A Hierarchy of Evolution Programs: An Experimental Study

Zbigniew Michalewicz
Department of Computer Science
University of North Carolina
Charlotte, NC 28223, USA
zbyszek@mosaic.uncc.edu

Abstract

In this paper we present the concept of evolution programs and discuss a hierarchy of such programs for a particular problem. We argue that (for a particular problem) stronger evolution programs (in terms of the problem-specific knowledge incorporated in the system) should perform better than weaker ones. This hypothesis is based on a number of experiments and a simple intuition that problem-specific knowledge enhances an algorithm in terms of its performance; at the same time it narrows the applicability of an algorithm.

1 Introduction

In general, AI problem solving strategies are categorized into *strong* and *weak* methods. A weak method makes few assumptions about the problem domain; hence it usually enjoys wide applicability. On the other hand, it can suffer from combinatorially explosive solution costs when scaling up to larger problems [12]. This can be avoided by making strong assumptions about the problem domain, and consequently exploiting these assumptions in the problem solving method. But a disadvantage of such strong methods is their limited applicability: very often they require significant redesign when applied even to related problems.

At early stages of AI, the general problem solvers (GPSs) were designed as generic tools for approaching complex problems. However, as it turned out, it was necessary to incorporate problem-specific knowledge due to unmanageable complexity of these systems. A similar phenomenon occurred with respect to genetic algorithms (GAs): until recently they were perceived as generic tools useful for optimization of many hard problems. However, it seems that pure GAs (as GPSs) are too domain independent to be useful in many applications.

Recently we proposed [32] a notion of so-called evolution programs (EPs). Roughly speaking, an evolution program is a genetic algorithm enhanced by problem specific knowledge; this knowledge is incorporated in appropriate data structures and problem specific operators. Clearly, many evolution programs can be formulated for a given problem. Such programs may differ in many ways; they can use different data structures for implementing a single individual, "genetic" operators for transforming individuals, methods for creating an initial population, methods for handling constraints of the problem, and parameters (population size, probabilities of applying different operators, etc.). However, they share a common principle: a population of individuals undergoes some transformations, and during this evolution process the individuals strive for survival.

The idea of incorporating a problem specific knowledge in genetic algorithms is not new and has been recognized for some time. Several papers [2], [15], [16], [21], [41], have discussed initialization techniques, different representations, decoding techniques (mapping from genetic representations to 'phenotypic' representations), and the use of heuristics for genetic operators. In [9] 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."

However, the evolution programs have an additional flavor: they approach constrained optimization problems in a systematic way. Evolution programs are not tailored to a particular problem (as it is often the case with the GA-based systems), but their domains are much wider; they are applicable to many different problems with similar classes of constraints (we will see it clearly later in sections 4 and 5).

In this paper we introduce the concept of hierarchy of evolution programs for a particular problem. We argue that if an evolution program EP_q is a stronger method than an evolution program EP_p (for a particular problem), then EP_q should perform better than a weaker system, EP_p. We do not have any proof of this hypothesis, of course, since it is based solely on a number of experiments and a simple intuition that problem-specific knowledge enhances an algorithm in terms of its performance; at the same time it narrows the applicability of an algorithm. However, it is important to emphasize, that developing a stronger, high-performance system, may take much longer time if it involves extensive problem analysis to design specialized representation, operators, and performance enhancements. This is one of the main observations made in the paper; a single (constrained) problem was selected to serve as a vehicle for examining a tradeoff between efforts of (1) finding an effective binary representation and (2) developing code for several, more specialized systems. We shall return to this topic in the last section of the paper.

The paper is organized as follows. We begin by explaining what evolution programs are and how they differ from genetic algorithms. Section 3 introduces the concept of a hierarchy of evolution programs, whereas section 4 provides an example of such hierarchy for a nonlinear transportation problem. Five evolution programs are discussed and computational results are presented. The last section contains some concluding remarks.

2 Evolution Programs

The evolution program (see Figure 1) is a probabilistic algorithm which 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, in any evolution program, is implemented as some (possibly complex) 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 program converges — the best individual hopefully represents a near-optimum (reasonable) solution.

procedure evolution program
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

Figure 1: The structure of an evolution program

It should be clear that the concept of evolution programs is based entirely on the idea of genetic algorithms [25]; the difference is that we allow *any* data structure (i.e., chromosome representation) suitable for a problem together with *any* set of "genetic" operators. In other words, the structure of a genetic algorithm is the same as the structure of an evolution program (Figure 1) and the differences are hidden on the lower level. In EPs chromosomes need not be represented by bit-strings and the alternation process includes other "genetic" operators appropriate for the given structure and the given problem.

The binary alphabet offers the maximum number of schemata per bit of information [17] and consequently the bit string representation of solutions has dominated genetic algorithm research. The binary coding also facilitates theoretical analysis and allows elegant genetic operators: the strongest theoretical results for GAs have assumed binary alphabet for mathematical simplicity. However, it should be emphasized that Holland's work [25] describes genetic algorithms using finite alphabets, and that the 'implicit parallelism' result does not depend on using bit strings [1].

Why do we depart from binary-coded genetic algorithms towards more flexible evolution programs? Even though nicely theorized, binary-coded GAs failed to provide for successful applications in many areas. It seems that the major factor behind this failure is the same one responsible for their success: domain independence. It is clear that binary representations are not always appropriate for highly constrained problems and other representations are often more natural. The central question is how to take advantage of such representations in evolution programs.

During the last ten years, various application–specific variations on the genetic algorithm were reported [7], [19], [23], [24], [31], [39], [40], [43]. These variations include variable length strings (including strings whose elements were *if–then–else* rules [39]), richer structures than binary strings (for example, matrices [43]), and experiments with modified genetic operators to meet the needs of particular applications [29]. In [34] there is a description of a genetic algorithm which uses backpropagation (a neural network training technique) as an operator, together with mutation and crossover that were tailored to the neural network domain. Davis and Coombs [6], [8] described a genetic algorithm that carried out one stage in the process of designing packet-switching communication network; the representation used was not binary and five "genetic" operators (knowledge based, statistical, numerical) were used. These operators were quite different to binary mutation and crossover. Clearly, most researches "modified" their implementations of genetic algorithms either by using non-string chromosome representation or by designing problem specific genetic operators to accommodate the problem to be solved, thus building successful evolution programs.

It seems that a "natural" representation of a potential solution for a given problem plus a family of applicable "genetic" operators might be quite useful in the approximation of solutions of many problems, and this nature-modeled approach (evolution programming) is a promising direction for problem solving in general.

The basic conceptual difference between genetic algorithms and evolution programs is presented in Figures 2 and 3. Classical genetic algorithms, which operate on binary strings, require a modification of an original problem into appropriate (suitable for GA) form; this would include mapping between potential solutions and binary representation, taking care of decoders or repair algorithms, etc. This is not usually an easy task.

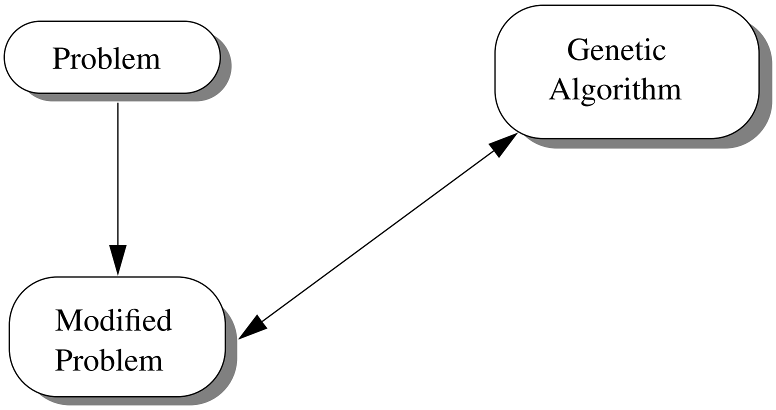

Figure 2: Genetic algorithm approach

On the other hand, evolution programs would leave the problem unchanged, modifying a chromosome representation of a potential solution (using "natural" data structures), and applying appropriate "genetic" operators.

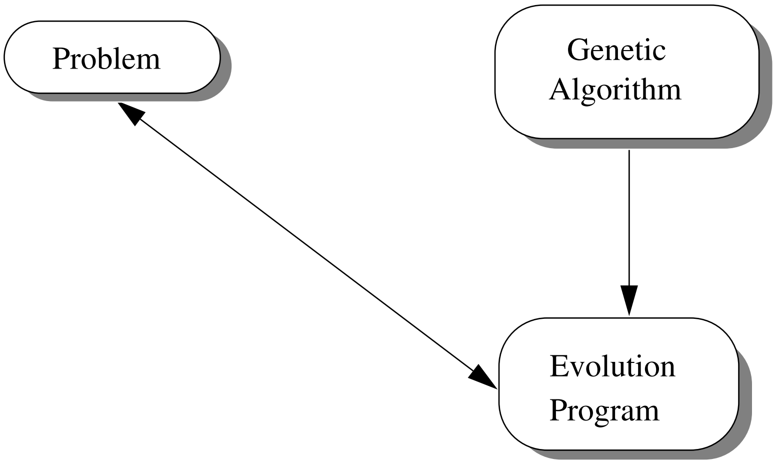

Figure 3: Evolution program approach

In other words, to solve a nontrivial problem using an evolution program, we can either transform the problem into a form appropriate for the genetic algorithm (Figure 2), or we can transform the genetic algorithm to suit the problem (Figure 3). Clearly, classical GAs take the former approach, and EPs the latter.

