GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints

Zbigniew Michalewicz Cezary Z. Janikow

Many interesting optimization problems have no known fast solution algorithms either because their objective functions are complicated (and possibly discontinuous) or because they contain difficult constraint structures. For such problems, in practice, any near-optimal solution would be desirable if obtained with a reasonable effort. For many problems with difficult objective functions this can be done using probabilistic search methods; however, such methods have no general way of handling constraints.

In general, constraints are an integral part of the formulation of any problem. In [6] 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.

Constraint satisfaction problems (CSPs) have been studied extensively in the operations research (OR) and artificial intelligence (AI) literature. In OR formulations constraints are quantitative, and the solver (such as the Simplex algorithm) optimizes (maximizes or minimizes) the value of a specified objective function subject to the constraints. In contrast, AI research has focused on inference-based approaches with mostly symbolic constraints. The inference mechanism employed include theorem provers, production rule interpreters, and various labeling procedures such as those used in truth maintenance systems."

In this article we present a new approach to solving numerical optimization problems with linear constraints, based on genetic algorithms. Our new methodology seems to fit between the OR and AI approaches. First, it uses the OR technique for problem representation, *i.e.* the formulation of constraints is quantitative. On the other hand, our method uses a genetic algorithm, which is considered as an AI based search method.

Genetic algorithms ([14], [5], [4], [8]) are a class of stochastic algorithms which simulate both the natural inheritance by genetics and the Darwinian strive for survival. As stated in [4]:

"…the metaphor underlying genetic algorithms 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."

Since the genetic approach is basically an accelerated search of the feasible solution space, introducing constraints can be potentially advantageous and can improve the behavior of the technique by limiting the space to be searched. However, all traditional GA approaches do not use this fact and rather employ techniques aimed at minimizing the negative effect of such constraints. This, in turn, often increases the search space by allowing some infeasible solutions outside the constrained solution space.

In some optimization techniques, such as linear programming, equality constraints are welcome since it is known that the optimum, if it exists, is situated at the surface of the convex set. Inequalities are converted to equalities by the addition of slack variables, and the solution method proceeds by moving from vertex to vertex, around the surface.

In contrast, for a method that generates solutions randomly, such equality constraints are a nuisance. In our system they 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 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.

This new method lies somewhere between basic hill-climbing mathematical techniques, that require a great deal of information about the function to be optimized and which can easily diverge to local optima, and the qualitative artificial intelligence search methods.

The purpose of this paper is twofold. Firstly, we present an overview of genetic algorithms for optimization problems and discuss the current approaches to handle constraints. (In [8], a major publication on genetic algo-

rithms, there is only a small section on "constraints".) Secondly, we present a new optimization methodology, based on genetic algorithms, which handles a set of linear constraints in a problem independent way. We also describe a possible application of the methodology to the nonlinear transportation problem and discuss some experimental results.

What is a Genetic Algorithm?

Genetic algorithms represent a class of adaptive algorithms whose search methods are based on simulation of natural genetics. They belong to the class of probabilistic algorithms; yet, they are very different from random ones, as they combine elements of both directed and stochastic search. Also, they are superior to *hill-climbing* methods, since at any time a GA provides for both exploitation of the best solutions, and exploration of the search space; because of this, GAs are also more robust that existing directed search methods. Another important property of such genetic based search methods is their domain-independence.

In general, a GA performs a multi-directional search by maintaining a population of potential solutions and encourages information formation and exchange between these directions. This population undergoes a simulated evolution: at each generation the relatively "good" solutions reproduce, while the relatively "bad" solutions die. To distinguish between different solutions we need some evaluation function which plays the role of an environment.

The structure of a simple genetic algorithm is shown in Figure 1. During iteration t, the genetic algorithm maintains a population of potential solutions (called chromosomes following the natural terminology) $P(t) = \{x^t_1,\ldots,x^t_n\}$. Each solution x^t_i is evaluated to give some measure of its "fitness." Then, a new population (iteration $t + 1$) is formed by selecting the more fitted individuals. Finally, random members of this new population undergo reproduction by means of crossover and mutation—to form new solutions.

Crossover combines the features of two parent chromosomes to form two similar off-spring by swapping corresponding segments of the parents. For example, if the two parents are $\langle v_1,\ldots,v_m\rangle$ and $\langle w_1,\ldots,w_m\rangle$ (with each element called a *gene*), then crossing the chromosomes after the kth gene ($1 \leq k \leq m$) would produce the offspring $\langle v_1,\ldots,v_k, w_{k+1},\ldots,w_m\rangle$ and $\langle w_1,\ldots,w_k, v_{k+1},\ldots,v_m\rangle$. The intuition behind the applicability of the crossover operator is information exchange between different potential solutions.

Mutation arbitrarily alters one or more genes of a selected chromosome by a random change with a probability equal to some mutation rate. The intuition behind the mutation operator is the induction of some extra variability into the population.

A genetic algorithm for a particular problem must have the following five components:

- A genetic representation for potential solutions to the problem,
- A way to create an initial population of potential solutions,
- An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness",
- Genetic operators that alter the composition of children during reproduction,

```
procedure genetic algorithm
begin
   t = 0
   initialize P(t)
   evaluate P(t)
   while (not termination-condition) do
   begin
      t = t + 1
      select P(t) from P(t − 1)
      recombine P(t)
      evaluate P(t)
   end
end
```

Figure 1. A simple genetic algorithm.

- Values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

The theoretical foundations of genetic algorithms rely on a binary string representation of solutions and on the notion of a schema (see *e.g.* [14])—a template allowing exploration of similarities among chromosomes. In a population of size n of chromosomes of length m, between 2^m and $n \cdot 2^m$ different schemata may be represented; at least n^3 of them are processed usefully—Holland has called this property an *implicit parallelism,* as it is obtained without any extra memory/processing requirements.

A growth equation for schemata shows that selection increases sampling rates of the above-average schemata, and that this change is exponential. However, no new schemata (not represented in the initial $t = 0$ sampling) can be formed, which prohibits the application of selection alone. This is exactly why the crossover operator is introduced—to enable structured yet random information exchange. Additionally, the mutation operator introduces greater variability into the population. It has been shown (*e.g.* [8]) that the negative effect of these two operators on the growth of a schema is minimal for a specific kind of schemata—called the *building blocks.* Therefore, the coding of a problem should encourage the formation of such blocks.

For several reasons (simplicity of analysis, the elegance of operators, requirements for speed), the binary string representation has dominated genetic algorithms research. However, the "implicit parallelism" result does not depend on the use of bit strings (see [1])—hence it may be worthwhile to experiment with richer data structures and other "genetic" operators. This may be important in particular in the presence of nontrivial constraints to the problem.

The Constraints Problem in Genetic Algorithms

Three different approaches to the constraint problem in genetic algorithms have previously been proposed. The first two involve transforming potential solutions of the problem into a form suitable for a genetic algorithm and using penalty functions or applications of "decoders" or "repair" algorithms. The third approach involves modifying the genetic algorithm to suit the problem by using new data structures and new genetic operators.

In this section, after defining the class of linearly constrained optimization problems, we briefly discuss these three approaches in turn.

The Problem

We consider a class of optimization problems that can be formulated as follows:
Optimize a function $f(x_1, x_2,\ldots,x_q)$, subject to the following sets of linear constraints:

1. Domain constraints: $l_i \le x_i \le u_i$ for $i = 1, 2,\ldots,q$. We write $\vec{l} \le \vec{x} \le \vec{u}$ where $\vec{l} = \langle l_1,\ldots,l_q\rangle$, $\vec{u} = \langle u_1,\ldots,u_q\rangle$, $\vec{x} = \langle x_1,\ldots,x_q\rangle$.
2. Equalities: $A\vec{x} = \vec{b}$, where $\vec{x} = \langle x_1,\ldots,x_q\rangle$, $A = (a_{ij})$, $\vec{b} = \langle b_1,\ldots,b_p\rangle$, $1 \le i \le p$, and $1 \le j \le q$ (p is the number of equations).
3. Inequalities: $C\vec{x} \le \vec{d}$, where $\vec{x} = \langle x_1,\ldots,x_q\rangle$, $C = (c_{ij})$, $\vec{d} = \langle d_1,\ldots,d_m\rangle$, $1 \le i \le m$, and $1 \le j \le q$ (m is the number of inequalities).

This formulation is general enough to handle a large class of standard Operations Research optimization problems with linear constraints and any objective function. The example considered later, the nonlinear transportation problem, is one of many problems in this class.

Penalty functions

One way of dealing with candidates that violate the constraints is to generate potential solutions without considering the constraints and then penalizing them by decreasing the "goodness" of the evaluation function. In other words, a constrained problem is transformed to an unconstrained problem by associating a penalty with all constraint violations and the penalties are included in the function evaluation. Thus, the earlier problem from is transformed into:
Optimize a function

$$f(x_1, x_2, \ldots, x_q) + \epsilon \cdot \delta \sum_{i=1}^{p+m} \Phi_{i,}$$

where p and m are, respectively, the numbers of equations and inequalities (i.e. $p + m$ is the number of all constraints), δ is a penalty coefficient, ϵ is -1 for maximalization and $+1$ minimalization problems, and Φ_i is a penalty related to the i-th constraint ($i = 1, \ldots, p + m$).

However, though the evaluation function is usually well defined, there is no accepted methodology for combining it with the penalty. Davis discusses this problem in [4]:

"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 structure 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".

In [30] and [26] the authors present the most recent approaches for using penalty functions in GAs for constrained optimization problems. However, the paper by Siedlecki and Sklansky [30] discusses a particular constrained optimization problem. This problem is frequently encountered in the design of statistical pattern classifiers, however, the proposed method is problem specific. The paper by Richardson, Palmer, Liepins, and Hillard [26] examines the penalty approach discussing the strengths and weaknesses of various penalty function formulations and illustrate a technique for solving three dimensional constrained problem. However, in the last section of their article the authors say:

"The technique used to solve the three dimensional problem described above can't be generalized since quantities like the trend and maximum derivative are seldom available".

