department of Electrical Engineering
Indiana University Purdue University Indianapolis
723 W. Michigan St., SL160
Indianapolis, IN 46202
eberhart,shi@engr.iupui.edu

Abstract

This paper compares two evolutionary computation paradigms: genetic algorithms and particle swarm optimization. The operators of each paradigm are reviewed, focusing on how each affects search behavior in the problem space. The goals of the paper are to provide additional insights into how each paradigm works, and to suggest ways in which performance might be improved by incorporating features from one paradigm into the other.

Introduction

Four well-known paradigms currently exist in evolutionary computation: genetic algorithms [5], evolutionary programming [4], evolution strategies [9], and genetic programming [8]. A new evolutionary computation technique, called particle swarm optimization (PSO), inspired by social behavior simulation, was originally designed and developed by Eberhart and Kennedy [2,3,6,7]. In PSO, instead of using more traditional genetic operators, each particle (individual) adjusts its "flying" according to its own flying experience and its companions' flying experience.

This paper compares genetic algorithms and particle swarm optimization. Operators that are used by each paradigm are reviewed. The focus is on how each operator affects the paradigm's behavior in the problem space.

There are, of course, many ways to implement a genetic algorithm (GA). Regardless of the specific implementation, it is generally agreed that GAs utilize one form or another of three operators: selection, crossover, and mutation. We will examine implementations of these operators, and compare them with PSO operators. In this paper, it can be assumed that, in most cases, a basic, binary version of a GA is being referred to, such as the "plain vanilla" GA in [2] or the elementary GA 1-1 in [1].

The authors readily concede that few applications use these basic GA configurations, and a number of modifications to these configurations are also examined. It should be noted that it is not the goal of this paper to compare GAs and PSO in order to declare one or the other as somehow better. Rather, the goals are to provide insights into how GAs and PSO work, and to suggest ways in which performance might be enhanced by incorporating features from one paradigm into the other.

Analysis

In PSO, a particle is analogous to a population member (chromosome) in a GA. Like a GA chromosome, a particle represents a candidate solution to the problem being addressed. Each particle is treated as a point in the D-dimensional problem space. The *i*th particle is represented as $X_I = (x_{i1}, x_{i2}, \ldots, x_{iD})$. The best previous position (the position giving the best fitness value) of the *i*th particle is recorded and represented as $P_I = (p_{i1}, p_{i2}, \ldots, p_{iD})$. The index of the best particle among all the particles in the population is represented by the symbol *g*. The rate of the position change (velocity) for particle *i* is represented as $V_I = (v_{i1}, v_{i2}, \ldots, v_{iD})$. The particles are manipulated according to the following equations:

$$v_{id} = w * v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

By adding the inertia weight *w* into PSO, a new version of PSO is introduced in [10]. The inertia weight *w* is employed to control the impact of the previous history of velocities on the current velocity, thereby influencing the trade-off between global (wide-ranging) and local (fine-grained) exploration abilities of the "flying points." A larger inertia weight facilitates global exploration (searching new areas) while a smaller inertia weight tends to facilitate local exploration to fine-tune the current search area. Suitable selection of the inertia weight provides a balance between global and local exploration abilities and thus requires fewer iterations on average to find the optimum [10]. (In some ways, the name *damping weight* might be more descriptive, but the term *inertia weight* is used in this paper.)

In a GA, each of the three main classes of operations (selection, crossover, and mutation) can be implemented in a number of ways. PSO does not label its operations in the same way as GAs, but analogies exist. These analogies depend, of course, on the implementation of the GA operation. Complicating any comparisons is the fact that the effects of the various operations often vary over the course of a run. (A *run* is defined as the total number of generations of the GA prior to termination. Termination usually occurs either because a prescribed fitness level has been achieved by at least one member of the population, or because the maximum number of generations allowed has been reached.)

Effects of GA crossover, for example, usually vary significantly during a run. At the beginning, the population members are usually randomized, so that crossover can have significant effects, moving a chromosome a relatively large distance in problem space. Toward the end of a run, a population has often converged, meaning that many, if not most, chromosomes have similar structures. Crossover then usually has less effect, and the resulting movements are relatively smaller. Adding another layer of complexity is the fact that the probability of crossover is sometimes varied during a run, often starting out at with a relatively large probability, and ending with a smaller one.

