Constraint handling in genetic algorithms using a gradient-based repair method

Piya Chootinan, Anthony Chen*

Department of Civil and Environmental Engineering, Utah State University, Logan, UT 84322-4110, USA

Available online 14 March 2005

Abstract

Constraint handling is one of the major concerns when applying genetic algorithms (GAs) to solve constrained optimization problems. This paper proposes to use the gradient information derived from the constraint set to systematically repair infeasible solutions. The proposed repair procedure is embedded into a simple GA as a special operator. Experiments using 11 benchmark problems are presented and compared with the best known solutions reported in the literature. Our results are competitive, if not better, compared to the results reported using the homomorphous mapping method, the stochastic ranking method, and the self-adaptive fitness formulation method.

Keywords: Constraint handling; Constrained optimization; Genetic algorithms; Hybrid method

1. Introduction

Genetic algorithms (GAs) are one of the evolutionary computing techniques, which have been widely used to solve complex optimization problems that are known to be difficult for the traditional optimization techniques (see [1,2] for a detailed description of the GA). These traditional calculus-based optimization techniques generally require the problem to possess certain mathematical properties, such as continuity, differentiability, convexity, etc., which may not be satisfied in many real-world problems. As such, the GA, which does not require these properties, has been considered and often adopted as a practical optimization tool in many disciplines. Although GAs perform well for unconstrained or simple constrained optimization problems (e.g., box constraint, spherical constraint, etc.), they may encounter difficulties

* Corresponding author.

E-mail address: achen@cc.usu.edu (A. Chen).
when applied to solving highly constrained problems. This is because the traditional search operators of GAs (i.e., crossover and mutation) are blind to constraints. In such a circumstance, it is very likely that the candidate solutions generated by these operators during the search process would violate certain constraints. A special and proper treatment of constraints is often required to maintain the applicability of GAs and to prevent the genetic search from leaving the feasible region.

Thus far, the most commonly used constraint-handling methods for GAs are the penalty and repair methods (see [3,4] for a comprehensive survey on constraint-handling techniques used with GAs). The penalty method converts the constrained optimization problem to an unconstrained one by augmenting the constraints to the objective function as a penalty term. When the solution is infeasible, the objective value is penalized by a pre-determined penalty or more sophisticatedly by the degree of constraint violation. This method allows keeping some genetic information from the infeasible solutions with the hope that they will contribute to the improvement of solutions later. The difficulty of using this method is the selection of an appropriate penalty value for the problem at hand. It has been shown that if the fixed penalty value is chosen too small or too large, the solution time can increase considerably and, in some cases, the solution quality can be very much dependent on the penalty value. To overcome this drawback, some researchers have suggested replacing the constant penalty by a variable penalty sequence (e.g., dynamic scheme [5] or self-adaptive scheme [6–9]). However, these schemes usually end up with the difficulty of selecting another set of parameters [3,4]. In contrast, the repair method attempts to fix infeasible solutions by taking advantage of problem’s characteristics. The repair method might be very effective if the relationship between decision variables and constraints could be easily characterized. However, developing a repair procedure is usually problem-dependent and time-consuming when the problem is involved with complex constraints.

This study proposes a repair method using the gradient information derived from the constraint set. The method is used to repair a certain number of infeasible solutions while the GA performs the stochastic search for the better solutions. The organization of this paper is as follows. Section 2 briefly discusses the constraint-handling techniques currently used. The framework of the proposed constraint-handling is detailed in Section 3. Section 4 compares the results obtained in this study against the optimal or best-known solutions of 11 benchmark problems reported in the literature. Finally, conclusions drawn from the experimental results are provided in Section 5.

2. Current practice on constraint handling in genetic algorithms

A general constrained optimization problem can be expressed as follows:

\[
\text{Optimize } f(x),
\]

S.t.

\[
el_m \leq g_m(x) \leq u_m, \quad m = 1, \ldots, M,
\]

\[
h_n(x) = c_n, \quad n = 1, \ldots, N,
\]

where \(x\) represents the solution vector, \(f(x)\) is the objective function of the problem, while \(g_m(x)\) and \(h_n(x)\) are the inequality and equality constraints, respectively, that define the feasible region. In the application of GAs to any constrained optimization problems, handling the constraints is one of the difficult tasks. Currently, several techniques have been proposed to overcome this obstacle. According to Coello [3],
and Koziel and Michaelwicz [4], existing techniques can be classified to methods based on the penalty function [5–9], methods based on preserving feasibility of solutions [4,10], and hybrid methods [11–13].