We do not believe this to be a promising direction. For a heavily constrained problem, such as the transportation problem, the probability of generating an infeasible candidate is too great to be ignored. The technique based on penalty functions, at the best, seems to work reasonably well for narrow classes of problems and for few constraints.

3.3 Decoders and repair algorithms

Another approach concentrates on the use of special representation mappings (decoders) which guarantee (or at least increase the probability of) the generation of a feasible solution and on the application of special repair algorithms to "correct" any infeasible solutions so generated. However, decoders are frequently computationally intensive to run ([4]), not all constraints can be easily implemented this way, and the resulting algorithm must be tailored to the particular application. The same is true for repair algorithms. Again, we do not believe this is a promising direction for incorporating constraints into genetic algorithms. Repair algorithms and decoders may work reasonably well but are highly problem specific, and the chances of building a general genetic algorithm to handle different types of constraints based on this principle seem to be slim.

3.4 Specialized data structures and genetic operators

The last and relatively new approach to incorporating constraints in genetic algorithms is to introduce richer data structures together with an appropriate family of applicable "genetic" operators which can "hide" the constraints presented in the problem (see 19, 20, 22).

Several experiments [13, 23, 32, 33] indicate the usefulness of this approach, but it is not always possible, for an arbitrary set of constraints, to develop an efficient data structure hiding such the constraints. In addition, such structures require specialized genetic operators which maintain feasibility. Such extensions lack also the theoretical basis enjoyed by the classical genetic operators. Despite these objections, experimental results suggest that this approach may be promising; on the other hand, the particular choice of representation and operators must be tailored to the specific problem to be solved, and selected by the user.

A New Methodology: The GENOCOP System

The proposed methodology provides a way of handling constraints that is both general and problem independent. It combines some of the ideas seen in the previous approaches, but in a totally new context. The main idea behind this approach lies in (1) an elimination of the equalities present in the set of constraints, and (2) careful design of

special “genetic” operators, which guarantee to keep all “chromosomes” within the constrained solution space. This can be done very efficiently for linear constraints and while we do not claim these results extend easily to nonlinear constraints, the former class contains many interesting optimization problems.

The Idea

Before we discuss the details of GENOCOP system, we present a small example which should provide some insight into the proposed methodology.

Let us assume we wish to minimize a function of six variables:

$$f(x_1, x_2, x_3, x_4, x_5, x_6),$$

subject to the following constraints:

$$x_1 + x_2 + x_3 = 5$$
$$x_4 + x_5 + x_6 = 10$$
$$x_1 + x_4 = 3$$
$$x_2 + x_5 = 4$$
$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0, x_6 \geq 0.$$

This problem is equivalent to the transportation problem with two sources and three destinations.

We can take an advantage from the presence of four independent equations and express four variables as functions of the remaining two:

$$x_3 = 5 - x_1 - x_2$$
$$x_4 = 3 - x_1$$
$$x_5 = 4 - x_2$$
$$x_6 = 3 + x_1 + x_2.$$

We have reduced the original problem to the optimization problem of a function of two variables x_1 and x_2:

$$g(x_1, x_2) = f(x_1, x_2, (5 - x_1 - x_2), (3 - x_1), (4 - x_2), (3 + x_1 + x_2),$$

subject to the following constraints (inequalities only):

$$x_1 \geq 0, x_2 \geq 0$$
$$5 - x_1 - x_2 \geq 0$$
$$3 - x_1 \geq 0$$
$$4 - x_2 \geq 0$$
$$3 + x_1 + x_2 \geq 0.$$

These inequalities can be further reduced to:

$$0 \leq x_1 \leq 3$$
$$0 \leq x_2 \leq 4$$
$$x_1 + x_2 \leq 5.$$

This would complete the first step of our algorithm: elimination of equalities.

Now let us consider a single point from the search space, $\vec{x} = \langle x_1, x_2 \rangle = \langle\ 1.8, 2.3\ \rangle$ If we try to change the value of variable x_1 without changing the value of x_2 (uniform mutation), the variable x_1 can take any value from the range: $[0, 5 - x_2] = [0, 2.7]$. Additionally, if we have two points within search space, $\vec{x} = \langle x_1, x_2 \rangle = \langle\ 1.8, 2.3\ \rangle$ and $\vec{x'} = \langle x'_1, x'_2 \rangle = \langle\ 0.9, 3.5\ \rangle$, then any linear combination $a\vec{x} + (1 - a)\vec{x'}$, $0 \leq a \leq 1$, would yield a point within search space, *i.e.* all constraints must be satisfied (whole arithmetical crossover). Therefore, both examples of

operators would not move a vector outside the constrained solution space.

The above example explains the main idea behind the GENOCOP system. Linear constraints were of two types: equalities and inequalities. We first eliminated all the equalities, reducing the number of variables and appropriately modifying the inequalities. Reducing the set of variables both decreased the length of the representation vector and reduced the search space. Since we were left with only linear inequalities, the search space was convex—which, in the presence of closed operators, could be searched efficiently. The problem then became one of designing such closed operators. We achieved this by defining them as being context-dependent, that is dynamically adjusting to the current context.

Now we are ready to discuss details of the GENOCOP system.

Elimination of Equalities

In this section we use notation introduced earlier. Suppose the equality constraint set is represented in matrix form:

$$A\vec{x} = \vec{b}.$$

We assume there are p independent equality equations (there are easy methods to verify this), *i.e.* there are p variables $x_{i1}, x_{i2}, \ldots, x_{ip}$ ($\{i_1,\ldots,i_p\} \subseteq \{1, 2,\ldots,q\}$) which can be determined in terms of the other variables. These can, therefore, be eliminated from the problem statement, as follows.

We can split the array A vertically into two arrays A_1 and A_2, such that the j-th column of the matrix A belong to A_1 iff $j \in \{i_1,\ldots,i_p\}$. Thus $A^{-1}{}_1$ exists. Similarly, we split matrix C and vectors $\vec{x}$, $\vec{l}$, $\vec{u}$ (*ie.* $\vec{x^1} = \langle x_{i1},\ldots,x_{ip} \rangle$, $\vec{l_1} = \langle l_{i1},\ldots,l_{ip} \rangle$, and $\vec{u_1} = \langle u_{i1},\ldots,u_{ip} \rangle$).

Then

$$A_1\vec{x^1} + A_2\vec{x^2} = \vec{b}.$$

It is easily seen that

$$\vec{x^1} = A_1{}^{-1}\,\vec{b} - A_1{}^{-1}A_2\vec{x^2}.$$

Using the above rule, we can eliminate the variables $x_{i1},\ldots,x_{ip}$ replacing them by a linear combination of remaining variables. However, each variable x_{ij} ($j = 1, 2,\ldots, p$) is constrained additionally by a domain constraint: $l_{ij} \le x_{ij} \le u_{ij}$. Eliminating all variables x_{ij} leads us to introduce a new set of inequalities:

$$\vec{l_1} \le A_1{}^{-1}\,\vec{b} - A_1{}^{-1}A_2\vec{x^2} \le \vec{u_1},$$

which is added to the original set of inequalities.

The original set of inequalities,

$$C\vec{x} \le \vec{d},$$

can be represented as

$$C_1\vec{x^1} + C_2\vec{x^2} \le \vec{d}.$$

This can be transformed into

$$C_1(A_1{}^{-1}\,\vec{b} - A_1{}^{-1}A_2\vec{x^2}) + C_2\vec{x^2} \le \vec{d}.$$

So, after eliminating the p variables $x_{i1}, \ldots, x_{ip}$, the final set of constraints consists of the following inequalities only:

1. original domain constraints: $\vec{l_2} \le \vec{x^2} \le \vec{u_2}$,

2. new inequalities: $A_1\overrightarrow{l_1} \leq \vec{b} - A_2\overrightarrow{x^2} \leq A_1\overrightarrow{u_1}$,
3. original inequalities (after removal of x^1 variables): $(C_2 - C_1A^{-1}{}_1A_2)\overrightarrow{x^2} \leq \vec{d} - C_1A^{-1}{}_1\vec{b}$.

Representation Issues

The binary representation traditionally used in genetic algorithms has some drawbacks when applied to multidimensional, high precision numerical problems. For example, for 100 variables with domains in the range $[-500, 500]$ where the precision of six digits after the decimal point is required the length of the binary solution vector is 3000. This, in turn, generates the search space of about 10^{1000}. For such problems genetic algorithms perform poorly. In addition, such binary representation lacks the ability to perform fine local exploitation (*e.g.* [11]).

We have designed (see [16, 25, 21]) a specialized version of GA to work on numerical optimization problems with real valued domains. The main objective behind this implementation was to move the genetic algorithm closer to the problem space. Such a move forced, but also allowed, the operators to be more problem specific—by utilizing some real space specific characteristics. For example, this representation has the property that two points close to each other in the representation space must also be close in the problem space, and vice versa. This is not generally true in the binary approach, where the distance in a representation is normally defined by the number of different bit positions, even though some approaches (as Gray coding, see [28]) reduces such discrepancy. We use a floating point representation as the one conceptually closest to the problem space and also allowing for an easy and efficient implementation of closed and dynamic operators.

Though the floating point representation does not enjoy the theoretical foundations given for the binary case (see [14]) it has been shown empirically to give respectable results for real valued parameter optimization [3]. The floating point representation is, of course, not essential to the implementation of this constraints handling methodology. Rather, it was selected for simplification of genetic operator's definitions and for obtaining a better performance of the genetic algorithm itself.

The exact representation used in GENOCOP system is as follows: for a problem with m variables, the i-th chromosome in a population $P(t)$, representing a permissible solution, is coded as a vector of m floating point numbers $s^t{}_i = \langle v_1, \ldots, v_m \rangle$ (when the generation number t and the chromosome number i are not important, we will write simply s). The precision of such a representation is fixed for a given machine, and based on the precision of the floating point (or double, if needed) type.

Initialization Process

A genetic algorithm requires a population of potential solutions to be initialized and then maintained during the process. The GENOCOP system generates an initial set of potential solutions as follows:

- A portion of potential solutions is selected randomly from the space of the whole feasible region,
- The remaining part of potential solutions consists entirely of points on the boundary of the solution space.

