A Perspective on Evolutionary Computation

Zbigniew Michalewicz

Department of Computer Science
University of North Carolina
Charlotte, NC 28223, USA

Abstract. During the last three decades there has been a growing interest in algorithms which rely on analogies to natural processes. The emergence of massively parallel computers made these algorithms of practical interest. The best known algorithms in this class include evolutionary programming, genetic algorithms, evolution strategies, simulated annealing, classifier systems, and neural networks.
In this paper we discuss a subclass of these algorithms—those which are based on the principle of evolution (survival of the fittest). A common term, recently accepted, refers to such techniques as 'evolutionary computation' methods.
The paper presents a perspective of the field of evolutionary computation. It discusses briefly the concept of evolutionary computation, presents the author's first experience with these methods, provides a discussion on relationship between evolutionary computation techniques and the problem specific knowledge, and identifies some current critical issues.

1 Introduction

The field of evolutionary computation is approaching a stage of maturity. There are several, well established international conferences that attract hundreds of participants (International Conferences on Genetic Algorithms—ICGA [28, 29, 45, 7, 24], Parallel Problem Solving from Nature—PPSN [49, 33], Annual Conferences on Evolutionary Programming—EP [20, 21, 50]); new annual conferences are getting started, e.g., IEEE International Conferences on Evolutionary Computation [42]. Also, there are many workshops, special sessions, and local conferences every year, all around the world. A new journal, *Evolutionary Computation* (MIT Press) [8], is devoted entirely to evolutionary computation techniques; many other journals organized special issues on evolutionary computation (e.g., [18, 36]). Many excellent tutorial papers [5, 6, 43, 53, 19] and technical reports provide more-or-less complete bibliographies of the field [1, 27, 44, 39]. There is also *The Hitch-Hiker's Guide to Evolutionary Computation* prepared initially by Jörg Heitkötter and currently by David Beasley [30], available on comp.ai.genetic interest group (Internet), and a new text, *Handbook of Evolutionary Computation*, is currently being prepared [4].

This paper discusses two issues related to evolutionary computation. First, a connection between the problem specific knowledge, representation of individuals, operators and the evolutionary computation techniques is explored to some

degree; this part provides ground for further discussion on similarities and differences between major evolutionary techniques, like genetic algorithms (GAs), evolutionary strategies (ES), and evolutionary programming (EP). Second, we concentrate on the current trends in the field.

The paper is organized as follows. The next section discusses briefly the concept of evolutionary computation. Section 3 presents the author's first experience with evolutionary techniques, and Section 4 provides a discussion on relationship between evolutionary computation techniques and the problem specific knowledge. The final section identifies some current critical issues in the field.

2 Evolutionary computation

Evolutionary computation algorithms are based on the principle of evolution (survival of the fittest). In these algorithms a population of individuals (potential solutions) undergoes a sequence of unary (mutation type) and higher order (crossover type) transformations. These individuals strive for survival: a selection scheme, biased towards fitter individuals, selects the next generation. After some number of generations, the program converges—the best individual represents a near-optimum solution.

The two most important issues in the evolution process are population diversity and selective pressure. These factors are strongly related: an increase in the selective pressure decreases the diversity of the population, and vice versa. In other words, strong selective pressure "supports" the premature convergence of the search and a weak selective pressure can make the search ineffective. Different evolutionary techniques use different scaling methods and different selection schemes (e.g., proportional selection, ranking, tournament) to strike a balance between these two factors.

However, the structure of any evolutionary computation algorithm is very much the same; a sample structure is shown in Figure 1.

The evolutionary algorithm maintains a population of individuals, $P(t) = \{x_1^t, \ldots, x_n^t\}$ for iteration t. Each individual represents a potential solution to the problem at hand, and is implemented as some data structure S. Each solution x_i^t is evaluated to give some measure of its "fitness". Then, a new population (iteration $t+1$) is formed by selecting the more fit individuals (select step). Some members of the new population undergo transformations (alter step) by means of "genetic" operators to form new solutions. There are unary transformations m_i (mutation type), which create new individuals by a small change in a single individual ($m_i : S \rightarrow S$), and higher order transformations c_j (crossover type), which create new individuals by combining parts from several (two or more) individuals ($c_j : S \times \ldots \times S \rightarrow S$). After some number of generations the algorithm converges—it is hoped that the best individual represents a near-optimum (reasonable) solution.

