Invited paper

The Spirit of Evolutionary Algorithms

Zbigniew Michalewicz[1], Susana Esquivel[2], Raul Gallard[2],
Maciej Michalewicz[3], Guo Tao[4] and Krzysztof Trojanowski[3]

[1] Department of Computer Science, University of North Carolina, Charlotte, USA, and Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
[2] Proyecto 338403, Departamento de Informatica, Facultad de Cs. Fisico-Matematicas y Naturales, Universidad Nacional de San Luis, 5700–San Luis, Argentina
[3] Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
[4] State Key Laboratory of Software Engineering, Wuhan University, Wuhan, Hubei, P.R. China

Evolutionary algorithms (EAs), which are based on a powerful principle of evolution: survival of the fittest, and which model some natural phenomena: genetic inheritance and Darwinian strife for survival, constitute an interesting category of modern heuristic search. During the last two decades there has been a growing interest in these algorithms; today, many complex software systems include at least some evolutionary component.

However, the process of building an evolutionary program is still art rather than science; often it is based on the intuition and experience of the designer. In this introductory article we present some important ideas behind the construction of evolutionary algorithms. These ideas are illustrated by three test cases: the transportation problem, a particular nonlinear parameter optimization problem, and the traveling salesman problem. We conclude the paper with a brief discussion on how an evolutionary algorithm can be tuned to the problem while solving it, which may increase further efficiency of the algorithm in a significant way.

1. Introduction

During the last two decades there has been a growing interest in algorithms which are based on the principle of evolution (survival of the fittest). A common term, accepted recently, refers to such techniques as *evolutionary computation* (EC) methods. The best known algorithms in this class include genetic algorithms, evolutionary programming, evolution strategies, and genetic programming. There are also many hybrid systems which incorporate various features of the above paradigms, and consequently are hard to classify; anyway, we refer to them just as EC methods.

The field of evolutionary computation has reached a stage of maturity. There are several, well established international conferences that attract hundreds of participants (International Conferences on Genetic Algorithms—ICGA [31, 32, 69, 10, 26, 18, 5], Parallel Problem Solving from Nature—PPSN [73, 46, 12, 78, 6], Annual Conferences on Evolutionary Programming—EP [22, 23, 76, 47, 24, 25]), and IEEE International Conferences on Evolutionary Computation [59, 60, 61, 62, 63]. Also, there are many workshops, special sessions, and local conferences every year, all around the world. Two journals, *Evolutionary Computation* (MIT Press) and *IEEE Transactions on Evolutionary Computation* are devoted entirely to evolutionary computation techniques. Many other journals have organized special issues on evolutionary computation (e.g., [19, 49, 52]). Many excellent tutorial papers [8, 9, 65, 80, 20] and technical reports provide more-or-less complete bibliographies of the field [1, 29, 68, 57]. There is also *The Hitch-Hiker's Guide to Evolutionary Computation* prepared initially by Jörg Heitkötter and currently by David Beasley [35], available (http://www.cis.ohio-state.edu/hypertext/faq/usenet/ai-faq/genetic/top.html), and a new text, *Handbook of Evolutionary Computation*, is available now [7]. Many textbooks [4, 13, 21, 28, 41, 42, 48, 54, 72] provide a detailed discussion on various aspects of evolutionary computations.

However, the process of buiding an evolutionary program is still art rather than science; often it is still based on the intuition and experience of the designer. In this introductory article we present some important ideas behind the construction of evolutionary algorithms. These ideas are illustrated by three test cases: the transportation problem, a nonlinear parameter optimization problem, and the traveling salesman problem. The systems developed for these problems provide many hints related to the design of evolutionary algorithms, in general.

The paper is organized as follows. The next section provides a short introductory description of evolutionary algorithms. Section 3 presents three interesting examples of evolutionary algorithms developed for specific problems. Section 4 concludes this article with a brief discussion on one of the most interesting developments in the field: adaptation of the algorithm to the problem, which need not be stationary.

2. Evolutionary Computation

In general, any abstract task to be accomplished can be thought of as solving a problem, which, in turn, can be perceived as a search through a space of potential solutions. Since usually we are after "the best" solution, we can view this task as an optimization process. For small spaces, classical exhaustive methods usually suffice; for larger spaces special artificial intelligence techniques must be employed. The methods of evolutionary computation are among such techniques; they are stochastic algorithms whose search methods model some natural phenomena: genetic inheritance and Darwinian strife for survival. As stated in [16]:

> "... the metaphor underlying genetic algorithms[1] is that of natural evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to a complicated and changing environment. The 'knowledge' that each species has gained is embodied in the makeup of the chromosomes of its members."

The structure of any evolutionary computation algorithm is shown in Figure 1.

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

Fig. 1. The structure of an evolutionary algorithm

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 ($c_j : S \times \ldots \times S \rightarrow S$).[2] After some number of generations the algorithm converges—it is hoped that the best individual represents a near-optimum (reasonable) solution.

Despite powerful similarities between various evolutionary computation techniques there are

[1] The best known evolutionary computation techniques are genetic algorithms; very often the terms *evolutionary computation* methods and *GA-based* methods are used interchangeably.

[2] In most cases crossover involves just two parents, however, it need not be the case. In a recent study [17] the authors investigated the merits of 'orgies', where more than two parents are involved in the reproduction process. Also, scatter search techniques [27] proposed the use of multiple parents.

also many differences between them (often hidden at a lower level of abstraction). They use different data structures S for their chromosomal representations, consequently, the 'genetic' operators are different as well. They may or may not incorporate some other information (to control the search process) in their genes. There are also other differences; for example, the two lines of the Figure 1:

select $P(t)$ from $P(t-1)$
alter $P(t)$

can appear in the reverse order: in evolution strategies first the population is altered and later a new population is formed by a selection process. Moreover, even within a particular technique there are many flavors and twists. For example, there are many methods for selecting individuals for survival and reproduction. These methods include (1) proportional selection, where the probability of selection is proportional to the individual's fitness, (2) ranking methods, where all individuals in a population are sorted from the best to the worst and probabilities of their selection are fixed for the whole evolution process,[3] and (3) tournament selection, where some number of individuals (usually two) compete for selection to the next generation: this competition (tournament) step is repeated population-size number of times. Within each of these categories there are further important details. Proportional selection may require the use of scaling windows or truncation methods, there are different ways for allocating probabilities in ranking methods (linear, nonlinear distributions), the size of a tournament plays a significant role in tournament selection methods. It is also important to decide on a generational policy. For example, it is possible to replace the whole population by a population of offspring, or it is possible to select the best individuals from two populations (population of parents and population of offspring)—this selection can be done in a deterministic or nondeterministic way. It is also possible to produce few (in particular, a single) offspring, which replace some (the worst?) individuals (systems based on such generational policy are called 'steady state'). Also, one can use an 'elitist' model which keeps the best individual from one generation to the next[4]; such model is very helpful for solving many kinds of optimization problems.