Their proportion *prop* is related to the relative frequencies of different operators: some operators (as boundary mutation) move the offspring towards the boundary of the solution space, whereas others (as arithmetical crossovers) spread the offspring over the whole feasible space.

Genetic Operators

The operators used in GENOCOP system are quite different from the classical ones for the following reasons:

1. We deal with a real valued space R^q, where a solution is coded as a vector with floating point type components.
2. The genetic operators are dynamic, *i.e.* a value of a vector component depends on the remaining values of the vector,
3. Some genetic operators are non-uniform, *i.e.* their action depends on the age of the population.

The value of the i-th component of a feasible solution $\vec{s} = \langle v_1, \ldots, v_m \rangle$ is always in some (dynamic) range $[l, u]$; the bounds l and u depend on the other vector's values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_m$, and the set of inequalities. We say that the i-th component (i-th gene) of the vector s is *movable* if $l < u$.

It is important to understand how these differences affect the genetic operators. The first one affects only

the way a mutation is performed after an appropriate gene has been selected. In contrast, the next two affect both the selection of a gene for mutation and the selection of crossover points. They also dictate the space (domain) and probability distribution for the mutation and the behavior during crossover.

Before we describe the operators, we present two important characteristics of convex spaces (due to the linearity of the constraints, the solution space is always a convex space $\boldsymbol{S}$), which play an essential role in the definition of these operators:

1. For any two points s_1 and s_2 in the solution space $\boldsymbol{S}$, the linear combination $a \cdot s_1 + (1 - a) \cdot s_2$, where $a \in [0, 1]$, is a point in $\boldsymbol{S}$.
2. For every point $s_0 \in \boldsymbol{S}$ and any line p such that $s_0 \in p$, p intersects the boundaries of $\boldsymbol{S}$ at precisely two points, say ${l_p}^{s_0}$ and ${u_p}^{s_0}$.

Since we are only interested in lines parallel to each axis, to simplify the notation we denote by ${l^s}_{(i)}$ and ${u^s}_{(i)}$ the i-th components of the vectors ${l^s}_p$ and ${u^s}_p$, respectively, where the line p is parallel to the axis i. We assume further that ${l^s}_{(i)} \leq {u^s}_{(i)}$.

Because of intuitional similarities, we cluster the operators into the standard two classes: mutation and crossover. The proposed crossover and mutation operators use the two properties to ensure that the offspring of a point in the convex solution space $\boldsymbol{S}$ belongs to $\boldsymbol{S}$. However, some operators (*e.g.* non-uniform mutation) have little to do with GENOCOP methodology: they have other "responsibilities" like fine tuning and prevention of premature convergence. For a detailed discussion on these topics, the reader is referred to [16, 17, 21].

Mutation group: Mutations are quite different from the traditional one with respect to both the actual mutation (a gene, being a floating point number, is mutated in a dynamic range) and to the selection of an applicable gene. A traditional mutation is performed on static domains for all genes. In such a case the order of possible mutations on a chromosome does not influence the outcome. This is not true anymore with the dynamic domains. To solve the problem we proceed as follows: we randomly select $p_{um} \cdot pop\_size$ chromosomes for uniform mutation, $p_{bm} \cdot pop\_size$ chromosomes for boundary mutation, and $p_{nm} \cdot pop\_size$ chromosomes for non-uniform mutation (all with possible repetitions), where p_{um}, p_{bm}, and p_{nm} are probabilities of the three mutations defined below. Then, we perform these mutations in a random fashion on the selected chromosome.

- Uniform mutation for this mutation we select a random gene k (from the set of movable genes of the given chromosome s determined by its current context). 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 $[l_{(k)}^{s_v^t}, u_{(k)}^{s_v^t}]$. The dynamic values $l_{(k)}^{s_v^t}$ and $u_{(k)}^{s_v^t}$ are easily calculated from the set of constraints (inequalities).
- Boundary mutation is a variation of the uniform mutation with v_k' being either $l_{(k)}^{s_v^t}$ or $u_{(k)}^{s_v^t}$, 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 $s_v^t = \langle v_1, \ldots, v_m \rangle$ is a chromosome and the element v_k was selected for this mutation from the set of movable genes, the result is a vector $s_v^{t+1} = \langle v_1, \ldots, v_k', \ldots, v_m \rangle$, with $k \in \{1, \ldots, n\}$, and

$$v_k' = \begin{cases} v_k + \triangle(t, u_{(k)}^{s_v^t} - v_k) & \text{if a random digit is 0} \\ v_k - \triangle(t, v_k - l_{(k)}^{s_v^t}) & \text{if a random digit is 1.} \end{cases}$$

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:

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

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.

Crossover group: Chromosomes are randomly selected in pairs for application of the crossover operators according to appropriate probabilities.

- simple crossover is defined as follows: if $s_v^t = \langle v_1, \ldots, v_m \rangle$ and $s_w^t = \langle w_1, \ldots, w_m \rangle$ are crossed after the k-th position, the resulting offspring are:

$s_v^{t+1} = \langle v_1, \ldots, v_k, w_{k+1}, \ldots, w_m \rangle$ and $s_w^{t+1} = \langle w_1, \ldots, w_k, v_{k+1}, \ldots, v_m \rangle$. 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 offspring outside of the convex solution space $\boldsymbol{S}$. To avoid this problem, we use the property of the convex spaces saying, that there exist $a \in [0, 1]$ such that

$$s_v^{t+1} = \langle v_1, \ldots, v_k, w_{k+1} \cdot a + v_{k+1} \cdot (1 - a), \ldots, w_m \cdot a + v_m \cdot (1 - a) \rangle \in S$$

and

$$s_w^{t+1} = \langle w_1, \ldots, w_k, v_{k+1} \cdot a + w_{k+1} \cdot (1 - a), \ldots, v_m \cdot a + w_m \cdot (1 - a) \rangle \in S$$

The only question to be answered yet is how to find the largest a to obtain the greatest possible information exchange: due to the real interval, we cannot perform an extensive search. In GENOCOP we implemented a binary search (to some depth only for efficiency). 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. Note, that the value of a is determined separately for each single arithmetical crossover and each gene. Single arithmetical crossover is defined as follows: if $s_v^t = \langle v_1, \ldots, v_m \rangle$ and $s_w^t = \langle w_1, \ldots, w_m \rangle$ are to be crossed, the resulting offspring are $s_v^{t+1} = \langle v_1, \ldots, v_k', \ldots, v_m \rangle$ and $s_w^{t+1} = \langle w_1, \ldots, w_k', \ldots, w_m \rangle$, where $k \in [1, m]$, $v_k' = a \cdot w_k + (1 - a) \cdot v_k$, and $w_k' = a \cdot v_k + (1 - a) \cdot w_k$. Here, a is a dynamic parameter calculated in the given context (vectors s_v, s_w) so that the operator is closed (points x_v^{t+1} and x_w^{t+1} are in the convex constrained space $\boldsymbol{S}$). Actually, a is a random choice from the following range:

$$a \in \begin{cases} [max(\frac{l_{f(k)} - w_k}{v_k - w_k}, \frac{u_{f(k)} - v_k}{w_k - v_k}), min(\frac{l_{f(k)} - v_k}{w_k - v_k}, \frac{u_{f(k)} - w_k}{v_k - w_k})] & \text{if } v_k > w_k \\ [0, 0] & \text{if } v_k = w_k \\ [max(\frac{l_{f(k)} - v_k}{w_k - v_k}, \frac{u_{f(k)} - w_k}{v_k - w_k}), min(\frac{l_{f(k)} - w_k}{v_k - w_k}, \frac{u_{f(k)} - v_k}{w_k - v_k})] & \text{if } v_k < w_k \end{cases}$$

To increase the applicability of this operator (to ensure that a will be non-zero, which actually always nullifies the results of the operator) it is wise to select the applicable gene as a random choice from the intersection of movable genes of both chromosomes. Note again, that the value of a is determined separately for each single arithmetical crossover and each gene.
Whole arithmetical crossover is defined as a linear combination of two vectors: if s_v^t and s_w^t are to be crossed, the resulting offspring are $s_v^{t+1} = a \cdot s_w^t + (1 - a) \cdot s_v^t$ and $s_w^{t+1} = a \cdot s_v^t + (1 - a) \cdot s_w^t$. This operator uses a simpler static system parameter $a \in [0..1]$, as it always guarantees closedness (according to characteristic (1) of this section).

The Transportation Problem

The transportation problem (see, for example, [31]) is a simple optimization problem with nontrivial constraints. A single commodity must be transported from a number of sources to a number of destinations. A destination can fulfill its demand from one or more sources. The objective is to determine the amount to be shipped from each source to each destination, so that the total transportation cost is minimized.

If the transportation cost on a given route is proportional to the number of units transported, we have a *linear transportation problem,* otherwise we have a *nonlinear transportation problem.* While linear problems can be solved by OR methods, the nonlinear case lacks a general solving methodology.

Assume there are n sources and k destinations, the amount of supply at source i is *sour(i),* the demand at destination j is *dest(j),* and the transportation cost is given as a function f_{ij} of x_{ij}—the amount transported from source i to destination j. The transportation problem is formulated as:

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

subject to

$$\sum_{j=1}^{k} x_{ij} \leq sour(i), \text{ for } i = 1, \ldots, n,$$

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

$$x_{ij} \geq 0, \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, k.$$

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. (If $f_{ij}(x_{ij}) = cost_{ij} \cdot x_{ij}$ for all i and j, the problem is linear.)

The above formulation requires that the total supply $\sum_{j=1}^{k} sour(i)$ must at least equal total demand $\sum_{i=1}^{n} dest(j)$. When the total supply equals the total demand, the problem is called a *balanced transportation problem.* In this case all the constraints are equations; that is,

$$\sum_{j=1}^{k} x_{ij} = sour(i), \text{ for } i = 1, \ldots, n,$$

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

If the $sour(i)$'s and $dest(j)$'s in a balanced linear transportation problem are all integers, any optimal solution is an integer matrix, *i.e.* all x_{ij} ($i = 1, \ldots, n$, $j = 1, \ldots, k$) are integer. Moreover, the number of positive integers among the x_{ij}'s is at most $k + n - 1$.