It is quite hard to draw a line between genetic algorithms and evolution programs. Obviously, any genetic algorithm constitutes an example of an evolution program. However, not every evolution program is a genetic algorithm. For example, it is not clear whether we can use the term "genetic algorithm" for a system with matrix representation and arithmetical crossover [31]? Note that such a system does not have a support of any schema theorem nor of building-block hypothesis. Additionally, the system performs very well with a mutation as a single operator, i.e., where there is no 'recombination' operator at all!

However, we will not analyse this partition any further; instead we present some interesting observations on applying evolution programming techniques to one particular problem (nonlinear transportation problem).

3 A Hierarchy of Evolution Programs

Evolution programs fit somewhere between weak and strong methods. Some evolution programs (as genetic algorithms) are quite weak without making any assumption of a problem domain. Some other programs are more problem specific with a varying degree of problem dependence. For a particular problem P, in general, it is possible to construct a family of evolution programs EP_i, each of which would 'solve' the problem (Figure 4). 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 evolution program EP_5 (Figure 4) is the strongest (i.e., the most problem specific) and it addresses the problem P only. The system EP_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 evolution program, EP_4, can be applied to some (relatively small) class of problems, which includes the problem P; other evolution programs EP_3 and EP_2 work on larger domains, whereas EP_1 is the weakest method (i.e., domain independent) and can be applied to any optimization problem.

Let us denote by $dom(EP_i)$ a set of all problems to which the evolution program EP_i can by applied, i.e., the program returns a feasible solution. Clearly,

$$dom(EP_5) \subseteq dom(EP_4) \subseteq dom(EP_3) \subseteq dom(EP_2) \subseteq dom(EP_1).$$

Obviously, the above example is by no means complete: it is possible to create other evolution programs which would fit between EP_i and EP_{i+1} for some $1 \leq i \leq 4$. Of course, there might be also other evolution programs which overlap with others in the above hierarchy. In other words, the set of evolution programs is partially ordered; we denote the ordering relation by $\prec$ with the following meaning: if $EP_p \prec EP_q$ then the evolution program EP_p is a weaker method than EP_q, i.e., $dom(EP_q) \subseteq dom(EP_p)$. Referring to Figure 4, which displays a hierarchy of evolution programs EP_i, we can write

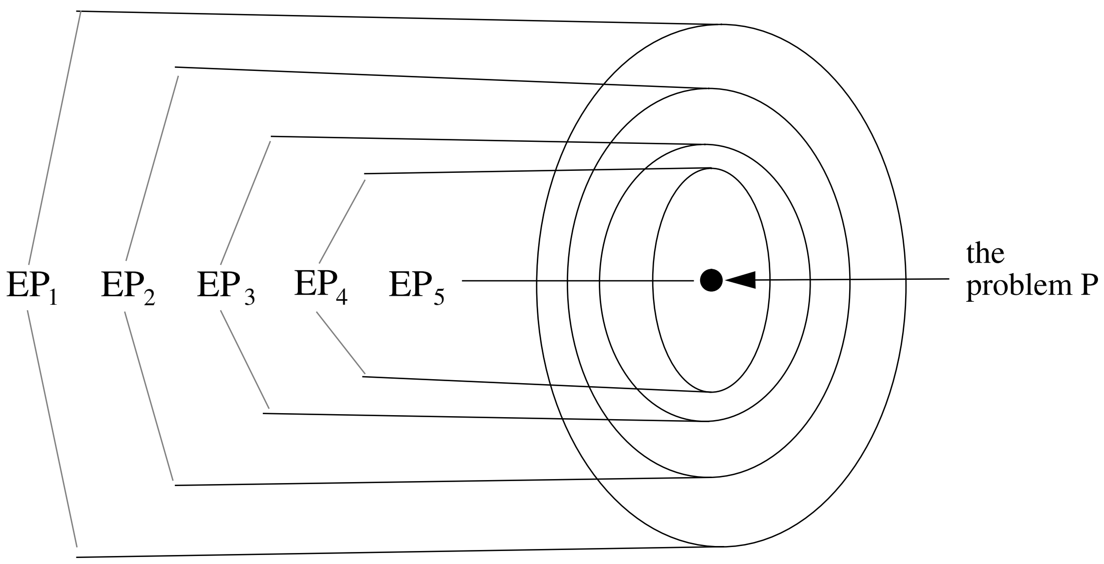

Figure 4: A hierarchy of evolution programs

$EP_1 \prec EP_2 \prec EP_3 \prec EP_4 \prec EP_5$.

The hypothesis is that if $EP_p \prec EP_q$, then the stronger method, EP_q, should in general perform better than a weaker system, EP_p. As stated in the Introduction, we do not have any proof of this hypothesis, however, some other researchers expressed the same idea [9]:

> "It is a truism in the expert system field that domain knowledge leads to increased performance in optimization, and this truism has certainly been borne out of my experience applying genetic algorithms to industrial problems. Binary crossover and binary mutation are knowledge-blind operators. Hence, if we resist adding knowledge to our genetic algorithms, they are likely to underperform nearly any reasonable optimization algorithm that does take into account of such domain knowledge."

Goldberg [18], [17], provides an additional perspective. Let us quote from [18]:

> "Certainly humans have developed very efficient search procedures for narrow classes of problems — genetic algorithms are unlikely to beat conjugate direction or gradient methods on continuous, quadratic optimization problems — but this misses the point. [...] The breadth combined with relative — if not peak — efficiency defines the primary theme of genetic search: robustness."

We visualized this observation on Figure 5, where a (classical) method, Q, works well for a problem, P, and nowhere else, whereas GAs perform reasonably across the spectrum. (The Figure 5 is a simplification of similar figures given in [18] and [17].)

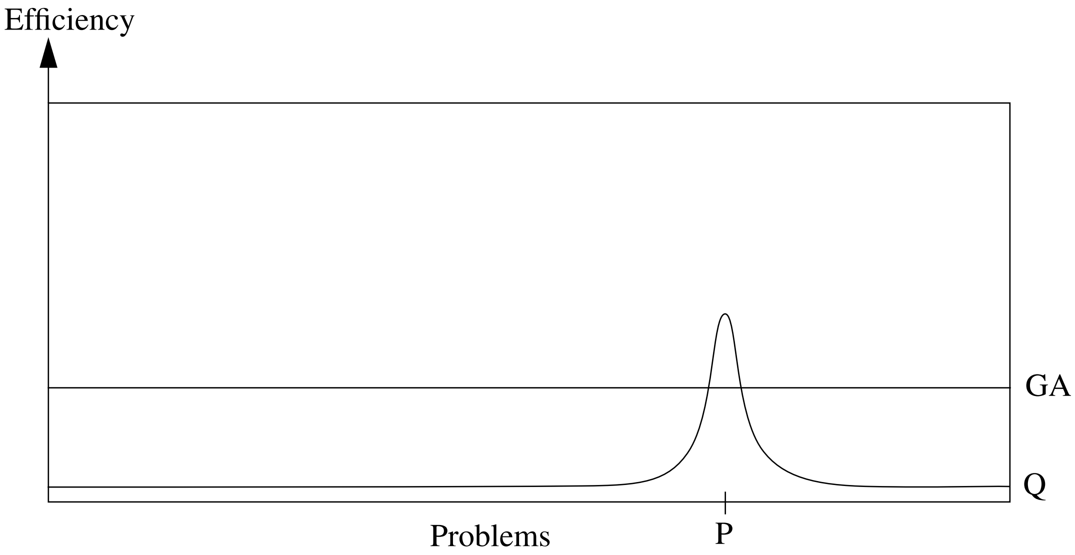

Figure 5: Efficiency/problem spectrum and GAs

However, in the presence of nontrivial constraints, the performance of GAs deteriorates quite often. On the other hand, evolution programs, by incorporating some problem-specific knowledge, may outperform even classical methods (Figure 6).

In the next section we illustrate the above ideas on one particular problem P (nonlinear transportation problem) and five evolution programs EP_i ($i = 1, \ldots, 5$). Also, we present the results of applying a classical method Q (a commercial system) to the problem P.

4 The Problem and Five Evolution Programs

In general, constraints are an integral part of the formulation of any problem. In [13] the authors wrote:

> "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."