The data structure S used for a particular problem and a set of 'genetic' operators constitute the most essential components of any evolutionary algorithm. For example, the original genetic algorithms, devised to model *adaptation*

```
procedure evolutionary algorithm
begin
    t ← 0
    initialize P(t)
    evaluate P(t)
    while (not termination-condition) do
    begin
        t ← t + 1
        select P(t) from P(t − 1)
        alter P(t)
        evaluate P(t)
    end
end
```

Fig. 1. The structure of an evolutionary algorithm

processes, mainly operated on binary strings and used recombination operator with mutation as a background operator [31]. Evolution strategies [48] used real variables[1] and aimed at *numerical optimization*. Because of that, the individuals incorporated also a set of strategic parameters. Evolution strategies relied mainly on mutation operator (Gaussian noise with zero mean); only recently a discrete recombination was introduced for object variables and intermediate recombination—for strategy parameters. On the other hand, evolutionary programming techniques [23] aimed at building a system to solve *prediction tasks*. Thus they used a finite state machine as a chromosome and 5 mutation operators (change the output symbol, change a state transition, add a state, delete a state, or change the initial state).[2]

In the next section we discuss a particular evolutionary system which is hard to classify into categories of genetic algorithms, evolution strategies or evolutionary programming. The system was developed for the transportation problem and it uses matrix representation for chromosomes with a problem-specific mutation and arithmetical crossover. It is unclear whether the system can be classified as genetic algorithm without any support of scheme theorem or building-block hypothesis. It uses floating point representation (as evolution strategies do), however, it does not incorporate any control parameters into the structure of its chromosomes. The matrix representation may suggest the case of evolution programming since finite state machines were represented as matrices, however, the problem is clearly 'to optimize' and not 'to predict'. After discussing the features of the evolutionary system for the transportation problem (next section) we will return to the general discussion on similarities and differences between various techniques from the perspective of incorporating

[1] However, they started with integer variables as an experimental optimum-seeking method.

[2] 'Change', 'add', and 'delete' are specialized versions of mutation.

problem-specific knowledge into these algorithms (Section 4).

3 Evolutionary algorithm for the transportation problem

In 1991 two papers were published on GA-based systems for the transportation problem: for linear [52] and nonlinear [37] cases. To the best of the author's knowledge they were the first GA-based system to use non-string chromosome structures;[3] specialized 'genetic' operators were introduced to preserve feasibility of solutions. This section summarizes this research by stating the problem and discussing possible representations and operators.

The transportation problem is one of the simplest constrained optimization problems that have been studied. It seeks the determination of a minimum cost transportation plan for a single commodity from a number of sources to a number of destinations. A destination can receive its demand from one or more sources. The objective of the problem is to determine the amount to be shipped from each source to each destination such that the total transportation cost is minimized.

If the transportation cost on a given route is directly proportional to the number of units transported, we have a *linear transportation problem*. Otherwise, we have a *nonlinear transportation problem*.

Assume there are n sources and k destinations. The amount of supply at source i is $source(i)$ and the demand at destination j is $dest(j)$. The cost of transporting flow x_{ij} from source i to destination j is given as a function f_{ij} . Thus the total cost is a separable function of the individual flows rather than interactions between them. The transportation problem is given as:

$$\text{minimize } total = \sum_{i=1}^{n} \sum_{j=1}^{k} f_{ij}(x_{ij})$$

subject to

$$\begin{gathered}\sum_{j=1}^{k} x_{ij} \leq source(i), \text{ for } i = 1, 2, \ldots, n, \\ \sum_{i=1}^{n} x_{ij} \geq dest(j), \text{ for } j = 1, 2, \ldots, k, \\ x_{ij} \geq 0, \text{ for } i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, k.\end{gathered}$$

The first set of constraints stipulates that the sum of the shipments from a source cannot exceed its supply; the second set requires that the sum of the shipments to a destination must satisfy its demand.

The above problem implies that the total supply $\sum_{i=1}^{k} source(i)$ must at least equal total demand $\sum_{j=1}^{n} dest(j)$. When total supply equals total demand (total flow), the resulting formulation is called a *balanced* transportation problem. It differs from the above only in that all constraints are equations; that is,

$$\begin{gathered}\sum_{j=1}^{k} x_{ij} = source(i), \text{ for } i = 1, 2, \ldots, n, \\ \sum_{i=1}^{n} x_{ij} = dest(j), \text{ for } j = 1, 2, \ldots, k.\end{gathered}$$

[3] Fogel, Owens & Walsh [22] in their EP technique represented finite state machines as matrices.