3. Three Evolutionary Algorithms

The data structure used for a particular problem together with a set of 'genetic' operators constitute the most essential components of any evolutionary algorithm. These are the key elements which allow us to distinguish between various paradigms of evolutionary methods: *genetic algorithms*, which usually operate on binary strings with crossover as a leading operator, *evolution strategies*, which process populations of floating-point vectors applying Gaussian mutation to all components of the vector, *evolutionary programming* technique, which was proposed for breeding finite state machines (5 mutation operators are involved), or *genetic programming*, which processes programs (again, with crossover performed on program-trees as a major operator here). In this paper, however, we discuss three evolutionary algorithms which are difficult to classify: some problem-specific knowledge was incorporated in all of them and, consequently, they resist the above classification. These three systems were developed for (1) the transportation problem, (2) a particular nonlinear parameter optimization problem, and (3) the traveling salesman problem, and are discussed in the following three subsections, respectively.

3.1. The Transportation Problem

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

[3] For example, the probability of selection of the best individual is always 0.15 regardless its precise evaluation; the probability of selection of the second best individual is always 0.14, etc. The only requirements are that better individuals have larger probabilities and the total of these probabilities equals to one.

[4] It means, that if the best individual from a current generation is lost due to selection or genetic operators, the system forces it into the next generation anyway.

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

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

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,

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

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 [16]:

> "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.[5] Also, there is a possibility of using a data structure appropriate for the problem at hand together with the set of specialized operators. 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 $m = (i-1)\ mod\ k + 1$.

However, it is difficult to design a meaningful set of penalty functions. If penalty functions are moderate, the system often returns [51] 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 about 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 of original variables x_{ij}.

It seems that the penalty function approach is not the most suitable one 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:

$$\begin{matrix}
\cdots & . & \cdots & . & \cdots \\
\cdots & . & \cdots & . & \cdots \\
\cdots & v_i & \cdots & v_j & \cdots \\
\cdots & . & \cdots & . & \cdots \\
\cdots & . & \cdots & . & \cdots \\
\cdots & v_k & \cdots & v_m & \cdots \\
\cdots & . & \cdots & . & \cdots \\
\cdots & . & \cdots & . & \cdots
\end{matrix}$$

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.

[5] Orvosh and Davis [58] 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 them with any other rate. In particular, it is better than with 'never replacing' or 'always replacing' strategies. However, the rule has some exceptions [48].

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 one 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 [48]), and it might be difficult to design a decoder scheme for continuous case. Recently, a general method for nonlinear parameter optimization problems was developed [43], but it can be applied to inequalities only.

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 that it contains only feasible individuals. In [53] 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 an individual 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 [53]) 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 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 [53, 48, 51]. It is worthwhile to underline, that the results were much better than these obtained from the GAMS (General Algebraic Modeling System) with MINOS optimizer.

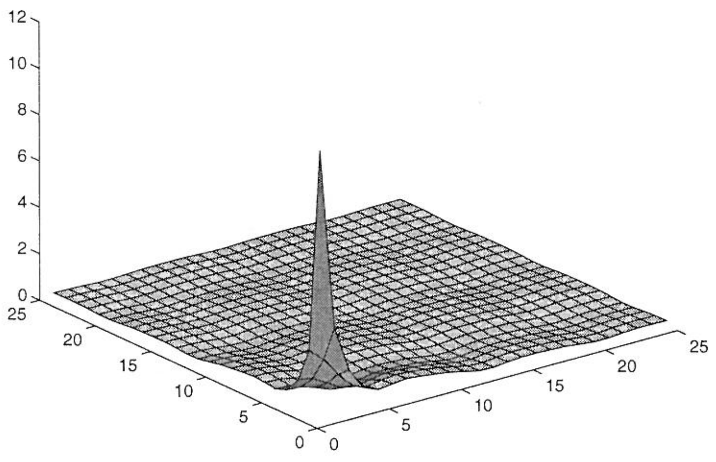

Fig. 2. The graph of function f for $n = 2$

3.2. A Nonlinear Parameter Optimization Problem

An interesting constrained numerical optimization test case emerged recently; the problem [39] is to maximize a function:

$$f(\vec{x}) = \left|\frac{\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} ix_i^2}}\right|,$$

where

$$\prod_{i=1}^{n} x_i \geq 0.75, \quad (1)$$
$$\sum_{i=1}^{n} x_i \leq 7.5n, \quad (2)$$
and $0 \leq x_i \leq 10$ for $1 \leq i \leq n$.

The problem has two constraints; the function f is nonlinear and its global maximum is unknown.

To illustrate some potential difficulties of solving this test case, a few graphs (for case of $n = 2$) are displayed in Figures 2 – 4. Figure 2 gives a general overview of the objective function f: the whole landscape seems to be relatively flat except for a sharp peek around the point $(0, 0)$. Figures 3 and 4 incorporate the active constraint: infeasible solutions were assigned a value of zero. In that case the objective function f takes values from the range $\langle 0, 1\rangle$; because of the scaling, the landscape

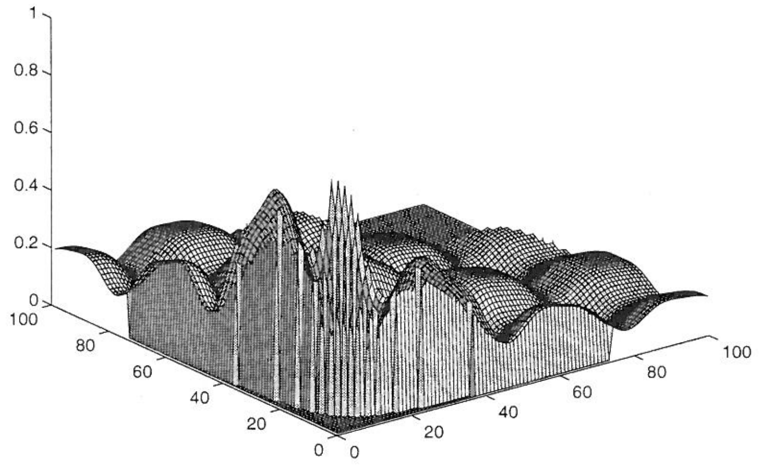

Fig. 3. The graph of function f for $n = 2$. Infeasible solutions were assigned value zero