In evolution programming, the problem of constraint satisfaction has a special flavor. It is not the issue of selecting an evaluation function with some penalties (as it is the case of binary-coded GAs), but rather selecting "the best" chromosomal representation of

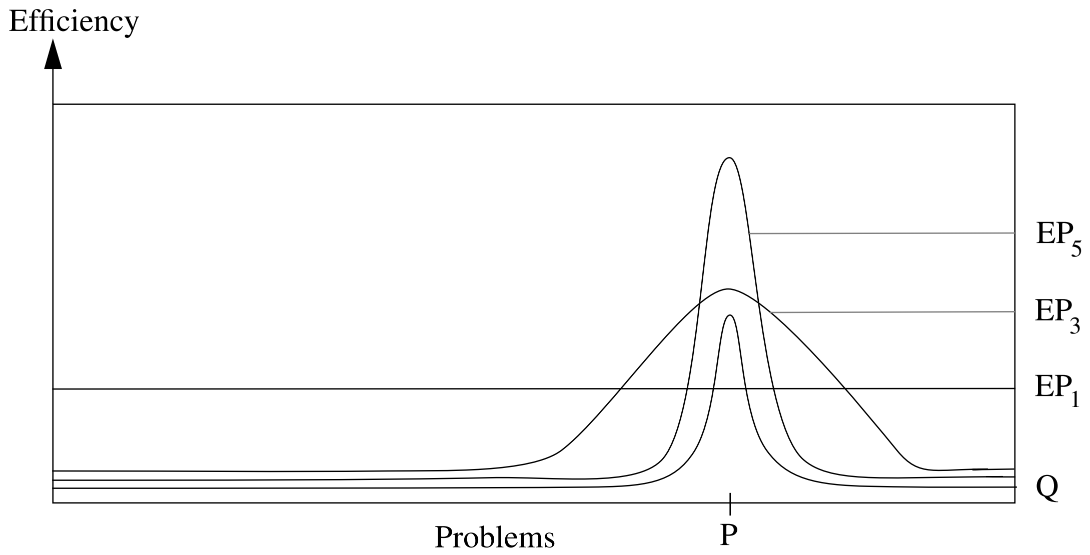

Figure 6: Efficiency/problem spectrum and EPs

solutions together with meaningful genetic operators to 'handle' all constraints imposed by the problem.

We have selected a nonlinear transportation problem as a vehicle to illustrate the ideas from the previous section. This is probably one of the simplest optimization problems that involves constraints in other than a trivial way. The transportation problem, as most highly constrained optimization problems, seems to be a good example of a class of problems for which it is difficult to find an effective fixed-length binary representation, and therefore is more likely to require problem-specific adaptations. Thus we shall use it as an example to investigate the relationship between different evolution programs which can be applied to solve it.

4.1 The Problem P

The transportation problem [42] 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,

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

Let us define a particular nonlinear balanced transportation problem P. Assume 3 sources and 4 destinations. The supply is:

$$source(1) = 10,\ source(2) = 15, \text{ and } source(3) = 20.$$

The demand is:

$$dest(1) = 3,\ dest(2) = 20,\ dest(3) = 5, \text{ and } dest(4) = 17.$$

The total flow in the problem P is 45. The optimum solution for the nonlinear transportation problem may contain neither zeros nor integer values (as it is the case in the linear transportation problem). For example, for some transportation cost functions f_{ij} the following solution might be optimal:

Amount transported

	3.0	**20.0**	**5.0**	**17.0**
10.0	1.34	1.52	0.01	7.13
15.0	1.15	10.39	0.39	3.07
20.0	0.51	8.09	4.60	6.80

For our test problem P we have used the same function f for each flow f_{ij}; a cost-matrix was used to provide variation between flows. The matrix provides the c_{ij}'s which act to scale the basic function shape.

We adopted the following function f of the flows x_{ij}:

$$f(x_{ij}) = \begin{cases} 0 & if \;\; x_{ij} = 0, \\ d + c_{ij} \cdot \sqrt{x_{ij}} & otherwise, \end{cases}$$

for $i = 1, 2, 3$, $j = 1, 2, 3, 4$, where $d = 5.0$, and

$$\begin{array}{llll} c_{11} = 0.0 & c_{12} = 21.0 & c_{13} = 50.0 & c_{14} = 62.0 \\ c_{21} = 21.0 & c_{22} = 0.0 & c_{23} = 17.0 & c_{24} = 54.0 \\ c_{31} = 50.0 & c_{32} = 17.0 & c_{33} = 0.0 & c_{34} = 60.0. \end{array}$$

So the problem P is to minimize

$\sum_{i=1}^{3} \sum_{j=1}^{4} f(x_{ij})$,

subject to the following constraints:

$$\begin{array}{l} x_{11} + x_{12} + x_{13} + x_{14} = 10 \\ x_{21} + x_{22} + x_{23} + x_{24} = 15 \\ x_{31} + x_{32} + x_{33} + x_{34} = 20 \\ x_{11} + x_{21} + x_{31} = 3 \\ x_{12} + x_{22} + x_{32} = 20 \\ x_{13} + x_{23} + x_{33} = 5 \\ x_{14} + x_{24} + x_{34} = 17. \end{array}$$

We solved the above problem P using GAMS (General Algebraic Modeling System), a package for the construction and solution of mathematical programming models [5], with MINOS optimizer. The GAMS' best solution was:

$\sum_{i=1}^{3} \sum_{j=1}^{4} f(x_{ij}) = 430.64$,

which was achieved for

$$\begin{array}{llll} x_{11} = 3.0 & x_{12} = 0.0 & x_{13} = 0.0 & x_{14} = 7.0 \\ x_{21} = 0.0 & x_{22} = 5.0 & x_{23} = 0.0 & x_{24} = 10.0 \\ x_{31} = 0.0 & x_{32} = 15.0 & x_{33} = 5.0 & x_{34} = 0.0. \end{array}$$

This result would serve us as a convenient reference point in evaluation of evolution programs presented in the paper. We refer to GAMS as a classical (gradient–based) method Q for a problem P (see Figure 6).

For a fair comparison of the evolution programs EP_i ($i = 1, \ldots, 5$) presented in this paper, we set population size to 70 and the number of generations to 5,000 for all our experiments. Each experiment was repeated 20 times; all averages for a particular experiment reported in the following subsections refer to averages obtained from these 20 runs. It is also important to point out that the presented evolution programs use different initialization techniques, however, we discuss them later in section 4.7.

4.2 Evolution Program EP_1

The weakest evolution program EP_1 used in the experiments was the GENESIS 1.2ucsd system[1] developed by Nicol Schraudolph at the University of California, San Diego (the system is based on GENESIS 4.5, a genetic algorithm package written by John Grefenstette). In principle, one can use such generic tool to optimize a variety of problems and the $dom(EP_1)$ is virtually unlimited.

Let us exercise the usefulness of this evolution program on our test case, problem P. It is clear that the system will not provide any useful solutions if constraints are not incorporated by means of penalty functions. For example, we performed several runs of EP_1, defining only a domain for each of the twelve variables. Here we did not have much choice—the domain for each variable was selected as a range from zero to the smaller marginal sum for a given row and column:

$$\begin{array}{llll}
0.0 \le x_{11} \le 3.0 & 0.0 \le x_{12} \le 10.0 & 0.0 \le x_{13} \le 5.0 & 0.0 \le x_{14} \le 10.0 \\
0.0 \le x_{21} \le 3.0 & 0.0 \le x_{22} \le 15.0 & 0.0 \le x_{23} \le 5.0 & 0.0 \le x_{24} \le 15.0 \\
0.0 \le x_{31} \le 3.0 & 0.0 \le x_{32} \le 20.0 & 0.0 \le x_{33} \le 5.0 & 0.0 \le x_{34} \le 17.0.
\end{array}$$

Obviously, none of the solutions found by the program satisfied constraints of the problem; a typical output is given below:

$$\begin{array}{llll}
x_{11} = 2.05 & x_{12} = 0.00 & x_{13} = 0.00 & x_{14} = 0.00 \\
x_{21} = 0.00 & x_{22} = 10.65 & x_{23} = 0.00 & x_{24} = 0.00 \\
x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 0.00 & x_{34} = 0.00.
\end{array}$$

As expected, the above nonfeasible solution is without any value for the user. It can be "improved" even further: a solution $x_{ij} = 0.0$ for all $1 \le i \le 3$, $1 \le j \le 4$ yields the optimum transportation cost (zero)!

Clearly, it is necessary to incorporate some penalties on constraints. Since the evolution program EP_1 should not depend on the problem to be solved, we experimented only with some standard penalty functions. We have considered two sets of such penalty functions. The first one (p_i's, moderate penalties) measures each penalty as a linear function of the violation of the constraint, the other set (q_i's, high penalties) squares the violation of the constraint. For our problem P with seven linear equalities, these functions are given below:

$$\begin{aligned}
p_1 &= c \cdot |x_{11} + x_{12} + x_{13} + x_{14} - 10|, \\
p_2 &= c \cdot |x_{21} + x_{22} + x_{23} + x_{24} - 15|, \\
p_3 &= c \cdot |x_{31} + x_{32} + x_{33} + x_{34} - 20|, \\
p_4 &= c \cdot |x_{11} + x_{21} + x_{31} - 3|, \\
p_5 &= c \cdot |x_{12} + x_{22} + x_{32} - 20|,
\end{aligned}$$

[1] The system was run with the dynamic parameter encoding option [37]; however, this option did not improve the performance of the system because the precision was not the issue here. The same comment applies to the evolution program EP_5 discussed later in the paper.