The penalty method is perhaps the most commonly used technique. It basically transforms the constrained problem to an unconstrained one by augmenting the constraints to the objective function as a penalty term. When the solution is infeasible, its objective value is penalized according to the degree of constraint violations. Generally, the penalized objective value is computed using the expression:

\[ \tilde{f}(x) = f(x) \pm p(x), \]  

(4)

where \( p(x) \) is the penalty function representing the degree of constraint violation computed as

\[ p(x) = \sum_{m=1}^{M} w^u_m \cdot \max\{0,0, g_m(x) - u_m\} - \sum_{m=1}^{M} w^\ell_m \cdot \min\{0,0, g_m(x) - l_m\} \]

\[ + \sum_{n=1}^{N} w_n \cdot |h_n(x) - c_n|, \]  

(5)

where \( w^u_m \) and \( w^\ell_m \) are the penalty values of the upper and lower bounds of the inequality constraints, and \( w_n \) are the penalty values for the equality constraints. For minimization problems, if the solution is infeasible, the objective value is increased by the penalty term. For maximization problems, the penalty term is used to decrease the objective value. In other words, a highly infeasible solution would be penalized and would rarely be selected by the reproduction scheme.

The major concern of this method is how to choose a proper penalty value \((w)\) for each constraint so as to efficiently guide the search toward a promising area of the search space. A large penalty value will lead to premature convergence (i.e., trade off too much optimality for feasibility), while a small penalty will not only increase computational time, but also admit too many infeasible solutions (i.e., not enough pressure given to the penalty term to maintain feasibility). To overcome this difficulty, several researchers have suggested replacing the constant penalty value by a variable penalty sequence (e.g., a dynamic scheme [5] or self-adaptive scheme [6–9]). The magnitude of the penalty could be adapted according to the degree of constraint violation, the overall performance of genetic search, or the growth of evolutionary process (number of generations). Instead of penalizing the infeasible solutions directly, Runarsson and Yao [8] determined the fitness of solutions by stochastically comparing a pair of adjacent solutions based on either the objective value or the degree of constraint violation. Whether the objective value or the degree of constraint violation is used for comparison depends on the pre-specified probability (i.e., model parameter). This comparison process (i.e., sorting) is repeated a certain number of times until there is no change in the order of solutions. The number of times a solution wins the comparison (i.e., rank of solution) determined the fitness value. Although these variable penalty methods have demonstrated to work well for several problems, they require proper selection of another set of parameters [3]. Among these techniques, the self-adaptive fitness formulation proposed by Farmani and Wright [9] is an exception (i.e., it does not require any extra parameters). The penalty determined using a two-stage process ensures that the slightly infeasible solutions with better objective values remain fit and have a chance for reproduction. The ranges of the objective value and degree of constraint violation (e.g., the best and the worst cases) are simultaneously considered when adjusting the penalty value adaptively at each generation.

The repair method aims to preserve the feasibility of solutions. It involves fixing infeasible solutions through a specific repair procedure designed for the problem at hand. If the problem has only simple
constraints such as non-negativity, simple bounds, and/or conservation (i.e., linear equality), the infeasible solutions can easily be repaired. For examples, solutions with negative values can simply be reset to zero to preserve the non-negative constraints, solutions outside of the boundaries can also be projected back to the nearest boundary, and solutions not satisfying the conservation constraint can simply be normalized (i.e., redistributed according to the coefficients of linear equation). However, real-world problems may have other types of constraint. Repairing infeasible solutions sometimes can be as complex as solving the original problem. Prior knowledge of the problem is required in order to construct an efficient repair procedure. However, the characteristics of the solution space for real-world problems are often unknown; hence, constructing the repair procedure is far from trivial. Alternatively, the feasibility of solutions can be preserved through specially designed GA operators, which are also dependent on the characteristics of the problem. Koziel and Michaelwicz [4] proposed a homomorphous mapping to facilitate generating and preserving feasible solutions. The basic idea is to transform the original feasible region of the problem to an $i$-dimensional cube ($i$ decision variables) in which the feasibility of solutions can be easily maintained.