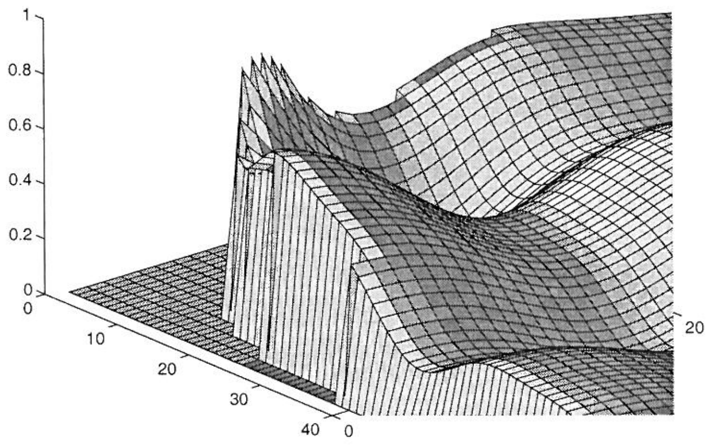

Fig. 4. The graph of function f for $n = 2$. Infeasible solutions were assigned value zero; only the corner around the global optimum is shown

is more visible. Figure 4 displays only the area of interest (i.e., the area of global optimum).

Many constraint-handling methods [51] were tried on this test case with quite poor results. As Keane [39] noted:

> "I am currently using a parallel GA with 12bit binary encoding, crossover, inversion, mutation, niche forming and a modified Fiacco-McCormick constraint penalty function to tackle this. For $n = 20$ I get values like 0.76 after 20,000 evaluations."

It seems that majority of constraint-handling methods (for a survey, see [51]) have serious difficulties in returning a high quality solution for the above problem. It might be possible, however, to build a dedicated system for this particular test case, which would incorporate the problem specific knowledge. This knowledge can emerge from analysis of the objective function f and the constraints; thus we assume that (a) the domain for all variables is $\langle 0.1, 10.0\rangle$, (b) constraint (1) is active at the global optimum, and (c) the constraint (2) is not.

Now it is possible to develop an evolutionary system which would search just the surface defined by the first constraint:

$$\prod_{i=1}^{n} x_i = 0.75.$$

Thus the search space is greatly reduced; we consider only points which satisfy the above equation. Such a system would start from a population of feasible points, i.e., the points for which a product of all coordinates is equal to 0.75. It is relatively easy to develop such initialization routine:

```
for $i = 1$ to $n$ do
begin
    if $i$ is even
        $x_{i-1}$ = random $(0.1, 10)$
        $x_i = 1/x_{i-1}$
end
if $n$ is odd
    $x_n = 0.75$
else
    for some random $i$
    $x_i = 0.75 \cdot x_i$
```

It is important to note, that the above initialization routine is not significant for the performance of any system; it just ensures that all points in the population are at the boundary between feasible and infeasible regions. Many other methods were tried with such initialized populations and gave poor results. Also, the value of the best individual in the population initialized in such a way is around 0.30.

The geometrical crossover takes two parents and produces a single offspring; for parents $\vec{x}^1$ and $\vec{x}^2$ the offspring $\vec{x}^3$ is

$$\vec{x}^3 = \langle \sqrt{x_1^1 \cdot x_1^2}, \ldots, \sqrt{x_n^1 \cdot x_n^2}\rangle. \quad (3)$$

Note, that the offspring $\vec{x}^3$ also lies on the boundary of feasible region:

$$\prod_{i=1}^{n} x_i^3 = 0.75.$$

Of course, it is an easy task to generalize the above geometrical crossover into

$$\vec{x}^3 = \langle (x_1^1)^\alpha \cdot (x_1^2)^{(1-\alpha)}, \ldots, (x_n^1)^\alpha \cdot (x_n^2)^{(1-\alpha)} \rangle, \qquad (4)$$

for $0 \le \alpha \le 1$. Also it is possible to include several (say, k) parents:

$$\vec{x}^{k+1} = \langle (x_1^1)^{\alpha_1} \cdot (x_1^2)^{\alpha_2} \cdot \ldots \cdot (x_1^k)^{\alpha_k}, \ldots, (x_n^1)^{\alpha_1} \cdot (x_n^2)^{\alpha_2} \cdot \ldots \cdot (x_n^k)^{\alpha_k} \rangle, \quad (5)$$

where $\alpha_1 + \ldots + \alpha_k = 1$. However, in the experiments reported in this paper, we limited ourselves to the simplest geometrical crossover (3).

The task of designing a feasibility-preserving mutation is relatively simple; if the i-th component is selected for mutation, the following algorithm is executed:

```
determine random $j$, $1 \le j \le n$, $j \ne i$
select $q$ such that:
    $0.1 \le x_i \cdot q \le 10.0$ and
    $0.1 \le x_j / q \le 10.0$
$x_i = x_i \cdot q$
$x_j = x_j / q$
```

A simple evolutionary algorithm (200 lines of code!) with geometrical crossover and problem-specific mutation gave an outstanding result. For the case $n = 20$ the system reached the value of 0.80 in less than 4,000 generations (with population size of 30, probability of crossover $p_c = 1.0$, and probability of mutation $p_m = 0.06$) in all runs. The best value found (namely 0.803553) was better than the best values of any method discussed earlier, whereas the worst value found was 0.802964. Similarly, for $n = 50$, all results (in 30,000 generations) were better than 0.83 (with the best of 0.8331937). All results of experiments were reported in [50].

It was interesting to note the importance of geometrical crossover. With fixed population size (kept constant at 30), the higher values of probability of crossover p_c, the better results of the system were observed. Similarly, the best mutation rates were relatively low ($p_m \approx 0.06$). In Figure 5 we illustrate the average values (out of 10 runs) of the best values found for different probabilities of mutation (with fixed $p_c = 1.0$).

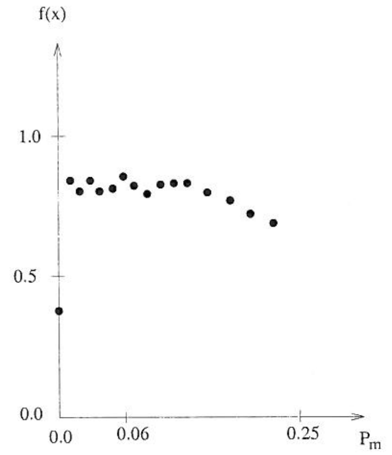

Fig. 5. The performance of the system with $p_c = 1.0$ and a variable mutation rate p_m

Clearly, geometrical crossover can be applied only for problems where each variable takes nonnegative values only, so its use is quite restricted in comparison with other types of crossovers (e.g., arithmetical crossover). However, for many engineering problems, all problem variables are positive; moreover, it is always possible to replace variable $x_i \in \langle a_i, b_i \rangle$ which can take negative values (i.e., where $a_i < 0$) with a new variable $y_i = x_i - a_i$.