PSO does not have a crossover operation. However, the concept of crossover is represented in PSO because each particle is stochastically accelerated toward its own previous best position, as well as toward the global best position (of the entire

population) or the neighborhood (local) best position, depending on the version of PSO being used.

The crossover concept is also apparent in the behavior of particles that appear approximately midway between "swarms" of particles that are clustering around local best positions, or, occasionally, between successive global best positions. These particles seem to be exploring (for a short time, anyway) a region that represents the geometric mean between two promising regions.

Because of the geometric "feel" of what happens in PSO, it may, in fact, be more analogous to the recombination operator in evolution strategies [9]. The fact that recombination occurs on a parameter-by-parameter basis (recombination is limited to parameter boundaries) further supports this analogy.

The effect of mutation during the run of a GA tends to be opposite to that of crossover in that mutation tends to have less impact near the beginning of a run, and more near the end. That is because the initial population is randomized, so flipping a bit here and there near the beginning may not change things as dramatically as flipping bits near the end of a run, when the population has converged. If the mutation rate is varied during a run, it is common to use a relatively small value in the beginning, and increase it toward the end.

It is theoretically possible for a GA chromosome to reach any point in the problem space via mutation. It is, however, unlikely, particularly near the end of a run. This is because a number of mutations may be needed to reach a distant point. Since mutation rates are typically quite low (0.1–1.0 percent is a common range), several generations of favorable mutations may be needed. Near the end of a run, however, when the population has converged and the average fitness value is high, mutation will quite likely result in a low-fitness chromosome that does not survive the selection process. In fact, the probability of survival decreases geometrically with generations. So even though a number of mutations would bring the chromosome into a high-fitness region, the chromosome never gets there because it doesn't survive selection.

So, even though a GA is theoretically ergodic (there exists a non-zero probability that a chromosome can occupy any state), it is not ergodic in a practical sense because of the multiple steps required. An evolutionary programming (EP) system is truly ergodic, since there is a finite probability that an individual can reach any point in problem space with one jump (in one generation).

The behavior of PSO systems seems to fall somewhere between GA and EP systems in this area. It may be that a PSO particle cannot reach any point in problem space in one iteration, although this might be possible at the beginning of the run, given sufficiently large *Vmax*. But since particles survive intact from one iteration to the next, any particle can eventually go anywhere, given enough iterations. A stronger case can thus be made for the ergodicity of PSO than for GAs.

Because each particle has a velocity, PSO mutation-like behavior is directional, with a kind of momentum built in, especially if *usebest* is activated on the command line. GA

mutation is generally considered to be omnidirectional in that any bit in an individual can be flipped. (Some GA mutation operators, such as bit position-based mutation, can affect the directionality.) EP mutation is also omnidirectional, and includes control of mutation severity on a parameter-by-parameter basis. The difference between *pbest* and the present location has some of this same flavor, but the maximum velocity is the same for all parameters.

The effect of selection in a GA is to support the survival of the fittest, a concept central to all evolutionary algorithms. GA selection can be implemented in one of a number of ways, including roulette wheel selection and tournament selection. Regardless of the selection method used, an *elitist* strategy is often used, which results in the chromosome with the highest fitness value always being copied into the next generation.

PSO does not utilize selection. All particles continue as members of the population for the duration of the run. A particle's path determines its ancestry. PSO is the only evolutionary algorithm that does not incorporate survival of the fittest, which features the removal of some candidate population members (individuals with lower fitness are removed with higher probability).

In the case of a GA, crossover occurs between (usually) randomly selected parents. The evolution of an individual chromosome typically involves exchanging genetic material with quite a few other randomly-selected individuals. In PSO, a particle does not explicitly exchange material with other particles, but its trajectory is influenced by them. A significant difference is that a given particle is influenced only by its own previous best position and that of the best position in the neighborhood or in the global population. In the local version of PSO, a particle is influenced only by one of its topological neighbors. And the topological geometry of the particle swarm remains constant throughout the run, so influences do not come from "random strangers."