Example 1. Assume 3 sources and 4 destinations. The supply is:

$$sour(1) = 15, sour(2) = 25, sour(3) = 5.$$

The demand is:

$$dest(1) = 5, dest(2) = 15, dest(3) = 15, dest(4) = 10.$$

The total supply and demand equal 45.

For a linear example, unit transportation costs $cost_{ij}$ ($i \in \{1, 2, 3\}$, and $j \in \{1, 2, 3, 4\}$) are given in the table below.

Cost

10	0	20	11
12	7	9	20
0	14	16	18

The optimal solution for the above cost function is summarized below. The total cost is 315. For this case, the solution consists of integer values of x_{ij}.

Amount transported

	5	15	15	10
15	0	5	0	10
25	0	10	15	0
5	5	0	0	0

However, the optimal solution for a nonlinear transportation problem need not be integer. For example for some nonlinear transportation functions $f(x_{ij})$ of the flows x_{ij}, the optimal solution for the above problem may contain neither zeros nor integer values, although all constraints are satisfied:

Amount transported

An Example of the GENOCOP Approach

In this section, we present an example of the GENOCOP approach for solving a transportation problem. Using notation from previous sections, the transportation problem from the Example 1 can be formulated as follows:

$$\text{minimize } (10x_1 + 20x_3 + 11x_4 + 12x_5 + 7x_6 + 9x_7 + 20x_8 + 14x_{10} + 16x_{11} + 18x_{12}),$$

where (sources):

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= 15 \\ x_5 + x_6 + x_7 + x_8 &= 25 \\ x_9 + x_{10} + x_{11} + x_{12} &= 5 \ (*) \end{aligned}$$

and (destinations):

$$\begin{aligned} x_1 + x_5 + x_9 &= 5 \\ x_2 + x_6 + x_{10} &= 15 \\ x_3 + x_7 + x_{11} &= 15 \\ x_4 + x_8 + x_{12} &= 10 \end{aligned}$$

One of these seven equalities can be removed to obtain six independent equations—we eliminate the equality (*). We can therefore eliminate six variables forming vector x^1. The problem specific domain constraints are:

$$\begin{aligned} 0 \le x_1 &\le 5 \\ 0 \le x_2 &\le 15 \\ 0 \le x_3 &\le 15 \\ 0 \le x_4 &\le 10 \\ 0 \le x_5 &\le 5 \\ 0 \le x_{5\ \text{Tm (65)Tj /F5}} & \end{aligned} \qquad \begin{aligned} 0 \le x_7 &\le 15 \\ 0 \le x_8 &\le 10 \\ 0 \le x_9 &\le 5 \\ 0 \le x_{10} &\le 5 \\ 0 \le x_{11} &\le 5 \end{aligned}$$

(there are six equations, since one equation was eliminated),

$$A_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \text{ and } \qquad A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix},$$

$$\begin{aligned} &\vec{b} = \langle\, 15, 25, 5, 15, 15, 10 \,\rangle, \\ &\vec{x} = \langle\, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12} \,\rangle, \\ &\overrightarrow{x^1} = \langle\, x_1, x_2, x_3, x_4, x_5, x_9 \,\rangle, \\ &\overrightarrow{x^2} = \langle x_6, x_7, x_8, x_{10}, x_{11}, x_{12} \,\rangle, \\ &\vec{l} = \langle\, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle, \\ &\overrightarrow{l_1} = \langle\, 0, 0, 0, 0, 0, 0 \,\rangle, \\ &\overrightarrow{l_2} = \langle\, 0, 0, 0, 0, 0, 0 \,\rangle, \\ &\vec{u} = \langle\, 5, 15, 15, 10, 5, 15, 15, 10, 5, 5, 5, 5 \,\rangle, \\ &\overrightarrow{u_1} = \langle\, 5, 15, 15, 10, 5, 5 \,\rangle, \\ &\overrightarrow{u_2} = \langle\, 15, 15, 10, 5, 5, 5 \,\rangle, \end{aligned}$$

and $A\vec{x} = \vec{b}$, $\vec{l} \le \vec{x} \le \vec{u}$, $A_1\overrightarrow{x^1} + A_2\overrightarrow{x^2} = \vec{b}$, $\overrightarrow{l_1} \le \overrightarrow{x^1} \le \overrightarrow{u_1}$, $\overrightarrow{l_2} \le \overrightarrow{x^2} \le \overrightarrow{u_2}$.

The set of inequalities (aside from the domain constraints) is empty. It is an easy exercise to get:

$$\begin{aligned} x_1 &= x_6 + x_7 + x_8 + x_{10} + x_{11} + x_{12} - 25, \\ x_2 &= 15 - x_6 - x_{10} \\ x_3 &= 15 - x_7 - x_{11} \\ x_4 &= 10 - x_8 - x_{12} \\ x_5 &= 25 - x_6 - x_7 - x_8 \\ x_9 &= 5 - x_{10} - x_{11} - x_{12} \end{aligned}$$

i.e. $\overrightarrow{x^1} = A_1{}^{-1}\vec{b} - A_1{}^{-1}A_2\overrightarrow{x^2}$.
The new inequalities

$$A_1\overrightarrow{l_1} \le \vec{b} - A_2\overrightarrow{x^2} \le A_1\overrightarrow{u_1},$$

are

$$\begin{aligned} &0 \le x_6 + x_7 + x_8 + x_{10} + x_{11} + x_{12} - 25 \le 5, \\ &0 \le 15 - x_6 - x_{10} \le 15, \\ &0 \le 15 - x_7 - x_{11} \le 15, \\ &0 \le 10 - x_8 - x_{12} \le 10, \\ &0 \le 25 - x_6 - x_7 - x_8 \le 5, \\ &0 \le 5 - x_{10} - x_{11} - x_{12} \le 5, \end{aligned}$$

The above set together with the problem specific domain constraints

$$\begin{aligned} &0 \le x_6 \le 15 \\ &0 \le x_7 \le 15 \\ &0 \le x_8 \le 10 \\ &0 \le x_{10} \le 5 \\ &0 \le x_{11} \le 5 \\ &0 \le x_{12} \le 5 \end{aligned}$$

constitutes the full set of constraints for the new six variable problem. The variables are independent and there are now no equations connecting them.

At this stage the structure of an individual solution is fixed (vector $\vec{x^2}$). Using the initialization routine, a population of feasible solutions is created. These individuals undergo evolution; the genetic operators are closed in the space of feasible solutions and cause the system to converge. Having the best chromosome $\vec{x^2}$ found, we restore the original twelve variables: these constitute the solution vector.

An example of a possible uniform mutation follows.

Example 2. Assume, the following vector undergoes the uniform mutation:

$$\vec{g} = \langle\, 7.70,\ 14.39,\ 0.00,\ 0.79,\ 0.43,\ 3.03 \,\rangle,$$

and that the second component (corresponding to variable x_7) is selected for mutation. From the set of inequalities it follows that

$$0 \le x_6 + x_7 + x_8 + x_{10} + x_{11} + x_{12} - 25 \le 5,$$

which (for fixed $x_6 = 7.70$, $x_8 = 0.00$, $x_{10} = 0.79$, $x_{11} = 0.43$, $x_{12} = 3.03$) gives

$$13.05 \le x_7 \le 18.05.$$

Other inequalities containing x_7 do not impose new limitations; the problem specific domain constraints require that $x_7 \le 15$. So, this mutation operation would change the value of variable x_7 from 14.39 into a random value within the range [13.05, 15.0].

Comparison of the GA Approaches

In this section we compare the different genetic algorithm approaches (discussed earlier) for solving constrained optimization problems and our new methodology (system GENOCOP).

The Problem

To get some indication of the usefulness of the proposed approach, we have selected a single example of a 7×7 transportation problem from [27] (Figure 2) and compared results from different approaches.

	20	20	20	23	26	25	26
27	x_1	x_2	x_3	x_4	x_5	x_6	x_7
28	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
25	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}
20	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}
20	x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
20	x_{36}	x_{37}	x_{38}	x_{39}	x_{40}	x_{41}	x_{42}
20	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}

Figure 2. The 7 x 7 transportation problem.

So, the problem is to minimize a function

$$f(\vec{x}) = f(x_1, \ldots, x_{49}),$$

subject to fourteen (thirteen independent) equations:

$$\begin{aligned}
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 &= 27 \\
x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} &= 28 \\
x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} &= 25 \\
x_{22} + x_{23} + x_{24} + x_{25} + x_{26} + x_{27} + x_{28} &= 20 \\
x_{29} + x_{30} + x_{31} + x_{32} + x_{33} + x_{34} + x_{35} &= 20
\end{aligned}$$

$$x_{36} + x_{37} + x_{38} + x_{39} + x_{40} + x_{41} + x_{42} = 20$$
$$x_{43} + x_{44} + x_{45} + x_{46} + x_{47} + x_{48} + x_{49} = 20$$

$$x_1 + x_8 + x_{15} + x_{22} + x_{29} + x_{36} + x_{43} = 20$$
$$x_2 + x_9 + x_{16} + x_{23} + x_{30} + x_{37} + x_{44} = 20$$
$$x_3 + x_{10} + x_{17} + x_{24} + x_{31} + x_{38} + x_{45} = 20$$
$$x_4 + x_{11} + x_{18} + x_{25} + x_{32} + x_{39} + x_{46} = 23$$
$$x_5 + x_{12} + x_{19} + x_{26} + x_{33} + x_{40} + x_{47} = 26$$
$$x_6 + x_{13} + x_{20} + x_{27} + x_{34} + x_{41} + x_{48} = 25$$
$$x_7 + x_{14} + x_{21} + x_{28} + x_{35} + x_{42} + x_{49} = 26.$$

We made experiments with six nonlinear cost functions (for a full discussion on the selection and classification of these functions, see [23]); they are drawn in Appendix A.