In general, it is a common situation for many constrained optimization problems that some constraints are active at the target global optimum. Thus the optimum lies on the boundary of the feasible space. On the other hand, it is commonly acknowledged that restricting the size of the search space in evolutionary algorithms (as in most other search algorithms) is generally beneficial. Hence, it seems natural in the context of constrained optimization to restrict the search of the solution to some part of the boundary of the feasible part of the space, i.e., the part of R^n where some of the inequality constraints actually are equalities.

In the case of linear programming (where both the constraints and the objective function are linear), theoretical results ensure that the solution is one of the summits of the surface: the well known simplex method thus only searches the set of these summits. However, in the nonlinear case with multiple constraints, there is no way to tell which one are active at the global optimum. This may cause some difficulties. In [70, 71] the authors explored a possibility of developing evolutionary systems for searching the boundary between feasible and infeasible parts of the search space; the results of first experiments can be found in these papers.

3.3. The Euclidean Traveling Salesman Problem

The traveling salesman problem (TSP) is one of the most widely studied NP-hard combinatorial optimization problems. Its statement is deceptively simple, and yet it remains one of the most challenging problems in Operational Research.

Let $G = (V, E)$ be a graph where V is a set of vertices and E is a set of edges. Let $C = (c_{ij})$ be a distance (or cost) matrix associated with E. The TSP requires determination of a minimum distance circuit (Hamiltonian circuit or cycle) passing through each vertex once and only once. C is said to satisfy the triangle inequality if and only if $c_{ij} + c_{jk} \geq c_{ik}$ for all $i, j, k \in V$ (in such a case we talk about $\triangle$TSP). Euclidean TSP problems (ETSP), i.e., problems where V is a set of points in R^2 and c_{ij} is an Euclidean (straight-line) distance between i and j, are, of course, special cases of $\triangle$TSP.

A lot of algorithms have been proposed to solve TSP. Some of them (based on dynamic programming or branch and bound methods) provide the global optimum solution (the largest nontrivial instance of the TSP solved to optimality is of 7397 cities [3], however, it required almost 4 years of CPU time on network of machines). Other algorithms are heuristic ones, which are much faster, but they do not guarantee the optimal solutions. There are well known algorithms based on 2-opt or 3-opt change operators, Lin-Kerninghan algorithm (variable change) as well algorithms based on greedy principles (nearest neighbor, spanning tree, etc). The TSP was also approached by various "modern heuristic" methods, like simulated annealing, evolutionary algorithms, tabu search, even neural networks. However, these techniques were mainly applied to test cases with relatively small number of cities (usually less than 1000), whereas such problems are now solved routinely within a few hours [37].

The evolutionary algorithm based on a special operator *inver-over*,[6] which incorporates the knowledge taken from other individuals in the population. One can view this operator as a mixture of inversion and recombination: on one hand, the inversion is applied to a part of a single individual, however, the selection of a segment to be inverted depends on other individuals in the population.

It seems that the proposed algorithm still can't compete (at least as far as computational time is concern) with efficient approaches based on local search [37], however, it has a few advanteges. First of all, it is extremely simple and easy to implement (less than 100 lines of C code). Additionally, experimental results indicate that this operator outperforms all other evolutionary operators (whether unary or binary), which have been proposed in the past for the TSP (PMX, OX, CX, ER, EER, simple inversion, etc). Moreover, the evolutionary algorithm based on the proposed operator is quite fast (in comparison with other evolutionary techniques) and the quality of results are very high. For test cases, where the number of cities is around 100, the algorithm reaches the optimum in every execution. For larger instances (10,000 cities) the results stay within 3% from the estimated optimum.

The algorithm developed for the ETSP has the following characteristics:

- there is a population P of m individuals, $P = \{S_1, S_2, ..., S_m\}$,
- each individual competes with its offspring only,
- there is only one inversion operator used; however, this inversion is not random, but adaptive: it takes a clue from the current population,
- the number of times the operator is applied to an individual during a single generation, is variable.

[6] The name for this operator was invented by Bob Reynolds during the EP'98 conference.

```
random initialization of the population $P$
while (not satisfied termination-condition) do
{
  for each individual $S_i \in P$ do
  {
    $S' = S_i$
    select (randomly) a city $c$ from $S'$
    repeat
    {
      if ($rand() \leq p$)
        select the city $c'$ from the remaining cities in $S'$
      else
      {
        select (randomly) an individual from $P$
        assign to $c'$ the 'next' city to the city $c$ in the selected individual
      }
      if (the next city or the previous city of city $c$ in $S'$ is $c'$)
        exit from repeat loop
      inverse the section from the next city of city $c$ to the city $c'$ in $S'$
      $c = c'$
    }
    if ($eval(S') \leq eval(S_i)$)
      $S_i = S'$
  }
}
```

Fig. 6. The outline of the algorithm for the ETSP

Such an algorithm can be perceived as a set of m parallel hill-climbing procedures, which preserve the spirit of Lin-Kerninghan algorithm (each hill-climber performs a variable number of edge-swaps). However, the operator used here has adaptive components: (1) the number of inversions and (2) the segment to be inverted, depend on the current population, i.e., on the current state of the search. So it is possible to view this algorithm as an evolutionary one with a strong selective pressure, and with an adaptive operator, which is an interesting combination of a simple inversion and crossover (as the second city is selected on the basis of another individual from the population).

Figure 6 provides a more detailed description of the whole algorithm in general and of the proposed operator in particular. With a low probability p^7 the second city for inversion is selected randomly. This is necessary: without a possibility to generate new connections, the algorithm would search only among connections between cities present in the initial population. If $rand() > p$, a randomly selected mate provides a clue for the second marker for inversion. In that case the inversion operator resembles crossover, as part of the pattern (at least 2 cities) of the second individual appears in the offspring.