$p_6 = c \cdot |x_{13} + x_{23} + x_{33} - 5|,$
$p_7 = c \cdot |x_{14} + x_{24} + x_{34} - 17|,$

and $q_i = p_i^2/c$ $(i = 1, \ldots, 7)$. In all experiments we used $c = 10.0$; for this number the penalties constitute a significant percentage of the total cost (which, as indicated by the results of the GAMS system, is around 400). The results of the experiments were quite interesting.

The following point represents a *typical* output for experiments with penalties p_i:

$$\begin{array}{llll} x_{11} = 3.00 & x_{12} = 3.77 & x_{13} = 0.00 & x_{14} = 0.00 \\ x_{21} = 0.00 & x_{22} = 1.23 & x_{23} = 0.00 & x_{24} = 13.77 \\ x_{31} = 0.00 & x_{32} = 15.00 & x_{33} = 5.00 & x_{34} = 0.00. \end{array}$$

The above solution is just 'typical': we are unable to provide the *best* output due to the fact that it is relatively hard to evaluate the goodness of nonfeasible solutions. To get a feasible solution from a nonfeasible one, we have to make a few adjustments and the final transportation cost depends on these. For example, the above solution may be corrected into the following feasible solution:

$$\begin{array}{llll} x_{11} = 3.00 & x_{12} = 3.77 & x_{13} = 0.00 & x_{14} = 3.23 \\ x_{21} = 0.00 & x_{22} = 1.23 & x_{23} = 0.00 & x_{24} = 13.77 \\ x_{31} = 0.00 & x_{32} = 15.00 & x_{33} = 5.00 & x_{34} = 0.00, \end{array}$$

which yields the total transportation cost of 453.43. Of course, some other corrections yield better or worse transportations costs. (The above correction was done manually. It was based on a simple observation that the totals of the first row and the fourth column are smaller than the corresponding marginal sums by 3.23; hence we added 3.23 to x_{14}).

In the above example, a manual correction of the nonfeasible solution resulted in a respectable value 453.43. However, it is important to stress that it was possible only because of low dimensions of the problem. The process of finding a 'good' correction of a nonfeasible solution for a 20×20 transportation problem might be as difficult as solving the original problem. It seems that stronger penalties should be used to force the solution into a feasible region.

Indeed, the approach of stronger penalties provided solutions which were "almost" feasible. The following point represents the best output for experiments with penalties q_i:

$$\begin{array}{llll} x_{11} = 3.00 & x_{12} = 6.98 & x_{13} = 0.00 & x_{14} = 0.00 \\ x_{21} = 0.00 & x_{22} = 0.00 & x_{23} = 3.06 & x_{24} = 11.93 \\ x_{31} = 0.00 & x_{32} = 13.02 & x_{33} = 1.93 & x_{34} = 5.03. \end{array}$$

The above solution can be transformed easily (manual rounding) into a feasible solution:

$$\begin{array}{llll} x_{11} = 3.00 & x_{12} = 7.00 & x_{13} = 0.00 & x_{14} = 0.00 \\ x_{21} = 0.00 & x_{22} = 0.00 & x_{23} = 3.00 & x_{24} = 12.00 \\ x_{31} = 0.00 & x_{32} = 13.00 & x_{33} = 2.00 & x_{34} = 5.00, \end{array}$$

which yields 502.53 as the total transportation cost. This cost is worse than the cost of 453.43 we obtained from the moderate penalties approach, however, it should be stressed again that the process of finding a 'good' correction in the moderate penalties approach 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}. Thus, in general, stronger penalties provide better results. At the same time these results are still worse than the results obtained from the commercial software GAMS (system Q in Figure 6). Also, it should be pointed out that 'very strong' penalties do not improve the performance of the program. In extreme, if we assign zero fitness to individuals which violate a constraint, very often the system would settle for the first feasible solution found.

The final (and predictable) conclusion from experiments with EP_1 is that the use of penalty functions do not guarantee feasible solutions and that a 'good' repair may be expensive.

4.3 Evolution Program EP_2

Evolution strategies (ESs) are evolution programs applicable to parameter optimization problems [3], [38]. Early evolution strategies used a floating point number representation, with mutation being the only alternation operator. Mutations were realized by replacing $\vec{x}$ by

$$\vec{x}^{t+1} = \vec{x}^t + N(0, \vec{\sigma}),$$

where $N(0, \vec{\sigma})$ is a vector of independent random Gaussian numbers with a mean of zero and standard deviations $\vec{\sigma}$. They have been applied to various optimization problems with continuously changeable parameters.

The multimembered evolution strategies evolved further [38] to mature as

$(\mu + \lambda)$–ESs and (μ, λ)–ESs;

the main idea behind these strategies was to allow control parameters (like mutation variance) to self-adapt rather than changing their values by some deterministic algorithm.

In the $(\mu+\lambda)$–ES, μ individuals produce λ offspring. The new (temporary) population of $(\mu+\lambda)$ individuals is reduced by a selection process again to μ individuals. On the other hand, in the (μ, λ)–ES, the μ individuals produce λ offspring ($\lambda > \mu$) and the selection process selects a new population of μ individuals from the set of λ offspring only. By doing this, the life of each individual is limited to one generation. This allows the (μ, λ)–ES to perform better on problems with an optimum moving over time, or on problems where the objective function is noisy.

The operators used in the $(\mu + \lambda)$–ESs and (μ, λ)–ESs incorporate two-level learning: their control parameter $\vec{\sigma}$ is no longer constant, nor is it changed by some deterministic algorithm, but it is incorporated in the structure of the individuals and undergoes the evolution process. To produce an offspring, the system acts in two stages:

- select two individuals,

  $$(\vec{x}^1, \vec{\sigma}^1) = ((x_1^1, \ldots, x_n^1), (\sigma_1^1, \ldots, \sigma_n^1)) \text{ and}$$
  $$(\vec{x}^2, \vec{\sigma}^2) = ((x_1^2, \ldots, x_n^2), (\sigma_1^2, \ldots, \sigma_n^2)),$$

  and apply a recombination (crossover) operator. There are two types of crossovers:

  – discrete, where the new offspring is
  $$(\vec{x}, \vec{\sigma}) = ((x_1^{q_1}, \ldots, x_n^{q_n}), (\sigma_1^{q_1}, \ldots, \sigma_n^{q_n})),$$
  where $q_i = 1$ or $q_i = 2$ (so each component comes from the first or second preselected parent),

  – intermediate, where the new offspring is
  $$(\vec{x}, \vec{\sigma}) = (((x_1^1 + x_1^2)/2, \ldots, (x_n^1 + x_n^2)/2), ((\sigma_1^1 + \sigma_1^2)/2, \ldots, (\sigma_n^1 + \sigma_n^2)/2)).$$

  Each of these operators can be applied also in a global mode, where the new pair of parents is selected for *each* component of the offspring vector.

- apply mutation to the offspring $(\vec{x}, \vec{\sigma})$ obtained; the resulting new offspring is $(\vec{x}', \vec{\sigma}')$, where

  $$\vec{\sigma}' = \vec{\sigma} \cdot e^{N(0, \Delta\vec{\sigma})} \text{ and}$$
  $$\vec{x}' = \vec{x} + N(0, \vec{\sigma}'),$$

  where $\Delta\vec{\sigma}$ is a parameter of the method.

Evolution strategies assume a set of $q \geq 0$ inequalities,

$$g_1(\vec{x}) \geq 0, \ldots, g_q(\vec{x}) \geq 0,$$

as part of the optimization problem. If during some iteration an offspring does not satisfy all of these constraints, then the offspring is disqualified, i.e., it is not placed in a new population. If the rate of occurrence of such illegal offspring is high, the ESs adjust their control parameters, e.g., by decreasing the components of the vector $\vec{\sigma}$.

We have used KORR 2.1, Hans-Paul Schwefel and Frank Hoffmeister implementation of a $(\mu + \lambda)$–ES and (μ, λ)–ES, as our next evolution program, EP_2. Clearly, evolution strategies are applicable to parameter optimization problems, hence $dom(EP_2) \subseteq dom(EP_1)$ and consequently, $EP_1 \prec EP_2$.