Chu and Beasley [11] incorporated several heuristics into their GA operators to solve the highly constrained set-partitioning problem. One heuristic example is the mating restriction in which two infeasible solutions are allowed to mate only if they are sufficiently compatible. Compatibility is defined by the number of constraints violated by the two infeasible solutions to determine a matching score. A pair of infeasible solutions is considered more compatible if one infeasible solution satisfies a certain number of constraints and the other infeasible solution satisfies the rest. Although the heuristics do not completely repair the infeasible solution, they indeed encourage the search to move toward the feasible region. Moreover, as mentioned by Chu and Beasley [11], the heuristics can be generalized and applied to any constrained optimization problems, not limited to the combinatorial problems as studied in [11].

Hybrid methods involve combining a GA with other techniques, mostly from numerical optimization [12,13] and artificial intelligence [14]. Kim and Myung [12] proposed using a two-stage evolutionary programming (EP) scheme in which the feasible (if possible) or near-feasible solutions are collected in the first stage, and those solutions are constrained to fulfill both optimality and feasibility by an augmented Lagrangian function in the second stage. Similarly, Adeli and Cheng [13] replaced the penalty term in the penalty method by the augmented Lagrangian function. In both studies, the penalty-like augmented Lagrangian function is adjusted according to the feasibility of solutions in each generation. The performance of hybrid methods is generally encouraging.

Recently, attention has been given to multi-objective solution approaches as another means for GAs to handle constraints [3,15,16]. The feasibility of solutions is treated as another objective of the problem, the modified problem becomes unconstrained, and several well-developed techniques for solving multi-objective optimization problems can be readily applied. This approach aims to overcome the drawback of the penalty method. Instead of penalizing the infeasible solutions, the fitness of each solution is evaluated according to both optimality and feasibility. In fact, one can view the penalty method as a special case of the multi-objective approach using a weighted-sum scheme to collectively evaluate multiple objectives. Preliminary results using the multi-objective approach show a significant improvement on GA solutions compared to the methods previously used for the same problems.

Coello [3] also emphasized in his study that a desirable constraint-handling technique should be as general as possible. In other words, it should require only minor modifications or fine-tuning to work with different problems. Besides, the incorporation of prior knowledge about the problem is worthwhile to consider in a complex constrained optimization problem.
3. Proposed constraint-handling method

3.1. Proposed repair method

The main idea of the proposed method is to utilize the gradient information derived from the constraint set to systematically repair the infeasible solutions. Basically, the gradient information is used to direct the infeasible solutions toward the feasible region. In general, the gradient could be derived directly from the constraints of which an explicit expression exists or could be approximated by an inexact method such as the finite difference [17] in the case that the constraints are very complicated to analytically evaluate or they are not available in explicit functional form (e.g., the constraint needs to be evaluated through a computer simulation or a solution of another optimization problem). Let \( \mathbf{V} \) consist of vectors of inequality constraints (\( \mathbf{g} \)) and equality constraints (\( \mathbf{h} \)) for the problem. The derivatives of these constraints with respect to the solution vector (\( I \) decision variables), \( \nabla_{\mathbf{x}} \mathbf{V} \), can be evaluated by

\[
\mathbf{V} = \begin{bmatrix} \mathbf{g}_{M \times 1} \\ \mathbf{h}_{N \times 1} \end{bmatrix}_{(M+N)\times 1} \quad \Rightarrow \quad \nabla_{\mathbf{x}} \mathbf{V} = \begin{bmatrix} \nabla_{\mathbf{x}} \mathbf{g} \\ \nabla_{\mathbf{x}} \mathbf{h} \end{bmatrix}_{(M+N)\times I},
\]

or approximated by the forward difference formulae [17].