In constructing an evolutionary algorithm for the transportation problem, a selection of appropriate data structure S together with the set of appropriate 'genetic' operators is of utmost importance. It is because there are several hard constraints to be satisfied. As stated by Davis [12]:

> "Constraints that cannot be violated can be implemented by imposing great penalties on individuals that violate them, by imposing moderate penalties, or by creating decoders of the representation that avoid creating individuals violating the constraint. Each of these solutions has its advantages and disadvantages. If one incorporates a high penalty into the evaluation routine and the domain is one in which production of an individual violating the constraint is likely, one runs the risk of creating a genetic algorithm that spends most of its time evaluating illegal individuals. Further, it can happen that when a legal individual is found, it drives the others out and the population converges on it without finding better individuals, since the likely paths to other legal individuals require the production of illegal individuals as intermediate structures, and the penalties for violating the constraint make it unlikely that such intermediate structures will reproduce. If one imposes moderate penalties, the system may evolve individuals that violate the constraint but are rated better than those that do not because the rest of the evaluation function can be satisfied better by accepting the moderate constraint penalty than by avoiding it. If one builds a "decoder" into the evaluation procedure that intelligently avoids building an illegal individual from the chromosome, the result is frequently computation-intensive to run. Further, not all constraints can be easily implemented in this way."

There are other possibilities as well. Sometimes it is worthwhile to design a repair algorithm which would 'correct' an infeasible solution into a feasible one. In such cases, there is an additional question to be resolved: it is whether the repaired chromosome should replace the original one in the population, or rather the repair process is run only for evaluation purpose.[4] Also, there is a possibility of using a data structure appropriate for the problem at hand together with the set of specialized operators; such approach was described in detail in [35]. We will examine briefly these possibilities in turn.

It is possible to build a "classical" genetic algorithm for the transportation problem, where chromosomes (i.e. representation of solutions) are bit strings—lists of 0's and 1's. A straightforward approach is to create a vector $\langle v_1, v_2, \ldots, v_p \rangle$ ($p = n \cdot k$), such that each component v_i ($i = 1, 2, \ldots, p$) is a bit vector $\langle w_0^i, \ldots, w_s^i \rangle$ representing a value associated with row j and column m in the allocation matrix, where $j = \lfloor (i-1)/k + 1 \rfloor$ and $m = (i-1) \; mod \; k + 1$.

[4] Recently Orvosh and Davis [40] reported so-call 5% rule which states that if replacing original chromosomes with a 5% probability, the performance of the algorithm is better than if replacing with any other rate. In particular, it is better than with 'never replacing' or 'always replacing' strategies. However, the rule has some exceptions [35].

However, it is difficult to design a meaningful set of penalty functions. If penalty functions are moderate, the system often returns [34] an infeasible solution $x_{ij} = 0.0$ for all $1 \leq i \leq n$, $1 \leq j \leq k$, which yields the "optimum" transportation cost (zero)! It seems that high penalties have much better chances to force solutions into a feasible region of the search space, or at least to return solutions which are 'almost' feasible. However, it should be stressed that

- with high penalties very often the system would settle for the first feasible solution found, or
- if a solution is 'almost' feasible, the process of finding a 'good' correction can be quite complex for high dimensional problems. We can think about this step as a process of solving a new transportation problem with modified marginal sums (which represent differences between actual and required totals), where variables, say, δ_{ij}, represent respective corrections to original variables x_{ij}.

We conclude that the penalty function approach is not the most suitable for solving constrained problems of this type.

Judging from the previous paragraph it should be clear that the 'repair algorithm' approach has also slim chances to succeed. Even if the initial population consists of feasible solutions only, there are some serious difficulties. For example, let us consider a required action when a feasible solution undergoes mutation. The mutation is usually defined as a change in a single bit in a solution-vector. This would correspond to a change of one value, v_i. This, in turn, would trigger a series of changes in different places (at least 3 other changes) in order to maintain the constraint equalities (note also, that we always have to remember in which column and row a change was made—despite a vector representation we think and operate in terms of rows and columns).

There are some other open questions as well. Assume that two random points (v_i and v_m, where $i < m$) are selected such that they do not belong to the same row or column. Let us assume that v_i, v_j, v_k, v_m ($i < j < k < m$) are components of a solution-vector (selected for mutation) such that v_i and v_k as well as v_j and v_m belong to a single column, and v_i and v_j as well as v_k and v_m belong to a single row. That is, in matrix representation:

$$
\begin{array}{ccccc}
\ldots & \cdot & \ldots & \cdot & \ldots \\
\ldots & \cdot & \ldots & \cdot & \ldots \\
\ldots & v_i & \ldots & v_j & \ldots \\
\ldots & \cdot & \ldots & \cdot & \ldots \\
\ldots & \cdot & \ldots & \cdot & \ldots \\
\ldots & v_k & \ldots & v_m & \ldots \\
\ldots & \cdot & \ldots & \cdot & \ldots \\
\ldots & \cdot & \ldots & \cdot & \ldots
\end{array}
$$

Now in trying to determine the smallest change in the solution vector we have a difficulty. If we increase the value v_i by a constant C we have to decrease each of the values v_j and v_k by the same amount. What happens if $v_j < C$ or

$v_k < C$? We could set $C = min(v_i, v_j, v_k)$, but then most mutations would result in no change, since the probability of selecting three non-zero elements would be close to zero for solutions from the surface of the simplex. Thus methods involving single bit changes result in inefficient mutation operators with complex expressions for checking the corresponding row or column of the selected element.