As stated earlier, EP_2 handles only inequality constraints. Because of that the problem P was rewritten to eliminate the equalities. As a result the objective function has only six variables: y_1, y_2, y_3, y_4, y_5, and y_6, and the transportation problem P is given as:

$$\min f(y_1) + f(y_2) + f(y_3) + f(10.0 - y_1 - y_2 - y_3) + f(y_4) + f(y_5) + f(y_6) + \\ f(15.0 - y_4 - y_5 - y_6) + f(3.0 - y_1 - y_4) + f(20.0 - y_2 - y_5) + \\ f(5.0 - y_3 - y_6) + f(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8.0),$$

where

$$y_1 = x_{11},\ y_2 = x_{12},\ y_3 = x_{13},\ y_4 = x_{21},\ y_5 = x_{22},\ y_6 = x_{23},$$

and the following eighteen constraints hold:

$$\begin{aligned}
g_1 &: \ y_1 \geq 0 \text{ (i.e., } x_{11} \geq 0),\\
g_2 &: \ y_2 \geq 0 \text{ (i.e., } x_{12} \geq 0),\\
g_3 &: \ y_3 \geq 0 \text{ (i.e., } x_{13} \geq 0),\\
g_4 &: \ y_4 \geq 0 \text{ (i.e., } x_{21} \geq 0),\\
g_5 &: \ y_5 \geq 0 \text{ (i.e., } x_{22} \geq 0),\\
g_6 &: \ y_6 \geq 0 \text{ (i.e., } x_{23} \geq 0),\\
g_7 &: \ 10.0 - y_1 - y_2 - y_3 \geq 0 \text{ (i.e., } x_{14} \geq 0),\\
g_8 &: \ 15.0 - y_4 - y_5 - y_6 \geq 0 \text{ (i.e., } x_{24} \geq 0),\\
g_9 &: \ 3.0 - y_1 - y_4 \geq 0 \text{ (i.e., } x_{31} \geq 0),\\
g_{10} &: \ 20.0 - y_2 - y_5 \geq 0 \text{ (i.e., } x_{32} \geq 0),\\
g_{11} &: \ 5.0 - y_3 - y_6 \geq 0 \text{ (i.e., } x_{33} \geq 0),\\
g_{12} &: \ y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8.0 \geq 0 \text{ (i.e., } x_{34} \geq 0),\\
g_{13} &: \ 3.0 - y_1 \geq 0 \text{ (i.e., } x_{11} \leq 3),\\
g_{14} &: \ 10.0 - y_2 \geq 0 \text{ (i.e., } x_{12} \leq 10),\\
g_{15} &: \ 5.0 - y_3 \geq 0 \text{ (i.e., } x_{13} \leq 5),\\
g_{16} &: \ 3.0 - y_4 \geq 0 \text{ (i.e., } x_{21} \leq 3),\\
g_{17} &: \ 15.0 - y_5 \geq 0 \text{ (i.e., } x_{22} \leq 15),\\
g_{18} &: \ 5.0 - y_6 \geq 0 \text{ (i.e., } x_{23} \leq 5).
\end{aligned}$$

The average value of the best transportation cost found (out of 20 independent runs) by EP_2 was 460.75, whereas the best solution found (which yields the total value of 420.74) was

$$\begin{array}{llll}
x_{11} = 3.00 & x_{12} = 2.00 & x_{13} = 5.00 & x_{14} = 0.00\\
x_{21} = 0.00 & x_{22} = 0.00 & x_{23} = 0.00 & x_{24} = 15.00\\
x_{31} = 0.00 & x_{32} = 18.00 & x_{33} = 0.00 & x_{34} = 2.00.
\end{array}$$

As expected, the results of EP_2 are better than results from the previous evolution program EP_1. Additional point for EP_2 is that there is no need for correcting the results to move them into the feasible region. On the other hand it seems that the performance of EP_2 depends on a starting point in the search space (which is given by the user). For that reason, it is quite hard to provide a complete analysis of the system.

4.4 Evolution Program EP_3

The third evolution program EP_3 described here is Genocop (for a full reference, see chapter 7 of [32]; see also [28]). The system was built to optimize a function with any set of linear constraints (equations and/or inequalities).

The chromosomal representation used in Genocop is as follows: for a problem with m variables, a chromosome in a population, representing a permissible solution, is coded as a vector of m floating point numbers $s = \langle v_1, \ldots, v_m \rangle$.

In Genocop the equalities are eliminated at the start, together with an equal number of problem variables; this action removes also part of the space to be searched. The remaining constraints, in the form of linear inequalities, form a convex set which must be searched for a solution. Its convexity ensures that linear combinations of solutions yield solutions without needing to check the constraints — a property used throughout this approach (the connection between convex domains and the special crossover used in the system is discussed fully in [32]). The inequalities can be used to generate bounds for any given variable: such bounds are dynamic as they depend on the values of the other variables and can be efficiently computed.

There are several operators in the Genocop system which proved to be useful on many test problems. These are:

- **uniform mutation:** for this mutation we select a random gene k (from the set of genes of the given chromosome s. If $s_v^t = \langle v_1, \ldots, v_m \rangle$ is a chromosome and the k-th component is the selected gene, the result is a vector $s_v^{t+1} = \langle v_1, \ldots, v'_k, \ldots, v_m \rangle$, where v'_k is a random value (uniform probability distribution) from the range $[left(k), right(k)]$. The values $left(k)$ and $right(k)$ are easily calculated from the set of constraints (inequalities); they depend on the remaining values of the chromosome: $v_1, \ldots, v_{k-1}, v_{k+1}, \ldots, v_m$.

- **boundary mutation:** is a variation of the uniform mutation with v'_k being either $left(k)$ or $right(k)$, each with equal probability.

- **non-uniform mutation:** is one of the operators responsible for the fine tuning capabilities of the system. It is defined as follows: if $s = \langle v_1, \ldots, v_m \rangle$ is a chromosome and the element v_k was selected for this mutation, the result is a vector $s' = \langle v_1, \ldots, v'_k, \ldots, v_m \rangle$, where

$$v'_k = \begin{cases} v_k + \triangle(t, right(k) - v_k) & \text{if a random digit is 0} \\ v_k - \triangle(t, v_k - left(k)) & \text{if a random digit is 1} \end{cases}$$

  The function $\triangle(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\triangle(t, y)$ being close to 0 increases as t increases. This property causes this operator to search the space uniformly initially (when t is small), and very locally at later stages. We have used the following function:

$$\triangle(t, y) = y \cdot \left(1 - r^{(1-\frac{t}{T})^b}\right),$$

where r is a random number from $[0..1]$, T is the maximal generation number, and b is a system parameter determining the degree of non–uniformity.

- **simple crossover:** is defined as follows: if $s_1 = \langle v_1, \ldots, v_m \rangle$ and $s_2 = \langle w_1, \ldots, w_m \rangle$ are crossed after the k-th position, the resulting offspring are: $s_1' = \langle v_1, \ldots, v_k, w_{k+1}, \ldots, w_m \rangle$ and $s_2' = \langle w_1, \ldots, w_k, v_{k+1}, \ldots, v_m \rangle$. Note that the only permissible split points are between individual floating points (using float representation it is impossible to split anywhere else).

  However, such operator may produce nonfeasible offspring. To avoid this problem, we use the property of the convex spaces saying, that there exist $a \in [0, 1]$ such that

  $$s_1' = \langle v_1, \ldots, v_k, w_{k+1} \cdot a + v_{k+1} \cdot (1-a), \ldots, w_m \cdot a + v_m \cdot (1-a) \rangle$$

  and

  $$s_2' = \langle w_1, \ldots, w_k, v_{k+1} \cdot a + w_{k+1} \cdot (1-a), \ldots, v_m \cdot a + w_m \cdot (1-a) \rangle$$

  are feasible. This is a straightforward observation: for $a = 0$ both offspring are just copies of the parents (hence both are feasible). On the other hand, for $a = 1$ the operator works just as the classical crossover swapping parts of the vectors and the offspring need not be feasible. The only question to be answered yet is how to find the largest a ($0 \leq a \leq 1$) to obtain the greatest possible 'information exchange'. In Genocop we implemented a simple iteration where a is initialized to 1, and, if at least one offspring does not satisfy constraints, then $a := a - \sigma$ (σ is a fixed parameter; in all experiments $\sigma = 0.1$). Then, a takes the largest appropriate value found, or 0 if no value satisfied the constraints. The necessity for such actions is small in general and decreases rapidly over the life of the population.

- **arithmetical crossover:** is defined as a linear combination of two vectors: if s_1 and s_2 are to be crossed, the resulting offspring are $s_1' = a \cdot s_1 + (1-a) \cdot s_2$ and $s_2' = a \cdot s_2 + (1-a) \cdot s_1$. This operator uses a static system parameter $a \in [0..1]$, as it always guarantees closedness (it will preserve all inequality constraints).

Since the Genocop (as our evolution program EP_3) can handle only linear constraints, it is clear that $dom(EP_3) \subseteq dom(EP_2)$ and consequently, $EP_2 \prec EP_3$.