Let's illustrate a single iteration of this operator on the following example. Assume that the current individual S' is

$$S' = (2, 3, 9, 4, 1, 5, 8, 6, 7),$$

($m = 9$) and the current city c is 3. If the generated random number $rand()$ does not exceed p, another city c' from the same individual S' is selected (say, c' is 8), and appropriate segment is inverted, producing the following offspring

$$S' \leftarrow (2, 3, 8, 5, 1, 4, 9, 6, 7)$$

(note the position of the cutting points for the selected segment, which are after cities 3 and 8). Otherwise (i.e., $rand() > p$), another individual is (randomly) selected from the population;

[7] Interestingly, experimental results indicated that the value of this parameter was independent of the number of cities in a test case. Note also, that the function $rand()$ generates a random float from the range [0..1].

assume, it is $(1, 6, 4, 3, 5, 7, 9, 2, 8)$. This individual is searched for the city c' "next" to city 3 (which is 5), thus the segment for inversion in S' starts after city 3 and terminates after city 5; consequently, the new offspring is

$$S' \leftarrow (2, 3, 5, 1, 4, 9, 8, 6, 7).$$

Note again, that a substring $3 - 5$ arrived from the "second parent". Note also, that in either case the resulting string is intermediate in the sense that the above inversion operator is applied several times before an offspring is evaluated.

The experimental results of the algorithm are presented in [77]. Almost all test cases were chosen from TSPLIB [66]. The optimal solution of each test case was known. The size of these test cases varied from 30 cities to 2,392 cities. We have also created one (random) instance (RAN10000) of 10,000 cities, and relied on the formula (derived empirically in [38]) for the expected ratio k of the Held-Karp bound to $\sqrt{n}$ for n-city random ETSP; for $n \geq 100$ it is:

$$k = 0.70805 + \frac{0.52229}{\sqrt{n}} + \frac{1.31572}{n} - \frac{3.07474}{n\sqrt{n}}.$$

So, the length of the optimal tour is estimated as $L^* = k\sqrt{n \cdot R}$, where n is the number of cities and R is the area of the square box within which the cities were randomly placed. For our instance, the number of cities is $n = 10,000$ and the edge length (of the square box) is 400, so the approximate length of the optimum solution is 28536.3.

The reported results demonstrated clearly the efficiency of the algorithm. For the first nine test cases the optimum was found in all runs (except the test case EIL101, where the algorithm failed only once in ten runs). The number of cities in these test cases varies from 30 to 105. For the test case with 144 cities the average solution was only 0.04% above the optimum, for the test case with 442 cities—0.63% above the optimum, and for the test case with 2392 cities—2.66%. Moreover, for a random test case with 10,000 cities the average solution stayed within 3.56% from the Held-Karp lower bound (whereas the best solution found in these ten runs was less than 3% above this lower bound). The running time of the algorithm was reasonable: few seconds for problems with up to 105 cities, below 3 minutes for the test case of 442 cities, below 90 minutes for the test case with 2392 cities. These represent fraction of time needed by other evolutionary algorithms based on crossover operators.

In [77] the above algorithm was compared to two other algorithms. The first one was based on simple inversion and the second one was based on Lin-Kerninghan algorithm.[8] The comparison with the first algorithm provided information on the significance of the proposed adaptive inversion operator versus blind inversion, whereas the other one—on relative merits of the tested algorithms. The first algorithm, for the test cases with around 100 cities, the error (percentage above the optimum) was much higher (more than 10%). Time of the run increased significantly, in some cases more than 100 times (the termination condition was left without a change). On the other hand, the Lin-Kerninghan algorithm takes a fraction of time necessary for our algorithm (e.g., below 1 second for the test case with 2,392 cities). However, the precision of results is much lower: none of the test cases resulted with the optimum solution in all ten runs. So, the proposed evolutionary algorithm has much better consistency than the Lin-Kerninghan algorithm. On the other hand, if Lin-Kerninghan algorithm was run for the same *time* as our evolutionary system (as opposed just to the same number of runs), it would win the competition easily.

There are a few interesting observations which can be made on the basis of the experiments:

- the proposed system is probably the quickest evolutionary algorithm for the TSP developed so far. All other algorithms based on crossover operator (whether PMX, OX, CX, ER, etc) provide much worse results in a much longer time;
- the proposed system has only three parameters: population size m, the probability p of generating random inversion, and the number of iterations in the termination condition; most of the other evolutionary systems have many additional parameters;

[8] We have experimented with the implementation provided by Bill Cook, available from ftp.caam.rice.edu/pub/people/bico/970827/.

- it is worthwhile to emphasize the precision and stability of the system for relatively small test cases (almost 100% accuracy for all considered test cases up to 105 cities); the computational time was also acceptable (3-4 seconds);
- the system introduces a new, interesting operator which combines features of inversion (or mutation) and crossover. Results of experiments reported in the previous section clearly indicate clearly that the proposed operator is significantly better than random inversion. The probability parameter p (in all experiments kept constant at 0.02) determines a proportion of blind inversions and guided (adaptive) inversions. The latter is much higher (0.02 versus 0.98 in reported results).

4. Discussion

The effectiveness of evolutionary computations depends on the representation used for the problem solutions, the reproduction operators used and the configuration of the evolutionary algorithm. Additionally, problem-specific knowledge should be incorporated into evolutionary systems. These ideas are not new and have been recognized for some time. Several researchers have discussed initialization techniques, different representations, and the use of heuristics for genetic operators. In [14] Davis wrote:

> "It has seemed true to me for some time that we cannot handle most real-world problems with binary representations and an operator set consisting only of binary crossover and binary mutation. One reason for this is that nearly every real-world domain has associated domain knowledge that is of use when one is considering a transformation of a solution in the domain [...] I believe that genetic algorithms are the appropriate algorithms to use in a great many real-world applications. I also believe that one should incorporate real-world knowledge in one's algorithm by adding it to one's decoder or by expanding one's operator set."

The main conclusions one can reach from the experiments described in the previous section are as follows:

- Coding of chromosome structures S should match the problem (in particular, it need not be binary). Note, that for the three test cases we have used a matrix representation, floating-point vector, and the permutation of integer numbers, respectively.
- The 'genetic' operators need not be 'genetic' and should incorporate the problem-specific knowledge. Note, that for the first test case, a special crossover and two special mutations operators were developed. The second test case required specialized operators which search the boundary between feasible and infeasible parts of the search space. For the third test case, a special operator (inver-over) was developed, which combined efficiency of unary operator with a flavor of recombination.
- The problem-specific knowledge incorporated into the system enhances an algorithm's performance and narrows its applicability. Clearly, the system developed for the transportation problem cannot be used for a general nonlinear programming problem nor for the traveling salesman problem. The same is true for the other two systems.
- Because of the special representation, operators (i.e., specialization of these systems), it is difficult to classify them into historical categories (i.e., genetic algorithms, evolution strategies, evolutionary programming, or genetic programming). For example, the system developed for the transportation problem uses matrix representation (used in early evolutionary programming to represent finite state machines) with floating point numbers (as in evolution strategies), and incorporates crossover (as in genetic algorithms). Probably the best thing to do is to label such a system just as an *evolutionary algorithm* (EA).