\[
\nabla_{\mathbf{x}} \mathbf{V} = \frac{1}{e} \begin{bmatrix} \mathbf{g}(\mathbf{x}|x_i = x_i + e) - \mathbf{g}(\mathbf{x}), \forall i = 1, \ldots, I \\ \mathbf{h}(\mathbf{x}|x_i = x_i + e) - \mathbf{h}(\mathbf{x}), \forall i = 1, \ldots, I \end{bmatrix}_{(M+N)\times I},
\]

where \( e \) is a small positive scalar for perturbation. Therefore, the relationship between changes of constraint violation (\( \Delta \mathbf{V} \)) and solution vector (\( \Delta \mathbf{x} \)) are determined by

\[
\Delta \mathbf{V} = \nabla_{\mathbf{x}} \mathbf{V} \times \Delta \mathbf{x} \quad \Rightarrow \quad \Delta \mathbf{x} = \nabla_{\mathbf{x}} \mathbf{V}^{-1} \times \Delta \mathbf{V}.
\]

From the expression given above, the first derivatives of constraints with respect to the solution vector provide a rate of change of constraint violation for a unit change of the solution vector. On the other hand, the inverse of the gradient matrix provides the rate of change of the solution vector with respect to the changes of constraint violation. With this information, it is clear that when the degree of constraint violation (\( \Delta \mathbf{V} \)) is known, the infeasible solutions can be repaired accordingly. Let \( \mathbf{u} \) and \( \ell \) be the vectors of the upper and lower limits of inequality constraints (Eq. (2)), and \( \mathbf{c} \) be the vector of the right-hand side of equality constraints (Eq. (3)), the degree of constraint violation is computed by

\[
\Delta \mathbf{V} = \begin{bmatrix} \min\{0, \mathbf{u} - \mathbf{g}(\mathbf{x})\} + \max\{0, \ell - \mathbf{g}(\mathbf{x})\} \\ \mathbf{h}(\mathbf{x}) - \mathbf{c} \end{bmatrix}.
\]

It should be noted that \( \nabla_{\mathbf{x}} \mathbf{V} \) is in general not a square matrix (\( M + N \neq I \)), thus not invertible. However, the Moore–Penrose inverse or pseudoinverse (\( \nabla_{\mathbf{x}} \mathbf{V}^+ \)) [18], which is the approximate inverse of \( \nabla_{\mathbf{x}} \mathbf{V} \), can be used instead in Eq. (8). It should further be noted that there is no need to adjust the values of non-violated constraints. Only the rows and columns of \( \nabla_{\mathbf{x}} \mathbf{V} \) corresponding to non-zero elements of \( \Delta \mathbf{V} \) are considered in the matrix inversion. To repair the infeasible solution, each individual value of the solution vector has to be adjusted in order to minimize the amount of constraint violation. In general, the adjustment according
to Eq. (8) might not be able to repair the infeasible solutions right away due to the number of constraints and non-linearity of the constraints. If there is only one linear constraint, this adjustment can lead to the feasible region in one step. This is because the adjustment ($\Delta x$) is basically a linear approximation. On the other hand, for the general case, the adjustment has to be repeated until all constraints are satisfied or the adjustment is insignificant (i.e., the norm of $\Delta x$ is less than a certain threshold, $\eta$). If it is impossible to make the infeasible solution feasible, a very high penalty (e.g., $p(x) = 10^{10}$) is applied. This repair procedure is embedded into a simple GA as a special operator. There is one parameter associated with the repair method, which is the repair rate ($P_r$). Similar to the probability of mutation ($P_m$), the repair rate indicates the expected number of infeasible solutions undergoing the repair. The repair procedure, coupled with GA as shown in Fig. 1, can be summarized as follows:

*Step 1:* For any solution, determine the degree of constraint violation, $\Delta V$, using Eq. (9). If the solution is infeasible, generate a random number $\gamma$ from the interval $[0,1]$. If $\gamma \leq P_r$, set $t = 1$ and go to step 2, otherwise return.