The transportation problem P is a problem with $m = 12$ variables; each chromosome is coded as a vector of twelve floating point numbers $\langle y_1, \ldots, y_{12} \rangle$. Then, the problem P is

$$\min \textstyle\sum_{i=1}^{12} f(y_i),$$

where

$$\begin{array}{llll} y_1 = x_{11}, & y_2 = x_{12}, & y_3 = x_{13}, & y_4 = x_{14}, \\ y_5 = x_{21}, & y_6 = x_{22}, & y_7 = x_{23}, & y_8 = x_{24}, \\ y_9 = x_{31}, & y_{10} = x_{32}, & y_{11} = x_{33}, & y_{12} = x_{34}, \end{array}$$

with six independent linear constraints:

$$
\begin{aligned}
&y_1 + y_2 + y_3 + y_4 = 10 \\
&y_5 + y_6 + y_7 + y_8 = 15 \\
&y_9 + y_{10} + y_{11} + y_{12} = 20 \\
&y_1 + y_5 + y_9 = 3 \\
&y_2 + y_6 + y_{10} = 20 \\
&y_3 + y_7 + y_{11} = 5
\end{aligned}
$$

(the seventh equation, $y_4 + y_8 + y_{12} = 10$, is unnecessary, as is linearly dependent on the given six equations); additional linear inequalities are

$$y_i \geq 0, \text{ for } i = 1, \ldots, 12.$$

We performed 20 runs of Genocop. The values of the total transportation cost varied from 420.74 (the worst case) for the following solution (rounded to the second digit after the decimal point):

$$
\begin{array}{llll}
x_{11} = 3.00 & x_{12} = 4.38 & x_{13} = 2.62 & x_{14} = 0.00 \\
x_{21} = 0.00 & x_{22} = 15.00 & x_{23} = 0.00 & x_{24} = 0.00 \\
x_{31} = 0.00 & x_{32} = 0.62 & x_{33} = 2.38 & x_{34} = 17.00,
\end{array}
$$

to the value of 356.98 (the best case) for a solution:

$$
\begin{array}{llll}
x_{11} = 3.00 & x_{12} = 7.00 & x_{13} = 0.00 & x_{14} = 0.00 \\
x_{21} = 0.00 & x_{22} = 13.00 & x_{23} = 2.00 & x_{24} = 0.00 \\
x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 3.00 & x_{34} = 17.00.
\end{array}
$$

The average (out of 20 runs) transportation cost returned by the Genocop system was 405.45. Of course, all obtained solutions were feasible. Clearly, Genocop as a more problem-specific system performed much better than evolution strategies EP_2.

4.5 Evolution Program EP_4

The next evolution program EP_4 described here is Genetic-2n[2] [31]. The system was built to optimize any nonlinear transportation problem, so clearly $dom(EP_4) \subseteq dom(EP_3)$ and consequently, $EP_3 \prec EP_4$. In Genetic-2n a matrix represents a potential solution; appropriate operators were defined for this representation.

The matrix representation was selected as the most natural one — after all, this is how it is presented and solved by hand. Evolution program EP_4 processes a population where each individual is a matrix. The system initializes the population using the following procedure:

[2]There are also systems Genetic-1 and Genetic-2 for the *linear* transportation problem based on vector and matrix representations, respectively; for details, see [32].

input: arrays $dest(k)$, $source(n)$;
output: an array (x_{ij}) such that $x_{ij} \geq 0$ for all i and j, $\sum_{j=1}^{k} x_{ij} = source(i)]$ for $i = 1, 2, \ldots, n$, and $\sum_{i=1}^{n} x_{ij} = dest(j)$ for $j = 1, 2 \ldots, k$, i.e., all constraints are satisfied.

procedure initialization;
begin
  set all numbers from 1 to $k \cdot n$ as unvisited
  repeat
    select an unvisited random number q from 1 to $k \cdot n$ and set it as visited
    set (row) $i = \lfloor (q-1)/k + 1 \rfloor$
    set (column) $j = (q-1) \; mod \; k + 1$
    set $val = min(source(i), \; dest(j))$
    set $x_{ij} = val$
    set $source(i) = source(i) - val$
    set $dest(j) = dest(j) - val$
  until all numbers are visited.
end

Procedure *initialization* creates a matrix of at most $k + n - 1$ non-zero elements such that all constraints are satisfied. Although other initialization procedures are feasible, this method will generate a solution that is at a vertex of the simplex which describes the convex boundary of the constrained solution space.

There is a large group of possible "genetic" operators we can apply to matrices; Genetic-2n uses the following:

- **mutation-1**: this operator would select part of a matrix and re-initialized it: i.e., the operator finds marginal sums, erases all entries in the selected part, and place some integers for all entries such that the new numbers satisfy constraints for marginal sums. As a consequence, this operator attempts to introduce as many zero entires into the matrix as possible (an example follows).

- **mutation-2**: this operator is identical to the previous one except it avoids choosing zero entries by selecting values from a range. The following line of the code for mutation-1:

  set $val = \min(source(i), dest(j))$

  is replaced by:

  set $val_1 = \min(source(i), dest(j)$
  if (i is the last available row) **or**
    (j is the last available column)
    then $val = val_1$
    else set val = random (real) number from $\langle 0, val_1 \rangle$

Some additional modifications are necessary as well; for details the reader is referred to [31] or [32].

- **arithmetical crossover**: for two matrices V_1 and V_2 this operator would produce two offspring, W_1 and W_2, such that $W_1 = q_1 \cdot V_1 + q_2 \cdot V_2$, and $W_2 = q_2 \cdot V_1 + q_1 \cdot V_2$, where q_1 and q_2 are any positive reals such that $q_1 + q_2 = 1$.

Note that all operators would preserve the constraints (sums for rows and columns). For a detailed description of these operators the reader is referred to [31] (it is interesting to note that the mutation-1 and mutation-2 operators correspond to a boundary and uniform mutations of the Genocop system).

The following example will illustrate mutation-1 operator. Let us consider our transportation problem P defined in the section 4.1. Assume that the following matrix V (an individual from the population) was selected as a parent for mutation-1:

2.0	7.0	1.0	0.0
1.0	3.0	1.0	10.0
0.0	10.0	3.0	7.0

Suppose that two rows $\{1, 3\}$ and three columns $\{1, 3, 4\}$ are selected. Then the corresponding submatrix W is:

2.0	1.0	0.0
0.0	3.0	7.0

Note, that $sour_W[1] = 3.0$, $sour_W[2] = 10.0$, $dest_W[1] = 2.0$, $dest_W[2] = 4.0$, $dest_W[3] = 7.0$. After the re-initialization of matrix W, it might have the following values:

0.0	0.0	3.0
2.0	4.0	4.0

So, finally, the child of matrix V after mutation-1, is:

0.0	7.0	0.0	3.0
1.0	3.0	1.0	10.0
2.0	10.0	4.0	4.0

We performed 20 runs of Genetic-2n. The values of the total transportation cost varied from 397.02 (the worst case) for a solution (rounded to the second digit after the decimal point):

$$\begin{array}{llll}
x_{11} = 3.00 & x_{12} = 5.00 & x_{13} = 0.00 & x_{14} = 2.00 \\
x_{21} = 0.00 & x_{22} = 15.00 & x_{23} = 0.00 & x_{24} = 0.00 \\
x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 5.00 & x_{34} = 15.00,
\end{array}$$

to the value of 356.98 (the best case) for a solution:

$$\begin{array}{llll} x_{11} = 3.00 & x_{12} = 7.00 & x_{13} = 0.00 & x_{14} = 0.00 \\ x_{21} = 0.00 & x_{22} = 13.00 & x_{23} = 2.00 & x_{24} = 0.00 \\ x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 3.00 & x_{34} = 17.00 \end{array}$$

(the same solution found by Genocop). However, the average (again, out of 20 runs) transportation cost returned from the Genetic-2n system was 391.65, much better than 405.45 of Genocop. Again, all solutions were feasible. Clearly, Genetic-2n (EP_4) as a more problem-specific system performed better than Genocop (EP_3).

4.6 Evolution Program EP_5

The final evolution program EP_5 decribed here is based again on GENESIS 1.2ucsd system, the very same system we used for experiments described in section 4.2. This time, however, we tried to "tune up" the set of penalty functions to focus the system just on problem P. Additionally, we eliminated all equations: the intuition being, that it should be easier to maintain inequality than equality constraints.