The situation is even worse if we try to define a crossover operator. Breaking a vector at a random point can result in the appearance of numbers v_i larger than all $sour(i)$ and $dest(j)$, obviously violating constraints. Even if we design a method to provide that all numbers in the solution-vectors of offspring resulting from crossovers are in a reasonable range it is more than likely that these new solutions would still violate the constraints. If we try to modify these solutions to obey all constraints we would then lose all similarities with the parents. Moreover, the way to do this is far from obvious. We conclude that the repair algorithm approach is not the most suitable for solving constrained problems of this type.

The third possibility, the use of decoders, is almost out of question. Decoders are used mainly for discrete optimization problems (e.g., knapsack problem, see [35]), and it might be difficult to design a decoder scheme for continuous case. A GA-based system based on decoders was built for the linear case of the transportation problem [52], but is impossible to use the proposed decoder for the nonlinear case [37]. It was clear that decoders were not the most suitable for solving constrained problems of this type.

The general conclusion from the above discussion is that the vector representation (whether used with penalty functions, repair algorithms, or decoders) is not the best data structure for the transportation problem. Perhaps the most natural representation of a solution for this problem is a two dimensional structure. After all, this is how the problem is presented and solved by hand. In other words, a matrix $V = (x_{ij})$ $(1 \leq i \leq k,\ 1 \leq j \leq n)$ may represent a solution; each x_{ij} is a real number.

It is relatively easy to initialize a population so it contains only feasible individuals. In [37] a particular *initialization* procedure is discussed which introduces as many zero elements as possible. Such *initialization* procedure can be used to define a set of 'genetic' operators (two mutations and one crossover) which would preserve feasibility of solutions:

mutation-1. Assume that $\{i_1, i_2, \ldots, i_p\}$ is a subset of $\{1, 2, \ldots, k\}$, and $\{j_1, j_2, \ldots, j_q\}$ is a subset of $\{1, 2, \ldots, n\}$ such that $2 \leq p \leq k$, $2 \leq q \leq n$.

Denote a parent for mutation by the $(k \times n)$ matrix $V = (x_{ij})$. Then we can create a $(p \times q)$ submatrix $W = (w_{ij})$ from all elements of the matrix V in the following way: an element $x_{ij} \in V$ is in W if and only if $i \in \{i_1, i_2, \ldots, i_p\}$ and $j \in \{j_1, j_2, \ldots, j_q\}$ (if $i = i_r$ and $j = j_s$, then the element x_{ij} is placed in the r-th row and s-th column of the matrix W).

Now we can assign new values $sour_W[i]$ and $dest_W[j]$ $(1 \leq i \leq p,\ 1 \leq j \leq q)$ for matrix W:

$$sour_W[i] = \sum_{j \in \{j_1, j_2, \ldots, j_q\}} x_{ij},\ 1 \leq i \leq p,$$

$$dest_W[j] = \sum_{i \in \{i_1, i_2, \ldots, i_p\}} x_{ij}\,,\ 1 \leq j \leq q.$$

We can initialize the matrix W (procedure *initialization*) so that all constraints $sour_W[i]$ and $dest_W[j]$ are satisfied. Then we replace corresponding elements of matrix V by new elements from the matrix W. In this way all the global constraints ($sour[i]$ and $dest[j]$) are preserved.

mutation-2. This operator is identical to mutation-1 except that in recalculating the contents of the chosen sub-matrix W, a modified version of the *initialization* routine is used (for details, see [37]) which avoids zero entries by selecting values from a range.

crossover. Starting with two parents (matrices U and V) the arithmetical crossover operator will produce two children X and Y, where $X = c_1 \cdot U + c_2 \cdot V$ and $Y = c_1 \cdot V + c_2 \cdot U$ (where $c_1, c_2 \geq 0$ and $c_1 + c_2 = 1$). As the constraint set is convex this operation ensures that both of children are feasible if both parents are. This is a significant simplification of the linear case where there was an additional requirement to maintain all components of the matrix as integers.

It is clear that all above operators maintain feasibility of potential solutions: (arithmetical) crossover produces a point between two feasible points of the convex search space and both mutations were restricted to submatrices only to ensure no change in marginal sums.

The experimental results of the developed system are discussed in [37, 35, 34]. It is worthwhile to underline, that the results were much better than these obtained from the GAMS (General Algebraic Modeling System)[5] with MINOS optimizer.

The main conclusions from the experiments on the transportation problem were as follows:

- the most interesting problems are constrainted ones,
- coding of chromosome structures S need not be binary,
- the 'genetic' operators need not be 'genetic' and may incorporate the problem specific knowledge, and
- the problem-specific knowledge incorporated into the system enhances an algorithm's performance and narrows its applicability.