- function

$$A(x) = \begin{cases} 0, \textit{ if } 0 < x \leq \boldsymbol{S} \\ c_{ij}, \textit{ if } \boldsymbol{S} < x \leq 2\boldsymbol{S} \\ 2c_{ij}, \textit{ if } 2\boldsymbol{S} < x \leq 3\boldsymbol{S} \\ 3c_{ij}, \textit{ if } 3\boldsymbol{S} < x \leq 4\boldsymbol{S} \\ 4c_{ij}, \textit{ if } 4\boldsymbol{S} < x \leq 5\boldsymbol{S} \\ 5c_{ij}, \textit{ if } 5\boldsymbol{S} < x, \end{cases}$$

where $\boldsymbol{S}$ is less than a typical x value.

- function B

$$B(x) = \begin{cases} c_{ij}\frac{x}{s}, \textit{ if } 0 \leq x \leq \boldsymbol{S} \\ c_{ij}, \textit{ if } \boldsymbol{S} < x \leq 2\boldsymbol{S} \\ c_{ij}(1 + \frac{x-2\boldsymbol{S}}{\boldsymbol{S}}), \textit{ if } 2\boldsymbol{S} < x, \end{cases}$$

where $\boldsymbol{S}$ is of the order of a typical x value.

- function C

$$C(x) = c_{ij}x^2$$

- function D

$$D(x) = c_{ij}\sqrt{x}$$

- function E

$$E(x) = c_{ij}\left(\frac{1}{1 + (x - 2\mathrm{S})^2} + \frac{1}{1 + (x - \frac{9}{4}\boldsymbol{S})^2} + \frac{1}{1 + (x - \frac{7}{4}\boldsymbol{S})^2}\right),$$

where $\boldsymbol{S}$ is of the order of a typical x value.

- function F

$$F(x) = c_{ij}x(\sin(x\frac{5\pi}{4S}) + 1),$$

where $\boldsymbol{S}$ is of the order of a typical x value.

The objective function for the transportation problem is of the form

$$\sum_{ij} f(x_{ij}),$$

where $f(x)$ is one of the functions above, the parameter c_{ij} is obtained from the parameter matrix (see Appendix B), and $\boldsymbol{S}$ is obtained from the attributes of the problem to be tested. To derive $\boldsymbol{S}$, it is necessary to estimate the value of a typical x value; this was done by the way of preliminary runs to estimate the number and magnitudes of non-zero x_{ij}'s. In this way the average flow on each arc was estimated and a value for $\boldsymbol{S}$ found. For function A we used $\boldsymbol{S} = 2$, while for B, E, and F, we used $\boldsymbol{S} = 5$.

Note that the objective function is identical on each arc, so a cost-matrix was used to provide a variation between arcs. The matrix provides the c_{ij}'s which act to scale the basic function shape, thus providing 'one degree' of variability.

Penalty Functions

The major issue in using penalty functions approach is assigning weights to the constraints: these weights play the role of penalties if a potential solution does not satisfy them. In experiments for the above transportation problem, the evaluation function, *Eval*, is composed of the optimization function f and the penalty P:

$$Eval(\vec{x}) = f(\vec{x}) + P,$$

For our experiments we have used a suggestion (see [26]) to start with relaxed penalties and to tight them as the run progresses. We used

$$P = k \cdot (\tfrac{t}{T})^p \cdot \bar{f} \cdot \textstyle\sum_{i=1}^{14} d_i\,,$$

where $\bar{f}$ is the average fitness of the population at the given generation t, k and p are parameters, T is the maximum number of generations, and d_i returns the "degree of constraint violation". Foe example, for a constraint:

$$\textstyle\sum_{i \in W} x_i = val, \;\; W \subseteq \{1, \ldots, 49\},$$

and a chromosome

$$\langle v_1, \ldots, v_{49} \rangle,$$

the penalty (degree of constraint violation) d_i is

$$d_i = \left| \textstyle\sum_{i \in W} v_i - val \right|.$$

We experimented with various values of p (close to 1), k (close to $\frac{1}{14}$, where 14 is total number of constraints), and $T = 8000$.

This method did not lead to feasible solutions: in over 1200 runs (with different seeds for random number generator and various values for parameters k and p) the best chromosomes (after 8000 generations) violated at least 3 constraints in a significant way. For example, very often the algorithm converged to the solution, where the numbers on one diagonal were equal to 20, and all others were zeros:

$$x_1 = x_9 = x_{17} = x_{25} = x_{33} = x_{41} = x_{49} = 20$$

and $x_i = 0$ for other i.

It is interesting to compare this "solution" with the arc parameter matrix given in Appendix B: this matrix has zeros on the diagonal which was the reason for the faulty convergence.

It should be clear that the methods based on penalty functions must fail for such a heavily constrained problem. The presence of fourteen constraints (which are equations) reduces the probability of locating a feasible solution; when all constraints are essential and must be satisfied, the chances for finding a feasible solution are less than slim.

Repair algorithms

Since the genetic operators often change a feasible solution of the transportation problem into a nonfeasible one, we may try to design appropriate repair algorithms for each operator to restore feasibility.

There are several problems to resolve. To simplify discussion, we will view a solution as a table. Assume that two random points (v_i and v_m, where $i < m$) are selected such that they do not belong to the same row or column. Let us assume that v_i, v_j, v_k, v_m ($i < j < k < m$) are components of a solution-vector (selected for mutation)

such that v_i and v_k, as well as v_j and v_m, belong to a single column, and v_i and v_j, as well as v_k and v_m, belong to a single row. That is, in matrix representation:

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

Assume that mutation would change the value of v_i. Further assume, that the new value for the v_i component of the solution vector is now $v_i + c$ ($c > 0$). The easiest repair algorithm would update the bits for the three other components of solution vector v_j, v_k, v_m, in the following way:

- $v_j' = v_j - c$,
- $v_k' = v_k - c$,
- $v_m' = v_m + c$.

However, it may happen that $v_j < c$, $v_k < c$, or that $v_m' > u_m$ (u_m is the maximum value from the domain of variable x_m). We could set $c = min(v_j, v_k, u_m - v_m)$, and such repair algorithm would work.

The situation is more complex if we try to repair a chromosome after applying crossover operator. Breaking a vector at a random point could result in a pair of chromosomes violating numerous constraints. If we try to modify these solutions to obey all constraints, they would lose most similarities with the parents. Moreover, the way to do this is far from obvious: if a vector $\vec{x}$ is outside the search space, "repairing" it might be as difficult as solving the original problem. Even if we succeeded in building a system based on repair algorithms, such system would be highly problem specific with little chances for generalizations.

For these reasons we have not developed any system based on this approach. However, we developed a system based on "repair algorithms" methodology using specialized data structures and operators.

Specialized Data Structures and Operators

In this section we discuss the applicability of the third approach for solving the constrained problem: the use of new data structures and operators. This approach—for transportation problem—is fully discussed in [33] and [23]: the resulting genetic system based on matrix representation was named GENETIC-2.

The representation: We select a two dimensional structure for representing a solution (a chromosome) to the transportation problem: a matrix $V = (x_{ij})$ ($1 \leq i \leq k, 1 \leq j \leq n$). This is the natural representation used when such problems are solved by hand.

The initialization: We require a way to create an individual solution which satisfies all constraints. Two types of initialization procedure have been defined. The first introduces as many zero entries into the matrix as possible. The second is modified to avoid choosing zero entries by selecting values from a range.

These procedures will be used to start the algorithm by generating enough initial solutions to fill the population. They can be used in combination, the proportion of each type chosen depending on the form, linear or nonlinear, of objective function to be optimized. They are also used as subprocedures in one of the mutation operators.

Procedure **initialization-1** (Figure 3) creates a matrix of at most $k + n - 1$ non-zero elements such that all constraints are satisfied. This is the method used for the linear case of the transportation problem (see [32, 33]). Although other 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. The solutions it generates are later modified during the evolution process by genetic operators.

input: arrays $dest(k)$, $sour(n)$;
output: an array $(x)_{ij}$ such that $x_{ij} \geq 0$ for all i and j,
$\sum_{j=1}^{k} x_{ij} = sour(i)$ 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-1;
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) $i = [(q - 1)/k + 1]$
    set (column) $j = (q - 1)$ *mod* $k + 1$
    set $val = min(sour(i), dest(j))$
    set $x_{ij} = val$
    set $sour(i) = sour(i) - val$
    set $dest(j) = dest(j) - val$
  until all numbers are visited.
end

Figure 3. The initialization-1 procedure.

The procedure **initialization-2** is changed from **initialization-1** first by changing the line:

set $val = min(sour(i), dest(j))$

by:

set $val_1 = min(sour(i), dest(j))$
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 $\langle 0, val_1 \rangle$

This change is desirable because it generates real numbers as solution elements in place of integers as in **initialization-1**. However, **initialization-2** must be further modified, since the resulting matrix may violate the set of constraints. So an additional (last) line must be added to the new **initialization-2** algorithm:
make necessary corrections.[1]

For full discussion on initialization algorithms for the transportation problem the reader is referred to [23].
The operators: We define two genetics operators, mutation and arithmetical crossover.
Mutation:
Assume that $\{i_1, \ldots, i_p\}$ is a subset of $\{1, \ldots, k\}$, and $\{j_1, \ldots, j_q\}$ is a subset of $\{1, \ldots, n\}$ such that $2 \leq p \leq k$, $2 \leq q \leq n$.
Denote a parent for mutation by $(k \times n)$ 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 $j \in \{j_1, \ldots, j_q\}$ (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 calculate 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 use either of the two initialization procedures to assign new values to the matrix W such that the constraint sums $sour_W(i)$ and $dest_W(j)$ remain satisfied. Then we replace appropriate elements of matrix V by the corresponding elements from the matrix W. In this way the global constraints ($sour(i)$ and $dest(j)$) are preserved.
The following example will illustrate the mutation operator (using the **initialization-1** procedure).

Example 3. Again, take the transportation problem from Example 1.