So again, the problem P was rewritten as:

$$\begin{array}{l} \min f(y_1) + f(y_2) + f(y_3) + f(10.0 - y_1 - y_2 - y_3) + f(y_4) + f(y_5) + f(y_6) + \\ \quad f(15.0 - y_4 - y_5 - y_6) + f(3.0 - y_1 - y_4) + f(20.0 - y_2 - y_5) + \\ \quad f(5.0 - y_3 - y_6) + f(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8.0), \end{array}$$

where $y_1 = x_{11}$, $y_2 = x_{12}$, $y_3 = x_{13}$, $y_4 = x_{21}$, $y_5 = x_{22}$, $y_6 = x_{23}$, and

$$\begin{array}{l} 0.0 \le y_1 \le 3.0, \\ 0.0 \le y_2 \le 10.0, \\ 0.0 \le y_3 \le 5.0, \\ 0.0 \le y_4 \le 3.0, \\ 0.0 \le y_5 \le 15.0, \\ 0.0 \le y_6 \le 5.0. \end{array}$$

The six penalty functions we tried to tune were:

$$p_1 = \begin{cases} w_1 \cdot (c_1 + y_1 + y_2 + y_3 - 10.0)^2 & if\ \ 10.0 - y_1 - y_2 - y_3 < 0.0 \\ 0.0 & otherwise \end{cases}$$

$$p_2 = \begin{cases} w_2 \cdot (c_2 + y_4 + y_5 + y_6 - 15.0)^2 & if\ \ 15.0 - y_4 - y_5 - y_6 < 0.0 \\ 0.0 & otherwise \end{cases}$$

$$p_3 = \begin{cases} w_3 \cdot (c_3 + y_1 + y_4 - 3.0)^2 & if\ \ 3.0 - y_1 - y_4 < 0.0 \\ 0.0 & otherwise \end{cases}$$

$$p_4 = \begin{cases} w_4 \cdot (c_4 + y_2 + y_5 - 20.0)^2 & if\ \ 20.0 - y_2 - y_5 < 0.0 \\ 0.0 & otherwise \end{cases}$$

$$p_5 = \begin{cases} w_5 \cdot (c_5 + y_3 + y_6 - 5.0)^2 & if\ \ 5.0 - y_3 - y_6 < 0.0 \\ 0.0 & otherwise \end{cases}$$

$$p_6 = \begin{cases} w_6 \cdot (c_6 + 8.0 - y_1 - y_2 - y_3 - y_4 - y_5 - y_6)^2 \\ \qquad if\ \ y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8.0 < 0.0 \\ 0.0 \quad otherwise, \end{cases}$$

where w_i's and c_i's are additional weights.

As usual, all penalties are added to the objective function. After many experiments (during which we increased and decreased the corresponding weights for constraints which were violated or satisfied, respectively), we arrived with the following set:

$$\begin{array}{ll} c_1 = 2.5, & w_1 = 2.0, \\ c_2 = 0.3, & w_2 = 1.3, \\ c_3 = 5.0, & w_3 = 2.5, \\ c_4 = 5.0, & w_4 = 2.0, \\ c_5 = 0.2, & w_5 = 1.3, \\ c_6 = 0.1, & w_6 = 2.0. \end{array}$$

We do not claim, of course, the the above set of weigths represents the optimal configuration: the tuning was done just "by hand"; if some constraint was not satisfied, we gradually increased the corresponding weights. However, we can make the following two observations:

- the system EP_5 with the above weights performs quite well on the problem P, and
- if we change the problem P by adding additional source or destination, or just by changing the problem specific weights c_{ij}, the evolution program EP_5 would not produce meaningful results.

It is clear then that $dom(EP_5) \subseteq dom(EP_4)$ and consequently, $EP_4 \prec EP_5$.

One of the runs of the EP_5 system gave the following solution:

$$\begin{array}{llll} x_{11} = 2.93 & x_{12} = 6.91 & x_{13} = 0.16 & x_{14} = 0.00 \\ x_{21} = 0.07 & x_{22} = 13.09 & x_{23} = 1.84 & x_{24} = 0.00 \\ x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 3.00 & x_{34} = 17.00. \end{array}$$

Note that all constraints are satisfied, and the value of the objective function is 391.2. The above solution can be manually corrected into the best solution found by Genocop and Genetic-2n:

$$\begin{array}{llll} x_{11}=3.00 & x_{12}=7.00 & x_{13}=0.00 & x_{14}=0.00 \\ x_{21}=0.00 & x_{22}=13.00 & x_{23}=2.00 & x_{24}=0.00 \\ x_{31}=0.00 & x_{32}=0.00 & x_{33}=3.00 & x_{34}=17.00. \end{array}$$

However, the system EP_5 found also a solution with a better value than 391.2, namely 378.25, for the following transportation plan:

$$\begin{array}{llll} x_{11}=2.53 & x_{12}=7.47 & x_{13}=0.00 & x_{14}=0.00 \\ x_{21}=0.47 & x_{22}=12.53 & x_{23}=2.00 & x_{24}=0.00 \\ x_{31}=0.00 & x_{32}=0.00 & x_{33}=3.00 & x_{34}=17.00, \end{array}$$

which is much harder to correct (remember that the optimal solution need not consist of integers. For example, one of the solutions we obtained from Genetic-2n was

$$\begin{array}{llll} x_{11}=3.00 & x_{12}=7.00 & x_{13}=0.00 & x_{14}=0.00 \\ x_{21}=0.00 & x_{22}=12.25 & x_{23}=2.75 & x_{24}=0.00 \\ x_{31}=0.00 & x_{32}=0.75 & x_{33}=2.25 & x_{34}=17.00, \end{array}$$

with the total transportation cost equal to 380.86).

In general, it should be possible to construct a "perfect" evolution program which is tailored to the problem P. We can add additional knowledge to such system by incorporating the transportation costs c_{ij}, characteristics of six independent constraints, possibly with some additional heuristic to modify a feasible solutions. Additional constraints can be added "to guide" the system in desirable direction. However, it should be noted that the difficulty in constructing such system is growing together with the dimensions of the problem and its usefulness would be quite limited (to the problem P only).

4.7 A Comparison

The experimental results presented in the previous sections confirmed the intuitive hypothesis that the problem specific knowledge enhances the performance of the algorithm, narrowing its applicability.

As mentioned in section 4.1, for a fair comparison of the evolution programs we set population size to 70, the number of generations to 5,000 for all our experiments and all runs were repeated 20 times. However, these evolution programs used different techniques in their initialization steps. The first evolution program EP_1 generates its population in a way that the individuals need not be feasible (since the constraints include equations, it would be very surprising if even one generated individual was feasible). The second evolution program EP_2 uses a single (feasible) individual as its starting point; twenty different initial feasible points were generated for these tests. The third program EP_3 makes some number (which is a parameter of the system) of attempts to find an initial feasible individual in the search space. If successful, the initial population would consist of *population_size* identical copies of the found individual. If unsuccessful, the system would prompt the user for a feasible initial point; the set of initial feasible points for these runs was the same one as used for EP_2. The fourth program EP_4 generates and maintains a population of feasible individuals, whereas EP_5 (like EP_1) generates initial population of (possibly) nonfeasible individuals.

In comparing our evolution programs it is important to know about these differences in initialization techniques; however, the results of our experiments indicated that the influence of a particular initialization technique on the system performance was negligible. This is not surprising: for a highly constrained problem in general (and for the transportation problem in particular), a 'feasible' point in the search space does not mean a 'good' point. A heuristic initialization works only in cases where a user has a *good* heuristic to incorporate in the system (and even then it must be done carefully to avoid premature convergence!). There was no improvement in EP_1 or EP_5 when they were initialized by feasible individuals unless one feasible individual was really good. The 'clever' initialization presented in section 4.5 for evolution program EP_4 generated a set of feasible points with average evaluation of 456. This initialization did not enhance the algorithm: it was simply necessary to start with feasible population, since the operators of EP_4 just maintain the feasibility. Other programs (EP_2 and EP_3) used a collection of relatively poor feasible points with the fitness value from the range $\langle 493, 610 \rangle$ (with the average of 562).

We conclude that the initialization process has not influenced the presented results.

5 Conclusions

In the previous section we have examined several evolution programs to approach a particular transportation problem P. It is interesting to note that we have used the same system (namely GENESIS 1.2ucsd) for a construction of the weakest (EP_1) and the strongest (EP_5) method. For the constrained problem P, the system EP_1 (equipped with a standard set of penalty functions) was not very useful, whereas it was a time consuming process to construct the problem-specific system EP_5 with its limited usefulness.

This is an interesting observation: genetic algorithms are perceived as weak methods; however, in the presence of nontrivial constraints, they must be changed into strong methods. Whether we consider a penalty function, decoder, or a repair algorithm, these must be tailored for a specific application. On the other hand, other evolution programs (perceived as much stronger, problem-dependent methods) suddenly seem much weaker. For example, both Genocop and Genetic-2n work well for large classes of problems. This demonstrates a huge potential behind the evolution programming approach. Note also, that stronger evolution programs (EP_3, EP_4) performed much better than a classical method, Q (see, again, Figure 6 from the section 3).

Still an interesting question remains open: for a given problem, P, how weak (or strong) should an evolution program be? In other words, for a given problem, P, should we construct EP_2, or rather EP_4? Our hypothesis suggests that incorporation of problem-specific knowledge gives better results in terms of precision. However, as indicated in the Introduction, the development of a stronger, high-performance system may take a long time if it involves extensive problem analysis to design specialized representation, operators, and performance enhancements. On the other hand, we may have already some standard packages, like Grefenstette's GENESIS, Whitley's GENITOR [44], Davis's OOGA [10], Schraudolph's GENESIS 1.2ucsd, or one of Schwefel's evolution strategy systems [3]. And if we try to find an effective binary representation for a given problem, this may result in little or no software adaptation!