In the next section we discuss further the last conclusion.

4 Evolutionary computation techniques and the problem specific knowledge

The idea of incorporating a problem specific knowledge in genetic algorithms is not new and has been recognized for some time. Several researchers have dis-

[5] GAMS is a package for the construction and solution of mathematical programming models [9]. GAMS represents a typical example of an industry-standard efficient gradient-controlled method.

Fig. 2. Efficiency/problem spectrum and a hierarchy of EAs

The specialized algorithm Q is suitable for a particular problem P. In general, it is possible to construct a family of evolutionary algorithms EA_i, each of which would 'solve' the problem P. The term 'solve' means 'provide a reasonable solution', i.e., a solution which need not, of course, be optimal, but is feasible (it satisfies problem constraints). The evolutionary algorithm EA_5 is the strongest one (i.e., the most problem specific) and it addresses the problem P only. The system EA_5 will not work well for any modified version of the problem (e.g., after adding a new constraint or after changing the size of the problem). The next evolutionary algorithm, EA_4, can be applied to some (relatively small) class of problems, which includes the problem P; other evolutionary algorithms EA_3 and EA_2 work on larger domains, whereas EA_1 is the weakest method (i.e., domain independent) and can be applied to any optimization problem[6] (in [34] we discussed five such evolutionary algorithms for a given problem P: the transportation problem).

Note that we talk about evolutionary algorithms in general, without specifying whether such system is a genetic algorithm, an evolution strategy, an evolutionary programming technique, or other. The reason was explained partially in the previous section, where we discussed a particular evolutionary system for the transportation problem, which was hard to classify into these categories. It seems that neither of the evolutionary techniques is perfect (or even robust) across the problem spectrum; only the whole family of algorithms based on evolutionary computation concepts (i.e., evolutionary algorithms) have the property of robustness. For example, if we concentrate on classical GAs (binary representation, crossover and mutation) and take into account that [15]:

> "...virtually all decision making situations involve constraints. What distinguish various types of problems is the form of these constraints. Depending on how the problem is visualized, they can arise as rules, data dependencies, algebraic expressions, or other forms",

it is clear, that major modifications should be incorporated in the GA to obtain satisfactory results. Because of these modifications, we deal rather with some evolutionary algorithms (as discussed in the previous section). Classical GAs aim mainly at adaptive problems; their applications in the area of Artificial Life prove this point. Similary, evolution strategies aim at numerical optimization, whereas evolutionary programming—at task prediction.[7] In the same time, application of GAs to numerical optimization results often in systems more similar to evolution strategies than genetic algorithms [35]! In some sense we can look at the main three instances of evolutionary computation (i.e, genetic algorithms, evolution strategy, evolutionary programming) from the perspective of incorporating problem-specific knowledge into evolutionary algorithm. Each of these techniques incorporates the problem-specific knowledge in their data structures (binary string, floating point string representing values of variables and values

[6] Note, that only EA_5, EA_3, and EA_1 are displayed in Figure 2.

[7] Only recently evolutionary programming techniques were extended to handle numerical optimization problems [17].

Fig. 3. Efficiency/problem spectrum and GA, ES, EP

In Figure 3 the problem A might be a problem of modeling ecological system, the problem B—numerical optimization problem, and C—for example, prisoner's dilemma problem.

In 1985 De Jong [14] addressed this issue from the perspective of genetic algorithms:

> "What should one do when elements in the space to be searched are most naturally represented by more complex data structures such as arrays, trees, digraphs, etc. Should one attempt to 'linearize' them into a string representation or are there ways to creatively redefine crossover and mutation to work directly on such structures."

It seems that more than often the second possibility (i.e., redefining crossover(s) and mutation(s) to work directly on complex structures) provides with better experimental results. This point of view is shared by Koza [32]:

> "Representation is the key issue in genetic algorithm work because the representation scheme can severely limit the window by which the system

observes its world."

Koza developed a new methodology, named 'genetic programming' (GP), which provides a way to run a search of the space of possible computer programs for the best one (the most fit). In other words, a population of computer programs is created, individual programs compete against each other, weak programs die, and strong ones reproduce (crossover, mutation).... It is important to note that the structure which undergoes evolution is a hierarchically structured computer program.[8] The search space is a hyperspace of valid programs, which can be viewed as a space of rooted trees. Each tree is composed of functions and terminals appropriate to the particular problem domain; the set of all functions and terminals is selected *a priori* in such a way that some of the composed trees yield a solution. The initial population is composed of such trees; construction of a (random) tree is straightforward. The evaluation function assigns a fitness value which evaluates the performance of a tree (program). The evaluation is based on a preselected set of test cases; in general, the evaluation function returns the sum of distances between the correct and obtained results on all test cases. The primary operator is a crossover that produces two offspring from two selected parents. The crossover creates offspring by exchanging subtrees between two parents. There are other operators as well: mutation, permutation, editing, and a define-building-block operation [32]. The paradigm of genetic programming constitutes another important example of incorporating problem specific knowledge by means of data structures and operators used.

