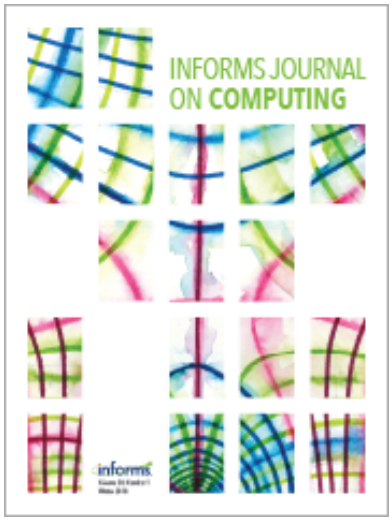

INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Commentary—Genetic Algorithms: A Practitioner's View

David Levine,

To cite this article:
David Levine, (1997) Commentary—Genetic Algorithms: A Practitioner's View. INFORMS Journal on Computing 9(3):256-259. https://doi.org/10.1287/ijoc.9.3.256

INFORMS Journal on Computing
Vol. 9, No. 3, Summer 1997

0899-1499/97/0903-0256 $05.00

COMMENTARY

Genetic Algorithms: A Practitioner's View

DAVID LEVINE / *The Boeing Company, P.O. Box 3707, MS 7L-20, Seattle, WA 98124-2207, Email: david.levine@boeing.com*

The feature article by Colin Reeves provides a useful introduction to genetic algorithms (GAs) for the operations researcher. In this commentary, I will elaborate on several of Reeves' points and introduce some others. My experience with GAs is as a practitioner and software developer, and it is from this perspective that I will comment on Reeves' article.

1. Advantages of Genetic Algorithms

When compared to other optimization methods, one advantage of GAs that stands out is the wide range of problems to which they have been applied. Reeves presents a number of examples of combinatorial optimization including location, packing, partitioning, and scheduling problems. These problems use binary or integer strings and have constraints that are either handled implicitly via the encoding or GA operators, or explicitly through penalty terms or repair mechanisms. In addition, Reeves briefly mentions the use of GAs for continuous optimization. I think it is useful to present some additional examples to emphasize just how general GAs are.

GAs make *no* assumptions about the function to be optimized. All that a GA requires is a performance measure, *some* form of problem representation, and operators that generate new population members. This general approach has been applied to many difficult and novel optimization problems. One such class contains those problems with complicated objective functions. Examples are functions with time-dependent or stochastic coefficients, or those where a user interactively views a solution and provides a subjective "function evaluation." A second class of problems has data structures that are uncommon in optimization problems. Examples are a graph that encodes a neural network topology and a tree that is a symbolic representation of a computer program. These problems are evaluated by executing the neural network or computer program on a set of test problems and returning a measure of how well it performed. GAs have also been applied to problems where the goal is to develop an optimal strategy. Here, the strings represent rules to be followed to achieve a specified objective.

A second major advantage of GAs is that they are a global optimization method. Many problems have a (sometimes large) number of local optima in which (some) traditional operations research (OR) methods may be trapped. When seeded with a large population of randomly initialized strings, GAs start with a uniform sample of the complete search space. Practice has shown that, from this type of initial population, GAs move successfully into high-performing areas of the search space, even in multidimensional search spaces. However, oftentimes, even though they are in a promising part of the search space, GAs either converge slowly or fail to converge to the best solution. For that reason, many practitioners combine the global search capabilities of GAs with the local optimization capabilities of a hill-climbing routine, as discussed in the next section.

For the operations researcher, several other attributes of GAs are advantageous. First, GAs provide flexibility in handling variations of an optimization problem that more specialized methods may have trouble accommodating. For example, in a GA, replacing a linear term in an objective function with a (perhaps more accurate) quadratic term requires only a simple change to the code that provides the function evaluation, not a new solution algorithm as might be required if a linear programming (LP) solver were being used. Second, GAs work directly with integer solutions. For problems where traditionally the LP relaxation is difficult to solve (e.g., set partitioning), the ability to bypass solving the LP may be advantageous. Third, at any iteration, a genetic algorithm contains a population of possible solutions. In practice, it may be more desirable to analyze a family of related solutions, rather than a single point solution. As noted by Arabeyre et al., "The knowledge of a family of good solutions is far more important than obtaining an isolated optimum."[1]

2. Hybrid Genetic Algorithms

An important theme in Reeves' article is that many successful GA applications use problem-specific information. The information may be in the encoding, in specialized mutation and crossover operators, in a repair operator, or in a hill-climbing routine. The last usage, combining a GA with a hill-climber, is often referred to as a *hybrid GA* and is a standard approach used by many GA practitioners.

Genetic algorithms and hill-climbing heuristics have complementary strong and weak points. GAs are good at finding promising areas of the search space, but are not good at converging to the optimal solution from within those areas. On the other hand, hill-climbing heuristics are good at converging to the optimal solution from a nearby point, but lack a global perspective. The experimental evidence of many researchers suggests that combining a GA's ability to widely sample a search space with a local search heuristic's hill-climbing ability is beneficial. Often, this hybrid algorithm will outperform either the GA or the hill-climber.