So, is it worthwhile to build a new system from scratch? What is a message we would like to send to a GA/EP practitioner? Should (s)he think simple and go for the most generic approach first? If yes, how and when should (s)he decide on the necessity of investing in more problem-specific software development?

It is quite difficult to provide answers for the above questions in general. If one is solving a transportation problem with hard constraints (i.e., constraints which must be satisfied), there is very little chance that some standard package would produce any feasible solution, or, if we start with a population of feasible solutions and force the system to maintain them, we may get no progress whatsoever — in such cases the system does not perform better than a random search routine. On the other hand, for some other problems such standard packages may produce quite satisfactory results. It seems that the responsibility for making such decisions lies with the user; the decision is a function of many factors, which include the demands of the precision of the required solution, time complexity of the algorithm, cost of developing a new system, feasibility of the solution found (i.e., quantity and importance of the problem constraints), frequency of using the developed system, and others. We illustrate this point by presenting the final example.

Assume that for some engineering application we deal frequently with various optimization problems which can be expressed in mathematical form (formulae for the function and constraints). Assume further, that a typical problem consists of 10–20 variables and the number of (linear) constraints (apart from variable bounds) usually stays between 8 and 15. For example, one of the problems requires minimization of the following function[3]:

$$f(\vec{x}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i,$$

subject to:

$$\begin{aligned}
&2x_1 + 2x_2 + x_{10} + x_{11} \le 10,\\
&2x_1 + 2x_3 + x_{10} + x_{12} \le 10,\\
&2x_2 + 2x_3 + x_{11} + x_{12} \le 10,\\
&-8x_1 + x_{10} \le 0,\\
&-8x_2 + x_{11} \le 0,\\
&-8x_3 + x_{12} \le 0,\\
&-2x_4 - x_5 + x_{10} \le 0,\\
&-2x_6 - x_7 + x_{11} \le 0,
\end{aligned}$$

[3]This optimization problem was taken from [22].

$-2x_8 - x_9 + x_{12} \leq 0,$
$0 \leq x_i \leq 1,\ i = 1,2,3,4,5,6,7,8,9,13,$
$0 \leq x_i,\ i = 10,11,12.$

It might be a time consuming task to adopt the code of the standard packages to approach this problem. There are nine nontrivial constraints here: the number of experiments to tune the penalty functions for this particular problem may discourage the user. Note also that any new problem from the above class of problems would require a separate set of experiments. Moreover, as discussed in the previous section, the penalty function approach does not guarantee the feasibility of the solution. So, if all constraints are important, the solution found may be worthless after all!

On the other hand, the Genocop finds the optimum easily (within 500 iterations) for the above problem: the global solution is $\vec{x}^* = (1,1,1,1,1,1,1,1,1,3,3,3,1)$, and $f(\vec{x}^*) = -15$. There is no need for any code adaptation; the number of different problems to solve is not the issue here. Additionally, we need not worry about the feasibility of the solution found. It seems that in the above circumstances the choice is clear![4]

All systems described in this paper are in public domain. for GENESIS 1.2ucsd contact Nicol Schraudolph (schraudo@cs.ucsd.edu), for KORR 2.1 — Frank Hoffmeister (iwan@gorbi.informatik.uni-dortmund.de), and for Genetic-2n and Genocop — the author of this article (zbyszek@mosaic.uncc.edu).

Acknowledgements:

The author wishes to thank Frank Hoffmeister and Matthew Hobbs for experiments with KORR 2.1 and GAMS, respectively, and Meena Natarajan for correcting typing errors in the earlier version of the paper. The author would also like to thank the anonymous referees for their constructive comments.

References

[1] Antonisse, H.J., *A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint*, in [36], pp.86–91.

[2] Antonisse, H.J. and Keller, K.S., *Genetic Operators for High Level Knowledge Representation*, in [23], pp.69–76.

[3] Bäck, T., Hoffmeister, F., and Schwefel, H.-P., *A Survey of Evolution Strategies*, in [4], pp.2–9.

[4] Belew, R. and Booker, L. (Editors), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[4] Similary, if we deal with a class of problems with nonlinear constraints, we may consider Genocop II — a new system (currently being developed) to handle nonlinear constraints [33].

[5] Brooke, A., Kendrick, D., and Meeraus, A., *GAMS: A User's Guide*, The Scientific Press, 1988.

[6] Coombs, S., and Davis, L., *Genetic Algorithms and Communication Link Speed Design: Theoretical Considerations*, in [23], pp.252–256.

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

[8] Davis, L., and Coombs, S., *Genetic Algorithms and Communication Link Speed Design: Constraints and Operators*, in [23], pp.257–260.

[9] Davis, L., *Adapting Operator Probabilities in Genetic Algorithms*, in [36], pp.61–69.

[10] Davis, L., (Editor), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[11] DeJong, K.A., *Genetic Algorithms: A 10 Year Perspective*, in [19], pp.169–177.

[12] DeJong K.A., Spears, W.M., *Using Genetic Algorithms to Solve NP-Complete Problems*, in [36], pp.124–132.

[13] Dhar, V. and Ranganathan, N., *Integer Programming vs. Expert Systems: An Experimental Comparison*, Communications of the ACM, Vol.33, No.3, pp.323–336, 1990.

[14] Fogel, L.J., Owens, A.J., and Walsh, M.J., *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966.

[15] Forrest, S., *Implementing Semantic Networks Structures using the Classifier System*, in [19], pp.24–44.

[16] Fox, B.R., and McMahon, M.B., *Genetic Operators for Sequencing Problems*, in [35], pp.284–300.

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

[18] Goldberg, D.E., *Zen and the Art of Genetic Algorithms*, in [36], pp.80–85.

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

[20] Grefenstette, J.J., *Optimization of Control Parameters for Genetic Algorithms*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 16, No.1, pp.122–128, 1986.

[21] Grefenstette, J.J., *Incorporating Problem Specific Knowledge into Genetic Algorithms*, in [7], pp.42–60.

[22] Floudas, C.A. and Pardalos, P.M., *A Collection of Test Problems for Constrained Global Optimization Algorithms*, Springer-Verlag, Lecture Notes in Computer Science, Vol.455, 1987.

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

[24] Groves, L., Michalewicz, Z., Elia, P., Janikow, C., *Genetic Algorithms for Drawing Directed Graphs*, Proceedings of the Fifth International Symposium on Methodologies of Intelligent Systems, North-Holland, Amsterdam, pp.268–276, 1990.

[25] Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[26] Jones, D.R. and Beltramo, M.A., *Solving Partitioning Problems with Genetic Algorithms*, in [4], pp.442–449.

[27] Michalewicz, Z. and Janikow, C., *Genetic Algorithms for Numerical Optimization*, Statistics and Computing, Vol.1, No.2, 1991.

[28] Michalewicz, Z. and Janikow, C., *Handling Constraints in Genetic Algorithms*, in [4], pp.151–157.

[29] Michalewicz, Z. and Janikow, C., *GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints*, accepted for publication, Communications of the ACM, 1992.

[30] Michalewicz, Z., Janikow, C., and Krawczyk, J., *A Modified Genetic Algorithm for Optimal Control Problems*, Computers & Mathematics with Applications, Vol.23, No.12, pp.83–94, 1992.

[31] Michalewicz, Z., Vignaux, G.A., Hobbs, M., *A Non-Standard Genetic Algorithm for the Nonlinear Transportation Problem*, ORSA Journal on Computing, Vol.3, No.4, pp.307–316, 1991.

[32] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, AI Series, New York, 1992.

[33] Michalewicz, Z., *Genocop II: A Nonlinear Programming Tool*, in preparation.

[34] Montana, D.J., and Davis, L., *Training Feedforward Neural Networks Using Genetic Algorithms*, Proceedings of the 1989 International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, Los Altos, CA, 1989.

[35] Rawlins, G., *Foundations of Genetic Algorithms*, First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann Publishers, Los Altos, CA, 1991.

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

[37] Schraudolph, N. and Belew, R., *Dynamic Parameter Encoding for Genetic Algorithms*, Machine Learning, Vol.9, pp.9–21, 1992.

[38] Schwefel, H.-P., *Numerical Optimization for Computer Models*, Wiley, Chichester, UK, 1981.

[39] Smith, S.F., *A Learning System Based on Genetic Algorithms*, PhD Dissertation, University of Pittsburgh, 1980.

[40] Smith, S.F., *Flexible Learning of Problem Solving Heuristics through Adaptive Search*, Proceedings of the Eighth International Conference on Artificial Intelligence, Morgan Kaufman Publishers, Los Altos, CA, 1983.

[41] Starkweather, T., McDaniel, S., Mathias, K., Whitley, C., and Whitley, D., *A Comparison of Genetic Sequencing Operators*, in [4], pp.69–76.

[42] Taha, H.A., *Operations Research: An Introduction*, 4th ed, Collier Macmillan, London, 1987.

[43] 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, pp.445–452, 1991.

[44] Whitley, D., *The GENITOR Algorithm and Selection Pressure: Why Rank–Based Allocation of Reproductive Trials is Best*, in [36], pp.116–121.