5 Current critical issues

There are several issues which deserve a special attention of the evolutionary computation community. As De Jong observed [13] recently:

> "... the field had pushed the application of simple GAs[9] well beyond our initial theories and understanding, creating a need to revisit and extend them."

In this section we examine a few important directions in which we expect a lot of activities and significant results.

Function optimization

For many years, most evolutionary techniques were evaluated and compared with each other in the domain of function optimization. In the view of the previous section, it is not surprising that quite often ES outperformed simple GAs; this was also the case with EP techniques when they were extended to handle numerical optimization problems [17]. To adapt a GA to the task of function optimization it was necessary to extend simple GA by additional features; these included

[8] Actually, Koza has chosen LISP's S-expressions for all his experiments.

[9] This is true not only for GAs, but for any evolutionary technique.

- dynamic scaled fitness,
- rank-proportional selection,
- inclusion of elitist strategy,
- adaptation of various parameters of the search (probabilities of operators, population size, etc.)
- various representations: binary or Gray coding (plus Delta Coding or Dynamic Parameter Encoding), and floating point representation,
- new operators (for binary and floating point representation).

Most of these modifications pushed a simple GA away from its theoretical basis, however, they enhanced the performance of the systems in a significant way. It seems that the domain of function optimization would remain as the primary test-bed for many new features. It is expected that new theories of evolutionary techniques for function optimization would emerge (e.g., breeder genetic algorithms [38]). Additionally, we should see a progress in

- development of constraint-handling techniques. This is a very important area in general, and for function optimization, in particular; most real problems of function optimization involve constraints. However, so far very few techniques were proposed, analysed, and compared with each other.
- development of systems for large-scale problems. Until now, most experiments assume relatively small number of variables. It would be interesting to analyse how evolutionary techniques scale up with the problem size for problems with thousand variables.
- development of systems for mathematical programming problems. Very little work was done in this area. There is a need to investigate evolutionary systems to handle integer/Boolean variables, to experiment with mixed programming as well as integer programming problems.

Representation and operators

Traditionally, GAs work with binary strings, ES—with floating point vectors, and EP—with finite state machines (represented as matrices), whereas GP techniques use trees as a structure for the individuals. However, there is a need for a systematic research on

- representation of complex, non-linear objects of varying size, and, in particular, representation of 'blueprints' of complex objects, and
- development of evolutionary operators for such objects at the genotype level.

This direction can be perceived as a step towards building complex hybrid evolutionary system which incorporate additional search techniques. For example, it seems worthwhile to experiment with Lamarckian operators, which would improve an individual during its lifetime—consequently, the improved, "learned" characteristics of such individual would be passed to the next generation.

Non random mating

Most current techniques which incorporate crossover operator use random mating, i.e, mating, where individuals are paired randomly. It seems that with the trend of movement from simple to complex systems, the issue of non random mating would be of growing importance. There are many possibilities to explore; these include introduction of sex or "family" relationships between individuals. Some simple schemes were already investigated by several researchers (e.g., Eshelman's incest prevention technique [16]), however, the ultimate goal seems to be to evolve rules for non random mating.

Self-adapting systems

Since evolutionary algorithms implement the idea of evolution, it is more than natural to expect some self-adapting characteristics of these techniques. Apart from evolutionary strategies, which incorporate some of its control parameters in the solution vectors, most other techniques use fixed representations, operators, and control parameters. One of the most promising research area is based on inclusion of self adapting mechanisms within the system for

- representation of individuals (as proposed by Shaefer [51]; the Dynamic Parameter Encoding technique [47] and messy genetic algorithms [26] also fall into this category),
- operators. It is clear that different operators play different roles at different stages of the evolutionary process. The operators should adopt (e.g., adaptive crossover [46]). This is true especially for time-varying fitness landscapes.
- control parameters. There were already experiments aimed at these issues: adaptive population sizes [2] or adaptive probabilities of operators [11]. However, much more remains to be done.

Co-evolutionary systems

There is a growing interest in co-evolutionary systems, where more than one evolution process takes place: usually there are different populations there (e.g., additional populations of parasites or predators) which interact with each other. In such systems the evaluation function for one population may depend on the state of the evolution processes in the other population(s). This is an important topic for modeling artificial life, some business applications, etc. Also, co-evolutionary systems might be important for approaching large-scale problems [41].