*Step 2:* Compute $\nabla_x V^+$ and $\Delta x$. Note that only non-zero elements of $\Delta V$ are included for further computations of $\nabla_x V^+$ and $\Delta x$.

*Step 3:* Update the solution vector by $x^{t+1} = x^t + \nabla_x V^+ \times \Delta V$.

*Step 4:* Evaluate the updated solution vector. If $\max_{i} |x_i^{t+1} - x_i^t| \geq \eta$ and solution is still infeasible, set $t = t + 1$ and go to step 2, otherwise return. $\eta$ is the minimum adjustment for the solution vector (e.g., $10^{-4}$).
3.2. Illustrative example

The above procedure can be demonstrated using the following example. Consider that there are four inequality constraints and one equality constraint, given by Eq. (10), defining the feasible region as shown in Fig. 2. It is clear that the solution must strictly satisfy the equality constraint (thicker line in Fig. 2) in order to be feasible.

\[
g_1 : x_1^2 + x_2^2 \leq 25, \quad g_2 : x_1 \leq 3, \\
g_3 : x_1 \geq 0, \quad g_4 : x_2 \geq 0, \\
h_1 : x_1 + x_2 = 5. \quad (10)
\]

Suppose at the first iteration, \( x_1 = 2 \) and \( x_2 = 2 \) (point A in Fig. 2). It can be seen that only \( h_1 \) is violated; therefore, the solution will be adjusted according to the degree of violation with respect to \( h_1 \).

The required components of Eq. (8) are computed as follows:

\[
V = \begin{bmatrix} 8 \\ 2 \\ 2 \\ 4 \end{bmatrix} \Rightarrow \Delta V = [1], \quad \text{and} \quad \nabla_x V = \begin{bmatrix} 1 & 1 \end{bmatrix} \Rightarrow \nabla_x V^+ = \begin{bmatrix} 0.500 \\ 0.500 \end{bmatrix}.
\]

Then, the solution is moved to \([2.500, 2.500]\) (point B in Fig. 2), which satisfies all constraints. Another example is the solution at point C at which \( x_1 = 6 \) and \( x_2 = 0 \). At this point, \( g_1, g_2, \) and \( h_1 \) are
violated. The degree of constraint violation and the gradient of the constraint set at this point are:

\[
V = \begin{bmatrix}
36 \\
6 \\
6 \\
0 \\
6
\end{bmatrix} \Rightarrow \Delta V = \begin{bmatrix}
-11 \\
-3 \\
-1
\end{bmatrix}, \text{ and}
\]

\[
\nabla_x V = \begin{bmatrix}
12 & 0 \\
1 & 0 \\
1 & 1
\end{bmatrix} \Rightarrow \nabla_x V^+ = \begin{bmatrix}
0.0828 & 0.0069 & 0 \\
-0.0828 & -0.0069 & 1
\end{bmatrix}.
\]

According to the information obtained above, the adjusted solution becomes \([5.0690, -0.0690]\) (point D in Fig. 2), which is still infeasible and requires further adjustment. \(g_1\) and \(g_2\) are still violated while \(h_1\) becomes satisfactory, but \(g_4\), which was satisfactory prior to the adjustment, becomes violated instead. Next, all violated constraints must be taken into further consideration. The degree of constraint violation and the gradient of the constraint set at point D are:

\[
V = \begin{bmatrix}
25.6992 \\
5.0690 \\
5.0690 \\
-0.0690 \\
5.0000
\end{bmatrix} \Rightarrow \Delta V = \begin{bmatrix}
-0.6992 \\
-2.0690 \\
0.0690
\end{bmatrix}, \text{ and}
\]

\[
\nabla_x V = \begin{bmatrix}
10.1379 & -0.1379 \\
1 & 0 \\
0 & 1
\end{bmatrix} \Rightarrow \nabla_x V^+ = \begin{bmatrix}
0.0977 & 0.0098 & 0.0135 \\
-0.0013 & 0.0135 & 0.9998
\end{bmatrix}.
\]

At this step, the solution is moved to point E \((x_1 = 4.9813 \text{ and } x_2 = -0.0270)\), which violates \(g_2, g_4\) and \(h_1\). The same process is repeated and, with two additional adjustments, the solution is moved to point G, which is on the feasible line, where \(x_1 = 3\) and \(x_2 = 2\). As illustrated by this second example, the set of violated constraints could be varied during the adjustment process. In other words, the proposed repair could make a constraint(s) that was originally satisfactory become violated after the adjustment or vice versa. However, the degree of constraint violation, as a whole, is reduced (e.g., the solution is moved closer to the feasible region).