In all presented evolutionary systems, all parameters were fixed: population size, probabilities of operators, etc. However, it seems that we can do better than that. As evolutionary algorithms implement the idea of evolution, and as evolution itself must have evolved to reach its current state of sophistication, it is natural to expect adaption to be used not only for finding solutions to a problem, but also for tuning the algorithm to the particular problem.

Adaptation gives us the opportunity to customize the evolutionary algorithm to the problem and to modify the configuration and the strategy parameters used while the problem solution is sought. This enables us not only to incorporate domain information and multiple reproduction operators into the EA more easily, but can allow the algorithm itself to select those values and operators which give better results. Also, these values can be modified during the run of the EA to suit the situation during that part of the run.

In EAs, not only do we need to choose the algorithm, representation and operators for the problem, but we also need to choose parameter values and operator probabilities for the evolutionary algorithm so that it will find the solution and, what is also important, find it efficiently. This is a time consuming task and a lot of effort has gone into automating this process. Researchers have used various ways of finding good values for the strategy parameters as these can affect performance of the algorithm significantly. Many researchers experimented with problems from a particular domain, tuning the strategy parameters on the basis of such experimentation (tuning "by hand"). Later, they reported their results of applying a particular EA to a particular problem, stating:

> For these experiments, we have used the following parameters: population size = 80, probability of crossover = 0.7, etc.

without much justification of the choice made. Other researchers tried to modify the values of strategy parameters during the run of the algorithm; it is possible to do this by using some (possibly heuristic) rule, by taking feedback from the current state of the search, or by employing some self-adaptive mechanism. Note that these changes may effect a single component of a chromosome, the whole chromosome (individual), or even the whole population. Clearly, by changing these values while the algorithm is searching for the solution of the problem, further efficiencies can be gained. The action of determining the variables and parameters of an EA to suit the problem has been termed *adapting the algorithm* to the problem, and in EAs this can be done while the algorithm is finding the problem solution [36].

Note also that the real-world problems often present another set of difficulties: they change. They change before they are modelled, they change while solutions are being derived, and they change after the execution of the best solution. Hence a real-time optimization algorithm needs to consider changes which may occur during the derivation or execution of the solution. In such cases (so-called non-stationary function optimization) the goal is not to locate the extrema but rather to track their progression through the space as closely as possible. A few researchers have extended Evolutionary Algorithms to handle non-stationary optimization; these extensions include:

- *Maintenance of the diversity level.* The presence of many potential solutions during the evolutionary search seems to be a useful feature of optimization in changing environments. As long as some level of the population diversity is upheld we could expect the algorithm to adapt to changes more easily. Hence maintaining diversity of the population could increase search performance of the algorithm [33, 56, 81].

- *Adaptation and self-adaptation mechanism.* Dynamical adjustment of the algorithm to the nonstationary environment is the next feature of the efficient optimisation. So adaptive and self-adaptive techniques are the next significant extension of evolutionary algorithm [2, 36]. We discussed this aspect earlier in this section of the paper.

- *Redundancy of genetic material.* One of the most important abilities in adaptation to changes is reasoning from previous experiences. So the next idea to handle non-stationary environments is based on addition of some memory structures to the chromosomes. We can classify these structures into several types [74]:

- *numerical memory* — where the modification of algorithm parameters is performed using experience of previous generations; e.g., [67, 74, 79];

- *symbolic memory* — where the algorithm gradually learns from the individuals in the populations and thus constructs beliefs about the relevance of schemes (Machine Learning theory is exploited) [75];

- *exact memory* — where existing structures are enhanced by additional genes, chomosomes (diploidy) or groups of chromosomes (polyploidy) [11, 30, 34, 44, 45, 55, 64, 82].

All these issues, together with parallel implementations of evolutionary algorithms, constitute the most interesting directions for further research.

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] ANGELINE, P., "Tracking Extrema in Dynamic Environments", in [25], pp.335–346.

[3] APPLEGATE, D. BIXBY, R.E., CHVATAL, V., AND COOK, W., *Finding cuts in the TSP: a preliminary report*, Report 95-05, DIMACS, Rutgers University, NJ.

[4] BÄCK, T., *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, 1995.

[5] BÄCK, T. (Editor), Proceedings of the 7th International Conference on Genetic Algorithms, Morgam Kaufmann, San Mateo, CA, 1997.

[6] Proceedings of the 5th Parallel Problem Solving from Nature, T. BÄCK, A.E. EIBEN, M. SCHOENAUER, AND H.-P. SCHWEFEL (Editors), Amsterdam, September 27–30, 1998.

[7] BÄCK, T., FOGEL, D., AND MICHALEWICZ, Z. (Editors), *Handbook of Evolutionary Computation*, Oxford University Press, New York, 1996.

[8] 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.

[9] 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.

[10] BELEW, R. AND BOOKER, L. (Editors), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[11] DASGUPTA, D., MCGREGOR, D. R., "Nonstationary Function Optimisation using the Structured Genetic Algorithm", in [46], pp.145–154.

[12] DAVIDOR, Y., SCHWEFEL, H.-P., AND MÄNNER, R. (Editors), Proceedings of the Third International Conference on Parallel Problem Solving from Nature (PPSN), Springer-Verlag, New York, 1994.

[13] DAVIS, L., *Handbook of Genetic Algorithms*, New York, Van Nostrand Reinhold, 1991.

[14] DAVIS, L., *Adapting Operator Probabilities in Genetic Algorithms*, in [69], pp.61–69.

[15] DAVIS, L. (Editor), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA, 1987.

[16] DAVIS, L. AND STEENSTRUP, M., "Genetic Algorithms and Simulated Annealing: An Overview", in [15], pp.1–11.

[17] EIBEN, A.E., RAUE, P.-E., AND RUTTKAY, ZS., "Genetic Algorithms with Multi-parent Recombination", in [12], pp.78–87.

[18] ESHELMAN, L.J., (Editor), Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1995.

[19] FOGEL, D.B. (Editor), IEEE Transactions on Neural Networks, special issue on Evolutionary Computation, Vol.5, No.1, 1994.