Diploid/polyploid versus haploid structures

Diploidy (or polyploidy) can be viewed as a way to incorporate memory into the individual's structure. Instead of single chromosome (haploid structure) representing a precise information about an individual, a diploid structure is made up of a pair of chromosomes: the choice between two values is made by some dominance function. The diploid (polyploid) structures are of particular significance in non-stationary environments (i.e., for time-varying objective functions)

and for modeling complex systems (possibly using co-evolution models). However, there is no theory to support the incorporation of a dominance function into the system; there is also quite small experimental data in this area.

Parallel models

Parallelism promises to put within our reach solutions to problems untractable before; clearly, it is one of the most important areas of computer science. Evolutionary algorithms are very suitable for parallel implememtations; as Goldberg [25] observed:

> "In a world where serial algorithms are usually made parallel through countless tricks and contortions, it is no small irony that genetic algorithms (highly parallel algorithms) are made serial through equally unnatural tricks and turns."

However, during the last 5 years several parallel models of evolutionary techniques were investigated. They can be classified into several categories (e.g., synchronous vs. asynchronous, or master-slave vs. network, fine-grain vs. coarse-grain, continuous vs. discontinuous neighborhood). Many experimental results indicate a significant speedup in the processing time, however, there is very little theory to assist in understanding parallel systems.

Parallel models can also provide a natural embedding for other paradigms of evolutionary computation, like non random mating, some aspects of self-adaptation, or co-evolutionary systems.

6 Conclusions

It might be meaningful to conclude the paper with a citation from the recent meeting of the evolutionary computation community, which provides the main direction for future research [23]:

> "If the aim is to generate artificial intelligence, that is, to solve new problems in new ways, then it is inappropriate to use any fixed set of rules. The rules required for solving each problem should simply evolve..."

Acknowledgements.

This material is based upon work supported by the National Science Foundation under Grant IRI-9322400. The author wishes to thank Sita S. Raghavan for comments on the first draft of the paper.

References

1. Alander, J.T., *An Indexed Bibliography of Genetic Algorithms: Years 1957–1993*, Department of Information Technology and Production Economics, University of Vaasa, Finland, Report Series No.94-1, 1994.

2. Arabas, J., Michalewicz, Z., and Mulawka, J., *GAVaPS — a Genetic Algorithm with Varying Population Size*, in [42].
3. Bäck, T., and Hoffmeister, F., *Extended Selection Mechanisms in Genetic Algorithms*, in [7], pp.92–99.
4. Bäck, T., Fogel, D., and Michalewicz, Z. (Editors), *Handbook of Evolutionary Computation*, IOP Press, in preparation.
5. Beasley, D., Bull, D.R., and Martin, R.R., *An Overview of Genetic Algorithms: Part 1, Foundations*, University Computing, Vol.15, No.2, pp.58–69, 1993.
6. Beasley, D., Bull, D.R., and Martin, R.R., *An Overview of Genetic Algorithms: Part 2, Research Topics*, University Computing, Vol.15, No.4, pp.170–181, 1993.
7. Belew, R. and Booker, L. (Editors), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1991.
8. De Jong, K.A., (Editor), *Evolutionary Computation*, MIT Press, 1993.
9. Brooke, A., Kendrick, D., and Meeraus, A., *GAMS: A User's Guide*, The Scientific Press, 1988.
10. Davis, L., (Editor), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA, 1987.
11. Davis, L., *Adapting Operator Probabilities in Genetic Algorithms*, in [45], pp.61–69.
12. Davis, L. and Steenstrup, M., *Genetic Algorithms and Simulated Annealing: An Overview*, in [10], pp.1–11.
13. De Jong, K., *Genetic Algorithms: A 10 Year Perspective*, in [28], pp.169–177.
14. De Jong, K., *Genetic Algorithms: A 25 Year Perspective*, in [54], pp.125–134.
15. Dhar, V. and Ranganathan, N., *Integer Programming vs. Expert Systems: An Experimental Comparison*, Communications of ACM, Vol.33, No.3, pp.323–336, 1990.
16. Eshelman, L.J. and Schaffer, J.D., *Preventing Premature Convergence in Genetic Algorithms by Preventing Incest*, in [7], pp.115–122.
17. Fogel, D.B., *Evolving Artificial Intelligence*, Ph.D. Thesis, University of California, San Diego, 1992.
18. Fogel, D.B. (Editor), IEEE Transactions on Neural Networks, special issue on Evolutionary Computation, Vol.5, No.1, 1994.
19. Fogel, D.B., *An Introduction to Simulated Evolutionary Optimization*, IEEE Transactions on Neural Networks, special issue on Evolutionary Computation, Vol.5, No.1, 1994.
20. Fogel, D.B. and Atmar, W., *Proceedings of the First Annual Conference on Evolutionary Programming*, La Jolla, CA, 1992, Evolutionary Programming Society.
21. Fogel, D.B. and Atmar, W., *Proceedings of the Second Annual Conference on Evolutionary Programming*, La Jolla, CA, 1993, Evolutionary Programming Society.
22. Fogel, L.J., Owens, A.J., and Walsh, M.J., *Artificial Intelligence Through Simulated Evolution*, John Wiley, Chichester, UK, 1966.
23. Fogel, L.J., *Evolutionary Programming in Perspective: The Top-Down View*, in [54], pp.135–146.
24. Forrest, S. (Editor), Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1993.
25. Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
26. Goldberg, D.E., Deb, K., and Korb, B., *Do not Worry, Be Messy*, in [7], pp.24–30.
27. Goldberg, D.E., Milman, K., and Tidd, C., *Genetic Algorithms: A Bibliography*, IlliGAL Technical Report 92008, 1992.
28. Grefenstette, J.J., (Editor), Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