There are two general schemes for combining a GA with a hill-climber. In the simplest scheme, the GA is run until the

stopping criteria are met, and then a hill-climbing heuristic is applied to each (or just the best) string. In this case, the hill-climber relies upon the GA to supply it with promising starting solutions from which to begin searching.

In the second scheme, a hill-climbing heuristic is integrated with a GA. In this case, we need to decide how often to apply the hill-climbing heuristic, how many strings in the population to apply it to, and whether to use a first-improving or best-improving strategy. My experience in applying a hybrid GA to the set partitioning problem is that a "work quicker, not harder" role for the hill-climbing heuristic is most effective. Reeves cites an example from his research where the best-improving strategy was the better choice. The choice may be problem-dependent and is an area of current research.

3. Parallel Genetic Algorithms

One objection to GAs is their computational cost. The traditional GA replaces *each* string at *each* iteration and this requires a large number of function evaluations. If the mutation or crossover operators are computationally expensive to apply, or a hill-climbing search heuristic is used, or a large population is needed, the GA computation time can be very large.

The high computational cost can be overcome by implementing the GA on a parallel computer. Two parallel implementations are possible. In the case where function evaluations are *not* the dominant cost, or the GA operators themselves are expensive to apply, the iterations of the loop that creates the new generation may be executed in parallel. Most steps in this loop (e.g., crossover, mutation, function evaluation, and hill-climbing) can be executed in parallel. This type of implementation works best on a shared-memory parallel computer so that problem data structures and parameters are easily available to all processors.

In many applications, the dominant computational cost is function evaluations; executing these in parallel can significantly reduce elapsed time. A natural parallel algorithm for this is a master–slave algorithm in which a master process executes all the GA steps *except* the function evaluations. The function evaluations are executed by slave processes. Executing only the function evaluations in parallel requires only modest data distribution requirements (just the strings to be evaluated) and minimal synchronization requirements, thereby permitting the algorithm to perform well on both shared- and distributed-memory parallel computers.

There are two important considerations in determining the performance benefit of a parallel implementation. First, the speedup will vary according to the amount of computation at each GA iteration and the parallel computing overheads. Second, the number of new strings created each generation can limit the degree of parallelism available. In the traditional GA, the entire newly generated population may be evaluated in parallel at each iteration. In the more popular steady-state model, however, only a few new strings are produced, and the parallelism available is more limited. The default in the PGAPack library,[5] where the number of new strings created each iteration is 10 percent of the total population, seems to provide a satisfactory balance between available parallelism and the benefit of an overlapping population.

In addition to the parallel implementation of the single population GA, there is growing interest in parallel GA *models*. In a parallel GA model, the full population exists in a distributed form, and each population member interacts only with a limited set of neighbors. Parallel GA models are classified into either the coarse-grained island model or the fine-grained neighborhood model.

In an island model (IM), a GA population is divided into several subpopulations. An independent GA is run on each subpopulation, each of which is randomly initialized. The migration of strings between subpopulations is a key feature of the IM. First, it allows the distribution and sharing of the above-average strings chosen to migrate. Second, the introduction of migrant strings into the local subpopulation helps to maintain diversity, because the migrant string arrives from a different subpopulation that has evolved independently.

In a neighborhood model (NM), a single population is mapped onto a logical structure (often a Cartesian grid). Each string in the population occupies a discrete point on the structure. The population is partitioned into neighborhoods so that selection and crossover occur only between strings within the same neighborhood, and there is a small overlap among contiguous neighborhoods. The intent is that the neighborhoods maintain diverse populations, and the overlap allows fit strings created through local interactions to propagate throughout the full population.

Both the IM and NM contain several parameters. In the IM, the parameters are: which islands exchange strings, how often strings are exchanged, how many strings are exchanged, which strings migrate, and which strings are replaced. In the NM, the parameters are: the neighborhood topology, the amount of overlap between the neighborhoods, and the string replacement strategy. The parameters of these models affect how quickly fit strings propagate through the population. A good choice of parameter settings will balance the need for subpopulation diversity (to maintain the GA's searching capability) against exploiting the fitter strings found by propagating these fit strings into other subpopulations. Practice indicates that, given the same number of strings, a parallel GA model usually outperforms a single population model because a multiple subpopulation model is better at maintaining diversity and thereby avoiding premature convergence.

Parallel computers are an attractive platform for the implementation of a parallel GA model. The calculations associated with each subpopulation in an island model and each neighborhood in a neighborhood model, may be computed in parallel and this leads to significant savings in elapsed time. This is important because it allows the *global* population size to grow without much increase in elapsed computation time.

4. Limitations of Genetic Algorithms

Probably the most significant limitation of GAs is premature convergence. This condition occurs when most strings in the

population have similar allele values. In this case, applying crossover to similar strings results in another similar string, and no new areas of the search space are explored. Many improvements to the GA, such as alternative selection methods, adaptive mutation rates, and parallel GA models, help to fight premature convergence.