[20] FOGEL, D.B., "An Introduction to Simulated Evolutionary Optimization", IEEE Transactions on Neural Networks, special issue on Evolutionary Computation, Vol.5, No.1, 1994.

[21] FOGEL, D.B., *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, NJ, 1995.

[22] FOGEL, D.B. AND ATMAR, W., Proceedings of the First Annual Conference on Evolutionary Programming, La Jolla, CA, 1992, Evolutionary Programming Society.

[23] FOGEL, D.B. AND ATMAR, W., Proceedings of the Second Annual Conference on Evolutionary Programming, La Jolla, CA, 1993, Evolutionary Programming Society.

[24] FOGEL, L.J., ANGELINE, P.J., BÄCK, T. (Editors), Proceedings of the Fifth Annual Conference on Evolutionary Programming, The MIT Press, 1996.

[25] Proceedings of the Sixth International Conference on Evolutionary Programming - EP'97, vol. 1213 in LNCS, Springer, 1997.

[26] FORREST, S. (Editor), Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1993.

[27] GLOVER, F., "Heuristics for Integer Programming Using Surrogate Constraints", Decision Sciences, Vol.8, No.1, pp.156–166, 1977.

[28] GOLDBERG, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[29] GOLDBERG, D.E., MILMAN, K., AND TIDD, C., *Genetic Algorithms: A Bibliography*, IlliGAL Technical Report 92008, 1992.

[30] GOLDBERG, D.E., SMITH, R.E., "Nonstationary Function Optimisation Using Genetic Algorithms with Dominance and Diploidy", in [32], pp.59–68.

[31] GREFENSTETTE, J.J., (Editor), Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[32] GREFENSTETTE, J.J., (Editor), Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[33] GREFENSTETTE, J.J., "Genetic algorithms for changing environments", in [46], pp.137–144.

[34] HADAD, B., S., EICK, C., F., "Supporting Polyploidy in Genetic Algorithms Using Dominance Vectors", in [25], pp.223–234.

[35] HEITKÖTTER, J., (Editor), *The Hitch-Hiker's Guide to Evolutionary Computation*, FAQ in comp.ai.genetic, issue 1.10, 20 December 1993.

[36] EIBEN, A.E., HINTERDING, R., AND MICHALEWICZ, Z., *Parameter Control in Evolutionary Algorithms*, Technical Report TR98-07, Department of Computer Science, Leiden University, Netherlands, 1998.

[37] JOHNSON, D.S., "The Traveling Salesman Problem: A Case Study", in *Local Search in Combinatorial Optimization*, E. Aarts and J.K. Lenstra (Editors), John Wiley, 1996, pp.215–310.

[38] JOHNSON, D.S., MCGEOCH, L.A., AND ROTHBERG, E.E., "Asymptotic experimental analysis for the Held-Karp traveling salesman bound", Proceedings of the Seventh Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, and SIAM, Philadelphia, PA, pp.341–350.

[39] KEANE, A.J., "A Brief Comparison of Some Evolutionary Optimization Methods", in V. Rayward-Smith, I. Osman, C. Reeves and G. D. Smith (Editors), *Modern Heuristic Search Methods*, J. Wiley, pp.255–272.

[40] KINNEAR, K.E. (Editor), *Advances in Genetic Programming*, MIT Press, Cambridge, MA, 1994.

[41] KOZA, J.R., *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[42] KOZA, J.R., *Genetic Programming – 2*, MIT Press, Cambridge, MA, 1994.

[43] KOZIEŁ, S. AND MICHALEWICZ, Z., "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization", Evolutionary Computation, Vol. 7, No. 1, pp. 19–44, 1999.

[44] KWASNICKA H., "Redundancy of Genotypes as the Way for Some Advanced Operators in Evolutionary Algorithms - Simulation Study", *VIVEK A Quarterly in Artificial Intelligence*, Vol. 10, No. 3, July 1997, National Centre for Software Technology, Mumbai, pp.2–11.

[45] LOUIS, S.J. AND JOHNSON, J., "Solving Similar Problems using Genetic Algorithms and Case–Based Memory", in [5], pp.283–290.

[46] 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.

[47] MCDONNELL, J.R., REYNOLDS, R.G., AND FOGEL, D.B. (Editors), Proceedings of the Fourth Annual Conference on Evolutionary Programming, The MIT Press, 1995.

[48] MICHALEWICZ, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 3rd edition, 1996.

[49] MICHALEWICZ, Z. (Editor), Statistics & Computing, special issue on evolutionary computation, Vol.4, No.2, 1994.

[50] MICHALEWICZ, Z., NAZHIYATH, G. AND MICHALEWICZ, M., "A note on usefulness of geometrical crossover for numerical optimization problems", in [24], pp.305–312.

[51] MICHALEWICZ, Z. AND SCHOENAUER, M., "Evolutionary Algorithms for Constrained Parameter Optimization Problems", Evolutionary Computation, Vol.4, No.1, 1996, pp.1–32.

[52] MICHALEWICZ, Z. AND SCHOENAUER, M., (Editors), Control & Cybernetics, special issue on evolutionary computation, Vol.26, No.3, 1997.

[53] 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.

[54] MITCHEL, M., *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[55] MORI, N., IMANISHI, S., KITA, H., NISHIKAWA, Y., "Adaptation to Changing Environments by Means of the Memory Based Thermodynamical Genetic Algorithm", in [5], pp.299–306.

[56] MORI, N., KITA, H., NISHIKAWA, Y., "Adaptation to Changing Environments by Means of the Memory Based Thermodynamical Genetic Algorithm", in [12], pp.513–522.

[57] NISSEN, V., *Evolutionary Algorithms in Management Science: An Overview and List of References*, European Study Group for Evolutionary Economics, 1993.

[58] ORVOSH, D. AND DAVIS, L., "Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints", in [26], p.650.

[59] Proceedings of the First IEEE International Conference on Evolutionary Computation, Orlando, 26 June – 2 July, 1994.

[60] Proceedings of the Second IEEE International Conference on Evolutionary Computation, Perth, 29 November – 1 December, 1995.

[61] Proceedings of the Third IEEE International Conference on Evolutionary Computation, Nagoya, 18–22 May, 1996.

[62] Proceedings of the Forth IEEE International Conference on Evolutionary Computation, Indianapolis, 13–16 April, 1997.

[63] Proceedings of the Fifth IEEE International Conference on Evolutionary Computation, Anchorage, 5–9 May, 1998.

[64] PUPPALA, N., SEN, S., GORDIN, M., "Shared memory based Cooperative Coevolution", in [63], pp.570–574.

[65] REEVES, C.R., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, London, 1993.