Assume that the following matrix V was selected as a parent for mutation:

Amount transported

	5.0	15.0	15.0	10.0
15.0	1.34	6.51	0.18	6.97
25.0	2.91	7.70	14.39	0.00
5.0	0.75	0.79	0.43	3.03

Let us assume that two rows {1, 3} and two columns {2, 4} were selected. The corresponding submatrix W is:

6.51	6.97
0.79	3.03

$sour_W(1) = 13.48$, $sour_W(2) = 3.82$, $dest_W(1) = 7.30$, $dest_W(2) = 10.00$. After the reinitialization of matrix W, it may have the following values:

3.48	10.00
3.82	0.00

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

Amount transported

	5.0	15.0	15.0	10.0
15.0	1.34	3.48	0.18	10.0
25.0	2.91	7.70	14.39	0.00
5.0	0.75	3.82	0.43	0.00

□

The proportion that each of the procedures **initialization-1** and **initialization-2** is used is controlled by one of the parameters of the system and depends on the nature of the transportation problem objective.

- **Arithmetical crossover**
  Starting with two parents (matrices U and V) the 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 $c_1 + c_2 = 1$). Since the constraint set is convex this operation ensures that both of children are feasible if both parents are.
  In the linear case of the transportation problem we used $c_1 = c_2 = 0.5$, and a special repair algorithm to eliminate fractions (see [33]).

Results

The results of various experiments of the system GENETIC-2 based on matrix representation are reported elsewhere (see [33] for linear case, and [23] for nonlinear case of the transportation problem). In this section we discuss the results of the system for a single problem presented earlier.

For each function (A–F) we have repeated 3 separate runs (with different seeds for random number generator) of 8,000 generations, and the best of those are reported in Figure 4, along with intermediate results at some generation intervals. For example, the values in column “1,000” indicate the partial result after 1,000 generations while running 8,000.

A single run of 8,000 generations took (on average) 2:28 CPU see on Cray Y-MP.

We postpone a discussion on the quality of these solutions after presenting the results from the GENOCOP system.

	Generations						
Function	1	100	500	1,000	2,000	4,000	8,000
A	1085.8	657.1	233.9	181.3	113.6	65.4	00.00
B	932.4	584.7	400.1	251.6	233.3	207.2	203.81
C	83079.1	24375.9	13875.5	3978.2	2799.8	2698.3	2564.23
D	1733.6	1209.8	601.4	578.4	480.4	480.2	480.16
E	225.2	221.1	205.0	204.9	204.9	204.8	204.73
F	3799.3	1674.1	1008.5	438.8	280.2	153.3	110.94

Figure 4. Progress of the GENETIC-2 for the 7×7 transportation problem.

GENOCOP System

In the selected problem, there are 13 independent and one dependent equality constraint sets; therefore, we eliminate thirteen variables: x_1, …, x_8, x_{15}, x_{22}, x_{29}, x_{36}, x_{44}. All remaining variables are renamed y_1, …, y_{36}, preserving order, *i.e.* $y_1 = x_9$, $y_2 = x_{10}$, …, $y_6 = x_{14}$, $y_7 = x_{16}$, …, $y_{36} = x_{49}$. Each chromosome is a float vector $\langle y_1, \ldots, y_{36} \rangle$. Each of these variables has to satisfy four two-sided inequalities resulting from original domains and our transformations. For example, for the first variable y_1 (which corresponds to the original x_9) we have:

$$0 \le y_1 \le 20$$
$$93 - \sum_{i=2}^{30} y_i + y_{31} \le y_1 \le 113 + y_{31} - \sum_{i=2}^{30} y_i$$
$$\sum_{i=31}^{36} y_i - 20 - y_7 - y_{13} - y_{19} - y_{25} \le y_1 \le \sum_{i=31}^{36} y_i - 20 - y_7 - y_{13} - y_{10} - y_{25}$$
$$8 - \sum_{i=2}^{6} \le y_1 \le 28 - \sum_{i=2}^{6}$$

The GENOCOP finds an optimal chromosome after a number of generations, and the equalities allow us to restore the original forty nine variables; these constitute the sought solution.

Again, for each function (A–F) we have repeated 3 separate runs of 8,000 generations; the best of those are reported in Figure 5, along with intermediate results at some generation intervals.

The average time for a single run of 8,000 generations was similar to that of GENETIC-2; however, times of computations in the GENOCOP system are not meaningful: this is due to the pilot implementation of the system for the 7×7 problem, which was partially hard-coded. The parameters used by GENOCOP in all runs are displayed in Figure 6.

Comparison

In testing an optimization algorithm on a linear problem it is possible to compare its solution with the known global optimum found using the standard linear programming technique—and, therefore, to determine how efficient the optimization algorithm is in absolute terms. For nonlinear objective functions the optimum may not be known. In such cases the results must be compared with those of other systems able to converge to a possibly good optimum.

	Generations						
Function	1	100	500	1,000	2,000	4,000	8,000
A	1085.8	856.5	273.4	230.5	153.0	94.5	24.15
B	932.4	595.8	410.6	258.4	250.3	209.2	205.60
C	83079.1	28947.2	14122.7	4139.0	2944.1	2772.7	2571.04
D	1733.6	1106.9	575.3	575.3	480.2	480.2	480.16
E	225.2	207.2	204.9	204.9	204.9	204.9	204.82
F	3799.3	1719.0	1190.1	550.8	320.9	166.4	119.61

Figure 5. Progress of the GENOCOP for the 7×7 transportation problem.

Parameter	Value
pop_size	40
Prob_mut$_{um}$	.08
prob_mut$_{bm}$	.03
prob_mut$_{nm}$	.07
prob_cross$_{sc}$	.10
prob_cross$_{sa}$	.10
prob_cross$_{wa}$	.10
a	.25
b	2.0
prop	0.5

Figure 6. Parameters used for the GENOCOP on the 7×7 transportation problem: population size (pop_size), probability of uniform mutation (prob_mut$_{um}$), probability of boundary mutation (prop_mut$_{bm}$), probability of non-uniform mutation (prob_mut$_{nm}$), probability of simple crossover (prob_cross$_{sc}$), probability of single arithmetical crossover (prob_cross$_{sa}$), probability of whole arithmetical crossover (prob_cross$_{wa}$), a coefficient for whole arithmetical crossover (a), a coefficient (b) for function $\triangle$ of the non-uniform mutation, and *prop* determines the proportion of initial potential solutions from the boundary of the solution space.

We have chosen to compare the GENOCOP algorithm with the genetic algorithm based on matrix representation GENETIC-2 and with the GAMS[2] system—a typical industry standard optimization package. We used the MINOS version of the optimizer; in the rest of the paper it is referred to briefly as GAMS.

For functions C, E, and F, the GAMS application was straightforward: using built-in nonlinear functions. Due to the requirement for gradient estimation of objective functions, GAMS could not handle functions A, B and D directly. In the case of A and B, the expression couldn't be formulated in GAMS while in the case of D (the square-root function) difficulty was encountered in measuring gradients near zero. Therefore, we made the following modifications to the problem for the GAMS runs:

- function A
  Separate arc-tangent functions are used to approximate each of the five steps. A parameter, P_A, was used to control the 'tightness' of the fit. The cost on arc [i,j] is:

$$c_{ij} \cdot \begin{pmatrix} arctan(P_A(x_{ij} - S))/\pi + \frac{1}{2} + \\ arctan(P_A(x_{ij} - 2S))/\pi + \frac{1}{2} + \\ arctan(P_A(x_{ij} - 3S))/\pi + \frac{1}{2} + \\ arctan(P_A(x_{ij} - 4S))/\pi + \frac{1}{2} + \\ arctan(P_A(x_{ij} - 5S))/\pi + \frac{1}{2} \end{pmatrix}$$

- function B
  The arc-tangent function was again used, this time to approximate each of the three gradients. A parameter, P_B, was used to control the tightness of the fit. The cost on arc [i,j] is:

$$c_{ij} \cdot \begin{pmatrix} (\frac{x_{ij}}{S}) \cdot (arctan(P_B x_{ij})/\pi + \frac{1}{2}) + \\ (1 - \frac{x_{ij}}{S}) \cdot (arctan(P_B(x_{ij} - S))/\pi + \frac{1}{2} + \\ (\frac{x_{ij}}{S} - 2) \cdot (arctan(P_B(x_{ij} - 2S))/\pi + \frac{1}{2}) \end{pmatrix}$$

- function D
  In order to avoid gradient problems at or near zero, the function D was changed to:

$$D'(x) = D(x + \epsilon).$$

For each problem multiple GAMS runs were made under different values of the parameter and the best result chosen. (The final objective function was calculated after the optimization based on the true function, rather than the modified function used within the optimization itself). GAMS was run on an Olivetti 386 with a math co-processor.

The comparative results of GAMS and GENOCOP are reported in Figure 7. In addition, all the best transportation flows for both systems and all six functions are given in Appendix C. It is clear that the proposed method performs much better than than the commercial package GAMS for most of the problems attempted. It is interesting to see that for practical cost functions (A & B) and for highly "irregular" cost function (F), the performance of GENOCOP is very good. For other functions (relatively smooth) the performance of GAMS and GENOCOP is quite similar.

We have also compared the GENOCOP system with GENETIC-2, which uses specialized data structures (see Figure 8). In general, their results are very similar. However, note that the matrix approach was tailored to the specific (transportation) problem, whereas GENOCOP is problem independent and works without any hard-coded domain knowledge. In other words, while one might expect the GENOCOP to perform similarly well for other constrained problems, the GENETIC-2 cannot be used at all.

While comparing all these three systems, it is important to underline that two of them, GAMS and GENOCOP, are problem independent: they are capable of optimizing any function subject to any set of linear constraints. The third system, GENETIC-2, was designed for transportation problems only: the particular constraints are incorporated into matrix data structures and special "genetic" operators. Comparing GENOCOP and GAMS we should also stress the fact that the GENOCOP system allows the user to provide the desirable number of iterations which influence the precision of the result. Also, any time during the execution of the program the user can request a current solution which always obeys the constraints. On the other hand, the GAMS is "all or nothing" proposition. An additional difference is that the GENOCOP's user can declare any function for optimization, as a C procedure. An example declaration (for function A) may look like:

Function	GAMS	GENOCOP	% difference
A	96.00	24.15	+ 297.52%
B	1141.60	205.60	+ 455.25%
C	2535.29	2571.04	− 1.41%
D	565.15	480.16	+ 17.70%
E	208.25	204.82	+ 1.67%
F	43527.54	119.61	+ 36291.22%

Figure 7. GENEOCOP versus GAMS: the results for the 7×7 problem, with cost matrix of Appendix B.