29. Grefenstette, J.J., (Editor), Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.
30. Heitkötter, J., (Editor), *The Hitch-Hiker's Guide to Evolutionary Computation*, FAQ in comp.ai.genetic, issue 1.10, 20 December 1993.
31. Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
32. Koza, J., *Genetic Programming*, MIT Press, 1992.
33. Männer, R. and Manderick, B. (Editors), Proceedings of the Second International Conference on Parallel Problem Solving from Nature (PPSN), North-Holland, Elsevier Science Publishers, Amsterdam, 1992.
34. Michalewicz, Z., *A Hierarchy of Evolution Programs: An Experimental Study*, Evolutionary Computation, Vol.1, No.1, 1993, pp.51–76.
35. Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 2nd edition, 1994.
36. Michalewicz, Z. (Editor), Statistics & Computing, special issue on evolutionary computation, Vol.4, No.2, 1994.
37. Michalewicz, Z., Vignaux, G.A., and Hobbs, M., *A Non-Standard Genetic Algorithm for the Nonlinear Transportation Problem*, ORSA Journal on Computing, Vol.3, No.4, 1991, pp.307–316.
38. Mühlenbein, H. and Schlierkamp-Vosen, D., *Predictive Models for the Breeder Genetic Algorithm*, Evolutionary Computation, Vol.1, No.1, pp.25–49, 1993.
39. Nissen, V., *Evolutionary Algorithms in Management Science: An Overview and List of References*, European Study Group for Evolutionary Economics, 1993.
40. Orvosh, D. and Davis, L., *Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints*, in [24], p.650.
41. Potter, M. and De Jong, K., *A Cooperative Coevolutionary Approach to Function Optimization*, George Mason University, 1994.
42. Proceedings of the First IEEE International Conference on Evolutionary Computation, Z. Michalewicz, J.D. Schaffer, H.-P. Schwefel, H. Kitano, D. Fogel (Editors), Orlando, 26 June – 2 July, 1994.
43. Reeves, C.R., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, London, 1993.
44. Saravanan, N. and Fogel, D.B., *A Bibliography of Evolutionary Computation & Applications*, Department of Mechanical Engineering, Florida Atlantic University, Technical Report No. FAU-ME-93-100, 1993.
45. Schaffer, J., (Editor), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1989.
46. Schaffer, J.D. and Morishima, A., *An Adaptive Crossover Distribution Mechanism for Genetic Algorithms*, in [29], pp.36–40.
47. Schraudolph, N. and Belew, R., *Dynamic Parameter Encoding for Genetic Algorithms*, CSE Technical Report #CS90–175, University of San Diego, La Jolla, 1990.
48. Schwefel, H.-P., *On the Evolution of Evolutionary Computation*, in [54], pp.116–124.
49. Schwefel, H.-P. and Männer, R. (Editors), Proceedings of the First International Conference on Parallel Problem Solving from Nature (PPSN), Springer-Verlag, Lecture Notes in Computer Science, Vol.496, 1991.
50. Sebald, A.V. and Fogel, L.J., *Proceedings of the Third Annual Conference on Evolutionary Programming*, San Diego, CA, 1994, World Scientific.

51. Shaefer, C.G., *The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique*, in [29], pp.50–55.
52. Vignaux, G.A., and Michalewicz, Z., *A Genetic Algorithm for the Linear Transportation Problem*, IEEE Transactions on Systems, Man, and Cybernetics, Vol.21, No.2, 1991, pp.445–452.
53. Whitley, D., *Genetic Algorithms: A Tutorial*, in [36], pp.65–85.
54. Zurada, J., Marks, R., and Robinson, C. (Editors), *Computational Intelligence: Imitating Life*, IEEE Press, 1994.

This article was processed using the LaTeX macro package with LLNCS style