A second limitation of GAs is solving constrained problems. The problem is that there is no guarantee that feasibility will be preserved under crossover or mutation, or even that an initial feasible population can be generated. Reeves discusses a broad range of approaches that have been tried. However, no general mechanism provides a completely satisfactory solution.

A third limitation is the state of GA theory. Currently, there is no proof of asymptotic convergence and optimality and there has been little analysis of the rate of convergence. A user does not know how far a GA-generated solution lies from the optimal solution. A practical consideration that arises is the selection of a stopping criterion. Common stopping criteria include a preset iteration limit, a measure of population similarity, and no change in the best solution found in a specified number of iterations.

Computational cost can also be a limitation. However, it can be lowered in two ways. First, GAs have a natural parallel computing implementation. In fact, given the anticipated growth in multiprocessor PCs, the ability to effectively use multiple processors is a significant advantage of GAs. Second, the commonly used steady-state reproductive strategy, in addition to being more effective on many problems, also significantly reduces the computational cost of a GA run.

5. Inappropriate Usage

In preparing this commentary, I asked colleagues in traditional optimization fields what they saw as the major limitations of GAs. A recurring response was "inappropriate usage." Usually, they had seen or heard of GAs being applied to problems for which other algorithms (e.g., linear programming, network optimization, gradient-based search) were clearly superior. Their concern was that if users experienced excessive computational cost or poor solution quality using GAs, then this would lead to disillusionment with optimization in general.

This is a valid point, but it also raises an important practical issue. For some problems, traditional OR approaches offer (sometimes substantially) better performance than GAs but may be complicated for the nonspecialist to apply. By contrast, GAs are easy to understand, usually straightforward to apply, and, for a wide variety of problems, find good solutions. Choosing among these methods is a large group of scientists, engineers, and managers who are not experts in optimization but have optimization problems that require solutions. They may not recognize that their problem is best solved by a traditional OR algorithm and may find reading the optimization literature a daunting task; or they may believe their problem is too hard for traditional algorithms. As a result, whether through ignorance, technical apprehension, or partial knowledge, they may apply GAs to problems where other algorithms are better suited.

Another type of inappropriate usage, often seen from users new to GAs, is to directly apply the traditional GA defined by Holland[4] and popularized by Goldberg.[3] Reeves points out alternatives to the traditional GA. I would go further and claim that these alternatives are necessary for a successful GA application. Specifically, I recommend that the following should be used.

- A steady-state reproductive strategy, rather than replacing the entire population at each GA iteration.
- Stochastic universal selection or tournament selection, rather than roulette-wheel selection.
- Uniform crossover or two-point crossover, rather than one-point crossover.
- An adaptive mutation rate, rather than a fixed mutation rate.
- A hybrid GA approach that takes into account problem-specific information.
- For continuous optimization problems, an explicit real-valued representation, rather than a binary representation.

The book by Davis[2] provides a nice introduction to many of these topics. It is worth noting that initializing the population in a nonrandom manner has not been successful in practice. GAs almost always do better sampling the full search space and recombining these solutions, rather than starting in a user-specified area of the search space.

Finally, I would like to address the issue of comparing GAs to traditional OR techniques. One reason Reeves cites for the lack of such comparisons is that the practitioner applying the GA may be more concerned with solving the problem at hand. Indeed, in many cases, the practitioner's focus is on comparing the GA to *itself* and on deciding which parameter values and operators give the best results. If a practitioner wants to compare the performance of a GA to an OR technique, many questions arise. Which OR technique should be used? Should it be a metaheuristic like tabu search or simulated annealing, a problem-specific heuristic, or an exact method? Should commercial software be used? What benchmark test problems should be used? These questions are not easy to answer, and present a challenge that practitioners may not have the time or interest to undertake.

6. Conclusions

Genetic algorithms are best seen as another tool in the operations researcher's toolkit. Like any tool, GAs have both strengths and weaknesses. Their strengths are their robustness, wide domain of applicability, global search capability, and natural parallel implementation. Their weaknesses are premature convergence, solving highly constrained problems, limited theoretical underpinnings, and (sometimes) computational cost.

Because of their wide applicability and ease of use, GAs have sometimes been applied inappropriately to problems for which there are better solution methods. Furthermore, users often do not take advantage of the many improvements to the traditional GA that have been developed. How-

ever, when properly implemented and combined with problem-specific approaches, a robust algorithm that performs well on many problems can result.

References

1. J. ARABEYRE, J. FEARNLEY, F. STEIGER, and W. TEATHER, 1969. The Airline Crew Scheduling Problem: A Survey. *Transportation Science 3*, 140–163.
2. L. DAVIS, 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.
3. D. GOLDBERG, 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.
4. J. HOLLAND, 1992. *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA.
5. D. LEVINE, 1995. *Users Guide to the PGAPack Parallel Genetic Algorithm Library*, ANL-95/18, Argonne National Laboratory. Available from URL ftp://info.mcs.anl.gov/pub/pgapack/pgapack.tar.Z.