Function	GENETIC-2	GENOCOP	% difference
A	00.00	24.15	
B	203.81	205.60	−0.87%
C	2564.23	2571.04	−0.26%
D	480.16	480.16	0.00%
E	204.73	204.82	−0.04%
F	110.94	119.61	−7.24%

Figure 8. GENETIC-2 versus GENOCOP: the results for the 7×7 problem, with cost matrix of Appendix B.

```
sum = 0.0;
    for (i = 0; i < k; + + i)
        for (j = 0; j < n; + + j){
            if ((x[i][j] ≥ 0.0) && (x[i][j] ≤ 2.0)
                sum = sum; else
            if ((x[i][j] > 2.0) && (x[i][j] ≤ 4.0))
                sum = sum + cost[i][j] * x[i][j]/5.0; else
            if ((x[i][j] > 4.0) && (x[i][j] ≤ 6.0))
                sum = sum + cost[i][j] * x[i][j]/5.0; else
            if ((x[i][j] > 6.0) && (x[i][j] ≤ 8.0))
                sum = sum + cost[i][j] * x[i][j]/5.0; else
            if ((x[i][j] > 8.0) && (x[i][j] ≤ 10.0))
                sum = sum + cost[i][j] * x[i][j]/5.0; else
            if ((x[i][j] > 10.0)
                sum = sum + cost[i][j] * x[i][j]/5.0;
    }
```

Note that the implemented function is not continuous. On the other hand, in GAMS the user has to approximate such noncontinuous function by a continuous one, as we did and discussed earlier in this section.

The better results of GENOCOP against GAMS are mainly due to the fact that GENOCOP does not make any assumptions on the characteristic of the optimized function. Because of that, GENOCOP avoids some local optima which are reported as final solution in the other system.

It also seems that the set of parameters we used for our experiments in GENOCOP should give satisfactory performance for many types of functions (in all experiments for all functions A–F the set of parameters was invariant; we were changing only the seed for random number generator). However, some further experiments are required, and it might be desirable to add a parameter tunning module to the GENOCOP system.

Conclusions

After considering alternative ways for handling constraints in genetic algorithms for optimization problems, we propose a new method for handling linear constraints. This new methodology should enable such constrained problems with difficult objective functions to be solved without incurring the heavy computational overhead associated with frequent constraint checking and without a need for sesigning a specific system's architecture.

The equality constraints are handled immediately by eliminating some variables, at one stroke removing constraints and reducing the search space. The inequalities are then processed to provide a set of bounds for each of the remaining variables considered in isolation. These bounds are dynamic in that they depend on the values of other variables of the current solution. Since the GA modifies each variable independently, this is not computationally complex.

Our results suggest that the method is useful as compared to the standard methods, and may lead towards the solution of some difficult Operations Research problems. We have performed a number of experiments using this technique on one type of highly constrained problems with rather difficult objective functions. These experiments suggest that, certainly in comparison with an excellent standard 'hill-climbing' algorithm, the GA results are better in most cases.

While the GA methods are likely to be slower in execution than traditional methods when operating on well-behaved objective functions, moving to difficult problems compensates much better approximations for the resources sacrifice. This is reasonable and acceptable if real-time performance is not crucial, as in many cases of practical optimization problems.

It is relatively easy to extend the GENOCOP system to handle discrete variables nominal, linear, and boolean). Such variables would undergo different mutations and crossovers, to keep the solution vector within the constrained space.

Acknowledgments: The authors wish to thank Lindsay Groves for his comments on the first draft of the paper, Tony Vignaux for simplification of some notation, and Matthew Hobbs for experiments with GAMS and preparation of Appendix A.
This research was supported by a grant from the North Carolina Supercomputing Center.

References

1. Antonisse, J. *A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint.* In [29].
2. Brooke, A., Kendrick, D., and Meeraus, A. *GAMS: A User's Guide.* The Scientific Press, 1988.
3. Bosworth, J., Foo, N., and Zeigler, B.P. *Comparison of Genetic Algorithms with Conjugate Gradient Methods* NASA (CR-2093),Washington, D.C., 1972.
4. Davis, L., (Ed.) *Genetic Algorithms and Simulated Annealing* Pitman, London, 1987.
5. De Jong, K.A. *Genetic Algorithms: A 10 Year Perspective* In [9], pp. 169–177.
6. Dhar, V. and Ranganathan, N. Integer programming vs. expert systems: An experimental comparison *Commun ACM 33,* 3 (1990).
7. Goldberg, D.E. Dynamic system control using rule learning and genetics algorithms In *Proceedings of the International Joint Conference on Artificial Intelligence,* 1985.
8. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning* Addison Wesley, 1989.
9. Grefenstette, J.J. *Proceedings of the First International Conference on Genetic Algorithms.* (Pittsburg, July 1985) Lawrence Erlbaum.
10. Grefenstette, J.J. Optimization of control parameters for genetic algorithms. *IEEE Trans Syst. Man, and Cybernetics, SMC-16,* 1. 1986.
11. Grefenstette, J.J. *Incorporating Problem Specific Knowledge into Genetic Algorithms.* In [4].
12. Grefenstette, J.J. *Proceedings of the Second International Conference on Genetic Algorithms.* (Cambridge, July 1987). Lawrence Erlbaum.
13. 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.* (Knoxville, 1990).
14. Holland, J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Mich., 1975.
15. Holland, J. *Escaping Brittleness, in Machine Learning II.* R. Michalski, J. Carbonell, T. Mitchel, Morgan Kaufmann, Los Altos, Calif., 1986.
16. Janikow, C., and Michalewicz, Z. Specialized genetic algorithms for numerical optimization problems. *Proceedings of the International Conference on Tools for AI.* (Washington, Nov. 1990).
17. Janikow, C., and Michalewicz, Z. On the convergence problem in genetic algorithms. *Fundamenta Informaticae* (1991).
18. Leler, W. *Constraint Programming Languages: Their Specifications and Generation.* Addison Wesley, 1988.
19. Michalewicz, Z., Vignaux, G.A., Groves, L. Genetic algorithms for optimization problems. *Proceedings of the 11th NZ Computer Conference.* (Wellington, New Zealand, Aug 1989).
20. Michalewicz, Z. *EVA Programming Environment.* UNCC Technical Report. 1990.
21. Michalewicz, Z. and Janikow, C. Genetic algorithms for numerical optimization. *Statistics and Computing, 1,* 2 (1991).
22. Michalewicz, Z. and Schell J. *Data Structures + Genetic Operators = Evolution Programs.* UNCC Technical Report. 1990.
23. Michalewicz, Z., Vignaux, G.A., Hobbs, M. A genetic algorithm for the nonlinear transportation problem. *ORSA Journal on Computing,* 1991.
24. Michalewicz, Z., Jankowski, A., Vignaux, G.A. *The Constraints Problem in Genetic Algorithms,* in *Methodologies of Intelligent Systems: Selected Papers.* M.L. Emrich, M.S. Phifer, B. Huber, M. Zemankova, Z. Ras (Eds), 1990.
25. Michalewicz, Z., Krawczyk, J., Kazemi, M, Janikow, C. Genetic algorithms and optimal control problems. *Proceedings of the 29th IEEE Conference on Decision and Control.* (Honolulu, Dec. 1990).
26. Richardson, J.T., Palmer, M.R., Liepins, G., and Hilliard, M. *Some Guidelines for Genetic Algorithms with Penalty Functions,* in [29].

27. Sasieni, M., Yaspan, A., and Friedman, L. *Operations Research Methods and Problems.* John Wiley and Sons, 1959.
28. Schaffer, J., Caruana, R., Eshelman, L., and Das, R. *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization.* In [29].
29. Schaffer, J. *Proceedings of the Third International Conference on Genetic Algorithms.* (George Mason Univ. June 1989). Morgan Kaufmann.
30. Siedlecki, W. and Sklanski, J. *Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and Its Use in Pattern Recognition.* In [29].
31. Taha, H.A. *Operations Research: An Introduction,* 4th ed. Collier Macmillan, London, 1987.
32. Vignaux, G.A. and Michalewicz, Z. Genetic algorithms for the transportation problem. *Proceedings of the 5th International Symposium on Methodologies for Intelligent Systems.* (Charlotte, N.C. Oct. 1989).
33. Vignaux, G.A. and Michalewicz, Z. A genetic algorithm for the linear transportation problem. *IEEE Trans. Syst., Man, and Cybernetics 21,* 2. 1991.

Zbigniew Michalewicz (zbyszek@uncc.edu) is a professor in the computer science department of the University of North Carolina, in Charlotte, N.C.
Cezary Janikow (janikow@radom.umsl.edu) is an assistant professor in the department of math and computer science at the University of Missouri at St. Louis.

[1]This step is straightforward: a linear-time algorithm increases only few values in the matrix to satisfy all constraints.
[2]the *student version* of GAMS, a package for the construction and solution of large and complex mathematical programming models ([2]).