[66] REINELT, G., "TSPLIB – A Traveling Salesman Problem Library", ORSA Journal on Computing, Vol.3, No.4, pp.376–384, 1991.

[67] REYNOLDS R., G., CHUNG C., J., "Knowledge–based Self–adaptation in Evolutionary Programming using Cultural Algorithms", in [62], pp.71–76.

[68] 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.

[69] SCHAFFER, J., (Editor), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1989.

[70] SCHOENAUER, M. AND MICHALEWICZ, Z., "Evolutionary Computation at the Edge of Feasibility", in [78], pp.245–254.

[71] SCHOENAUER, M. AND MICHALEWICZ, Z., "Boundary Operators for Constrained Parameter Optimization Problems", in [5], pp.322–329.

[72] SCHWEFEL, H.-P., *Evolution and Optimum Seeking*, John Wiley, Chichester, UK, 1995.

[73] 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.

[74] SEBAG, M., SCHOENAUER, M., RAVISE, C., "Toward Civilized Evolution: Developing Inhibitions", in [5], pp.291–298.

[75] SEBAG, M., SCHOENAUER, M., RAVISE, C., "Inductive Learning of Mutation Step-Size in Evolutionary Parameter Optimisation", in [25], pp 247–261.

[76] SEBALD, A.V. AND FOGEL, L.J., Proceedings of the Third Annual Conference on Evolutionary Programming, San Diego, CA, 1994, World Scientific.

[77] TAO, G. AND MICHALEWICZ, Z., "Inver-over operator for the ETSP", in [6], pp. 803–812.

[78] VOIGT, H.-M., EBELING, W., RECHENBERG, I., SCHWEFEL, H.-P. (Editors), Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature (PPSN), Springer-Verlag, New York, 1996.

[79] WHITE, T., OPPACHER, F., "Adaptive Crossover Using Automata", in [12], pp.229–238.

[80] WHITLEY, D., "Genetic Algorithms: A Tutorial", in [49], pp.65–85.

[81] VAVAK F., FOGARTY T.C., JUKES K., "Learning the Local Search Range for Genetic Optimisation in Nonstationary Environments", in [62], pp.355–360. .

[82] YOSHIDA, Y., ADACHI, N., "A Diploid Genetic Algorithm for Preserving Population Diversity – pseudo-Meiosis GA", in [12], pp.482–491.

Received: June, 1998
Accepted: December, 1998

Contact address:

Zbigniew Michalewicz
Department of Computer Science
University of North Carolina
Charlotte, NC 28223
USA
e-mail: zbyszek@uncc.edu

Susana Esquivel
Proyecto 338403
Departamento de Informatica
Facultad de Cs. Fisico-Matematicas y Naturales
Universidad Nacional de San Luis
Ejercito de los Andes 950, local 106
5700–San Luis
Argentina
e-mail: esquivel@unsl.edu.ar

Raul Gallard
Proyecto 338403, Departamento de Informatica, Facultad de Cs.
Fisico-Matematicas y Naturales, Universidad Nacional de San Luis,
Ejercito de los Andes 950, local 106, 5700–San Luis
Argentina
e-mail: rgallard@unsl.edu.ar

Maciej Michalewicz
Institute of Computer Science
Polish Academy of Sciences
ul. Ordona 21
01-237 Warsaw
Poland
e-mail: michalew@ipipan.waw.pl

Guo Tao
State Key Laboratory of Software Engineering
Wuhan University
Wuhan
Hubei, 430072
P.R. China
e-mail: gt@rjgc.whu.edu.cn

Krzysztof Trojanowski
Institute of Computer Science
Polish Academy of Sciences
ul. Ordona 21
01-237 Warsaw
Poland
e-mail: trojanow@ipipan.waw.pl

KRZYSZTOF TROJANOWSKI received the M.Sc. degree in computer science from the Warsaw Technical University, Poland, in 1994. He is currently pursuing the Ph.D. degree at the Institute of Computer Sciences, Polish academy of Sciences, Warsaw, Poland, under the supervision of Prof. Z. Michalewicz. His research interests include evolutionary computation techniques.

ZBIGNIEW MICHALEWICZ is Professor of Computer Science at the University of North Carolina at Charlotte. He completed his MSc degree at Technical University of Warsaw in 1974 and he received PhD degree from the Institute of Computer Science, Polish Academy of Sciences, in 1981. His current research interests are in the field of evolutionary computation. He has published several books, including a monograph (3 editions), and over 120 technical papers in journals and conference proceedings. He was the general chairman of the First IEEE International Conference on Evolutionary Computation held in Orlando, June 1994. He has been an invited speaker of many international conferences and a member of 40 various program committees of international conferences during the last 3 years. He is a current member of the editorial board and/or serves as associate editor of 8 international journals (including IEEE Transactions on Evolutionary Computation). He is also the executive vice-president of the IEEE Neural Network Council.

SUSANA ESQUIVEL received the Licentiate in Computer Science degree in 1988 from the Universidad Nacional de San Luis. Presently she is Associate Professor in Languages and Compilers. She has been an elected Academic Director of the Informatics Department since 1994. Her main research area since 1984 has been related to computer systems, intelligent tools and presently she conducts an investigation on Evolutionary Computation.

RAÚL GALLARD received the Computador Cientifico degree in 1975 from the University of Buenos Aires and the Master of Sciences degree from the University of Aston in Birmingham, UK, in 1982. From 1985 he is Full Professor at the Universidad Nacional de San Luis. In 1984 he created a research group on Computer Systems and from that time he has conducted six different projects in the area. Three other research projects under his supervision in different Argentinian Universities are related to theory and aplications of Evolutionary Computation. Professor Gallard is a voting member of the ACM, member of the IEEE and of a many other Computer Science and Educational organizations in Argentina. Research activities of his group are supported by the ANPCYT (National Agency for Promotion of Science and Technology). His current interest is mainly centered on Evolutionary Computation and Intelligent Systems.

MACIEJ MICHALEWICZ, Ph D., Assiociate Professor in Institute of Computer Science of Polish Academy of Sciences. Presently - Head of the Group of Foundations of Artificial Intelligence. Scientific interests are concentrated on data analysis systems, information retrieval systems, knowledge discovery and intelligent information systems. Scientific activities have always been connected with many practical implementations, mainly in medicine and industry.

GUO TAO is a Ph.D. student in computer science at the State Key Laboratory of Software Engineering, Wuhan University, China (degree expected in June, 1999). His research interest is involved in finding a simple and effective genetic algorithm for some problems, such as TSP, constrained optimization.