Recent work with PSO indicates that by properly setting the inertia weight w, the value of $Vmax$ can be set to the dynamic range of each parameter, and the global maximum can be found more quickly, on average. Even better performance can be achieved by reducing the value of w during a run [11]. The inertia weight thus appears to have characteristics that are reminiscent of the temperature parameter in simulated annealing. A high value of w at the beginning of the run facilitates global search, while a small w tends to localize the search. Since this is "work in progress," it is unknown how universal this approach will be for other problems.

Discussion

A number of ideas have come out of this comparison of GAs and PSO. In fact, it was this kind of thinking that led to the development of the inertia weight which is now part of PSO and is used routinely by the authors.

It is possible to conceive of several ways in which the elitist concept of GAs could be incorporated into PSO. In one sense, it's already there, particularly in the global version of PSO. Also, there is a parameter called *use-better* (which can be selected at

runtime in PSO) that causes a particle to continue in the direction in which it has been going if the current fitness is higher that that of the previous iteration. One way to explicitly implement an elitist strategy would be to carry a "global best" particle (perhaps with a stochastic velocity) into the next iteration; some particle, perhaps the one with the lowest fitness, could be eliminated from the population.

Another approach worthy of investigation is the incorporation of a Gaussian distribution into the stochastic velocity changes in PSO reflecting the way mutation is done in EP. The variance parameter might then play a role similar to that of the current inertia weight.

Finally, the assignment of a maximum velocity on a parameter-by-parameter basis is worthy of consideration. This is analogous to controlling the severity of mutation in GA or EP.

Conclusions

In this paper, we have analyzed the operators of GAs and PSO. It is hoped that the reader has gained some insights into how these paradigms work, and how the performance of one might be improved by incorporating features of the other.

The distinctions between the four main branches of evolutionary computation continue to blur. New approaches continue to be developed. Emphasis should be placed on new hybrid implementations. Standard benchmarking functions can be used to make comparisons, as can practical applications. It is hoped that the understanding of *how* these paradigms work will continue to be studied and improved.

References

1. Davis, L., Ed. (1991), Handbook of Genetic Algorithms, New York, NY: Van Nostrand Reinhold.
2. Eberhart, R. C., Dobbins, R. W., and Simpson, P. K. (1996), Computational Intelligence PC Tools, Boston: Academic Press.
3. Eberhart, R. C., and Kennedy, J. (1995), A new optimizer using particle swarm theory, Proc. Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan), IEEE Service Center, Piscataway, NJ, 39-43.
4. Fogel, L. J. (1994), Evolutionary programming in perspective: the top-down view, in Computational Intelligence: Imitating Life, J.M. Zurada, R. J. Marks II, and C. J. Robinson, Eds., IEEE Press, Piscataway, NJ.
5. Goldberg, D. E. (1989), Genetic Algorithms in Search, Optimization, and Machine Learning, Reading MA: Addison-Wesley.
6. Kennedy, J., and Eberhart, R. C. (1995), Particle swarm optimization, Proc. IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: 1942-1948.
7. Kennedy, J. (1997), The particle swarm: social adaptation of knowledge, Proc. IEEE International Conference on Evolutionary Computation (Indianapolis, Indiana), IEEE Service Center, Piscataway, NJ, 303-308.
8. Koza, J. R. (1992), Genetic Programming: On the Programming of Computers by Means of Natural Selection, Cambridge, MA: MIT Press.

9. Rechenberg, I. (1994), Evolution strategy, in Computational Intelligence: Imitating Life, J. M. Zurada, R. J. Marks II, and C. Robinson, Eds., IEEE Press, Piscataway, NJ.
10. Shi, Y. H., Eberhart, R. C., (1998), A modified particle swarm optimizer, Proc. of 1998 IEEE International Conference on Evolutionary Computation, Anchorage, AK, in press.
11. Shi, Y. H., Eberhart, R. C., (1998), Parameter selection in particle swarm optimization, Proc. EP98: The 7th Ann. Conf. on Evolutionary Programming, San Diego, CA, in press.