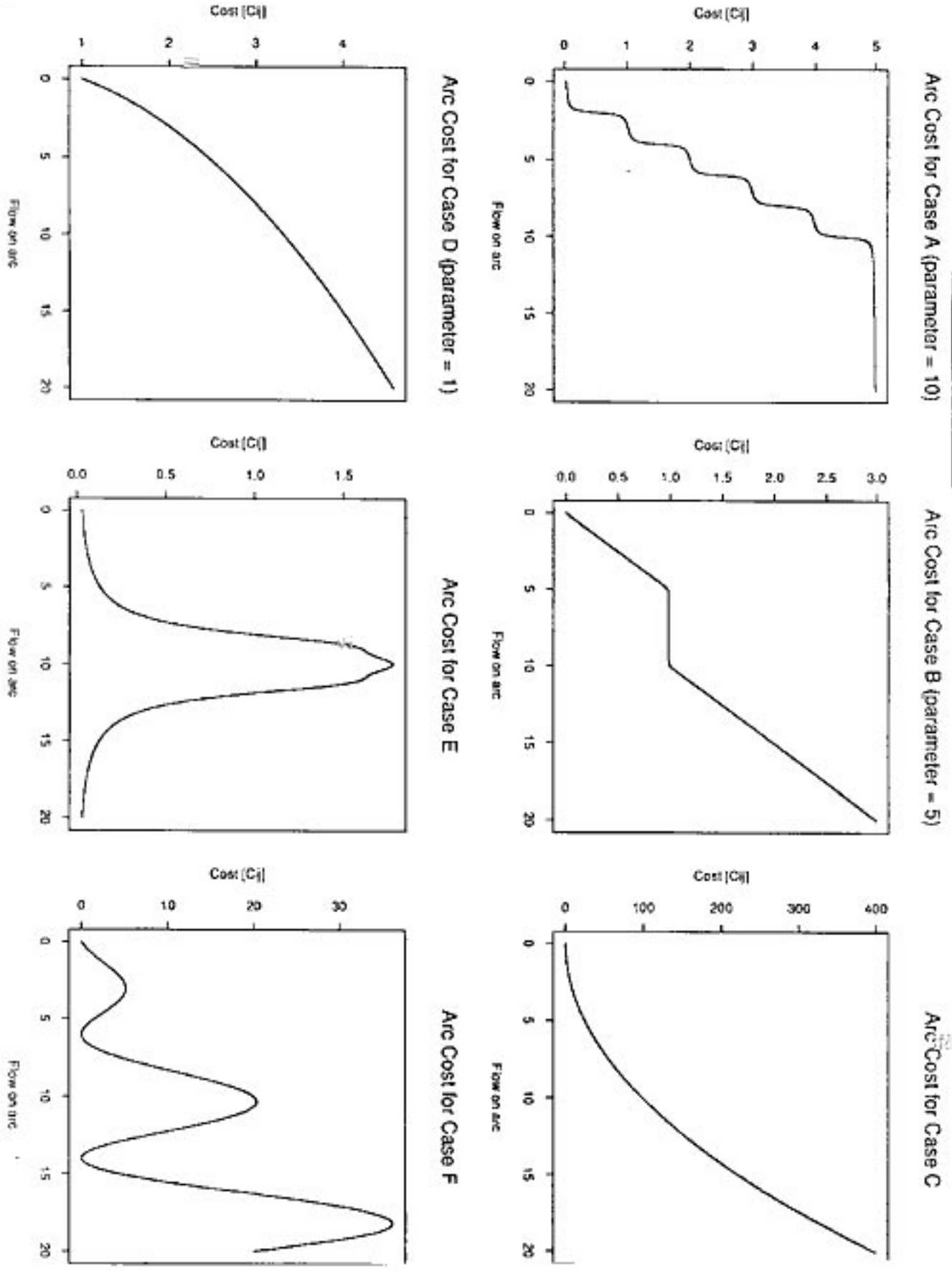
Appendix B. Example Problem Description

Number of Sources:			7				
Number of Destinations:			7				
Source Flows:	27	28	25	20	20	20	20
Destination Flows:	20	20	20	23	26	25	26

Arc Parameter Matrix (Source by Destination):

0	21	50	62	93	77	1000
21	0	17	54	67	1000	48
50	17	0	60	98	67	25
62	54	60	0	27	1000	38
93	67	98	27	0	47	42
77	1000	67	1000	47	0	35
1000	48	25	38	42	35	0

Appendix A. The Arc Cost Functions



Appendix C. Transportation flows

GENOCOP: function A Cost = 24.15

20.00	0.00	0.00	1.92	1.62	1.47	1.97
0.00	20.00	2.87	1.76	1.46	1.89	0.00
0.00	0.00	17.12	1.90	1.99	1.10	2.87
0.00	0.00	0.00	16.25	0.85	1.37	1.51
0.00	0.00	0.00	0.00	19.65	0.00	0.34
0.00	0.00	0.00	0.43	0.41	19.15	0.00
0.00	0.00	0.00	0.71	0.00	0.00	19.28

GENOCOP: function B Cost = 205.60

20.00	0.00	0.00	0.00	0.00	7.00	0.00
0.00	20.00	6.00	2.00	0.00	0.00	0.00
0.00	0.00	14.00	0.00	0.00	0.00	11.00
0.00	0.00	0.00	20.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	2.00	18.00	0.00
0.00	0.00	0.00	1.00	4.00	0.00	15.00

GENOCOP: function C Cost = 2571.04

20.00	0.00	1.29	1.86	1.58	2.11	0.14
0.00	20.00	1.42	1.90	2.02	0.07	2.57
0.00	0.00	17.28	1.13	1.13	1.72	3.72
0.00	0.00	0.00	18.09	1.10	0.00	0.79
0.00	0.00	0.00	0.00	19.55	0.44	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.60	0.63	18.75

GENOCOP: function D Cost = 480.16

20.00	7.00	0.00	0.00	0.00	0.00	0.00
0.00	13.00	15.00	0.00	0.00	0.00	0.00
0.00	0.00	5.00	0.00	0.00	0.00	20.00
0.00	0.00	0.00	20.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	3.00	6.00	5.00	6.00

GENOCOP: function E Cost = 204.82

0.56	0.00	0.00	0.43	0.00	0.00	26.00
0.00	0.25	0.00	2.00	0.75	25.00	0.00
0.00	0.00	0.00	0.00	25.00	0.00	0.00
19.43	0.00	0.00	0.56	0.00	0.00	0.00
0.00	19.75	0.00	0.00	0.25	0.00	0.00
0.00	0.00	20.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	20.00	0.00	0.00	0.00

GENOCOP: function F Cost = 119.61

8.30	6.30	6.38	0.00	6.00	0.00	0.00
0.00	13.69	0.30	14.00	0.00	0.00	0.00
6.00	0.00	13.30	0.00	0.00	0.00	5.69
5.69	0.00	0.00	9.00	0.00	0.00	5.30
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	5.00	15.00

GAMS: function A Cost = 96.00

20.00	1.28	0.94	1.58	1.52	1.58	0.08
0.00	18.71	0.38	1.58	1.57	0.12	5.61
0.00	0.00	18.66	1.55	1.47	1.58	1.72
0.00	0.00	0.00	18.27	1.25	0.00	0.47
0.00	0.00	0.00	0.00	19.47	0.52	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.71	1.18	18.10

GAMS: function B Cost = 1141.60

20.00	0.00	0.00	0.00	0.00	0.00	7.00
0.00	18.99	0.00	0.00	9.01	0.00	0.00
0.00	1.00	20.00	0.00	0.00	0.00	3.99
0.00	0.00	0.00	20.00	0.00	0.00	0.00
0.00	0.00	0.00	3.00	16.99	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	4.99	15.00

GAMS: function C Cost = 2535.29

20.00	0.52	0.85	1.82	1.58	2.07	0.13
0.00	19.47	1.85	1.89	2.03	0.14	2.58
0.00	0.00	17.29	1.17	1.07	1.75	3.70
0.00	0.00	0.00	18.10	1.27	0.04	0.57
0.00	0.00	0.00	0.00	19.73	0.26	0.00
0.00	0.00	0.00	0.00	0.00	0.00	20.00
0.00	0.00	0.00	0.00	0.29	0.70	19.99

GAMS: function D Cost = 565.15

20.00	2.00	0.00	0.00	0.00	5.00	0.00
0.00	18.00	4.00	0.00	6.00	0.00	0.00
0.00	0.00	16.00	3.00	0.00	0.00	6.00
0.00	0.00	0.00	20.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	20.00

GAMS: function E Cost = 208.25

0.00	0.00	0.00	0.00	1.00	0.00	26.00
0.00	0.00	0.45	0.00	2.54	25.00	0.00
0.00	0.00	0.77	23.00	1.22	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	18.77	0.00	1.22	0.00	0.00
0.00	20.00	0.00	0.00	0.00	0.00	0.00
20.00	0.00	0.00	0.00	0.00	0.00	0.00

GAMS: function F Cost = 43527.54

0.00	0.00	0.00	22.49	0.71	3.79	0.00
0.00	0.00	0.00	0.50	5.28	0.00	22.20
0.00	0.00	0.00	0.00	0.00	21.20	3.79
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	20.00	0.00	0.00	0.00	0.00
0.00	20.00	0.00	0.00	0.00	0.00	0.00
20.00	0.00	0.00	0.00	0.00	0.00	0.00

```
procedure genetic algorithm
begin
   t = 0
   initialize P(t)
   evaluate P(t)
   while (not termination-condition) do
   begin
      t = t + 1
      select P(t) from P(t 2 1)
      recombine P(t)
      evaluate P(t)
   end
end
```

Figure 1. A simple genetic algorithm.

	20	20	20	23	26	25	26
27	x_1	x_2	x_3	x_4	x_5	x_6	x_7
28	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
25	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}
20	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}
20	x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
20	x_{36}	x_{37}	x_{38}	x_{39}	x_{40}	x_{41}	x_{42}
20	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}

Figure 2. The 7 x 7 transportation problem.

```
input: arrays dest(k), sour(n);
output: an array (z)ij such that xij ≥ 0 for all i and j,
∑(j=1..k) xij = sour(i) for i = 1, 2, …, n, and ∑(i=1..n) xij = dest(j)
for j = 1, 2, …, k,
i.e. all constraints are satisfied.
procedure initialization-1;
begin
   set all numbers from 1 to k · n as unvisited
   repeat
      select an unvisited random number q from 1 to k · n
        and set it as visited
      set (row) i = [(q − 1)/k + 1]
      set (colun) j = (q − 1) mod k + 1
      set val = min(sour(i), dest(j))
      set xij = val
      set sour(i) = sour(i) − val
      set dest(j) = dest(j) − val
   until all numbers are visited.
end
```

Figure 3. The initialization-1 procedure.

	Generations						
Function	1	100	500	1,000	2,000	4,000	8,000