This simple example demonstrates the implementation of the proposed repair method. After the infeasible solutions are directed toward the feasible region, the GA plays a major role in searching for the optimal solution. In order to further illustrate its practical use, the proposed method is tested in the next section using a set of benchmark problems used in the literature.

4. Experiments

4.1. Description of test problem and GA parameters

In this study, we apply the proposed repair method with a real-coded GA to solve the same 11 benchmark problems studied by Michaelwicz and Schoenauer [19] and further studied in [4,8,9]. This test suite includes three maximization problems (G2, G3, and G8) and eight minimization problems (G1, G4 to
Table 1
Characteristics of benchmark problems [4]

<table>
<thead>
<tr>
<th>No.</th>
<th>( I )</th>
<th>Type of ( f(x) )</th>
<th>( \kappa (%) )</th>
<th>( LI )</th>
<th>( NE )</th>
<th>( NI )</th>
<th>( a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>Quadratic</td>
<td>0.0111</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>Non-linear</td>
<td>99.8474</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>Polynomial</td>
<td>0.0000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>Quadratic</td>
<td>52.1230</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Cubic</td>
<td>0.0000</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>Cubic</td>
<td>0.0066</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>Quadratic</td>
<td>0.0003</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>Non-linear</td>
<td>0.8560</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>Polynomial</td>
<td>0.5121</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>Linear</td>
<td>0.0010</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>Quadratic</td>
<td>0.0000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Since all constraints have explicit and simple functional forms (See [4] or Appendix A for complete formulations), the gradient of the constraints can be derived directly from the constraint set. The characteristics of problems, including the number of decision variables \( (I) \), type of objective function, size of feasible region \( (\kappa) \), and the number of linear inequality \( (LI) \), non-linear equality \( (NE) \), non-linear inequality \( (NI) \), and binding \( (a) \) constraints, are summarized in Table 1. The size of feasible region, empirically determined by simulation [4], indicates the difficulty to randomly generate a feasible solution.

The parameters and settings of the GA used in this study are as follows. The population size is 64 for problems with more than two decision variables or 32 otherwise. Crossover probability is 0.50 (arithmetic crossover), while mutation probability is 0.10 (random mutation). The initial population is randomly generated within the boundary specified for each decision variable. The roulette wheel (proportional) selection and the half-replacement strategy are used for reproduction. Chromosomes in each generation are ranked based on their fitness (see Eq. (4)) and divided into two parts. Only the chromosomes in the top half (i.e., better solutions) are eligible for reproduction. Chromosomes in the bottom half will be replaced by the offspring generated by crossover and mutation. The genetic search will be performed until the maximum number of generations (5000 generations) is reached, or the reference or better solution is found. In the case that the infeasible solution cannot successfully be repaired; a very high penalty will be applied (e.g., \( p(x) = 10^{10} \)). In this experiment, the proposed repair method coupled with a GA is performed 20 times for each problem in order to investigate its efficiency and robustness. Most of parameters used here are chosen similar to those used in [4,9] for comparison purpose.

4.2. Effects of repair probability \( (P_r) \)

One of the questions when employing the proposed method is how many infeasible solutions should be repaired in each generation. Certainly, we can repair every infeasible solution \( (P_r = 100\%) \) to increase the chance of getting a better solution. However, this will require a lot of computational effort, especially for problems with very complicated function evaluations. Since the main goal of the repair method is to
provide good genetic material for genetic search, only a subset of the infeasible solutions is needed to be repaired. However, it should be noted that the repair rate for each problem could be dependent on the size of the feasible space ($\kappa$) mentioned earlier.

One can expect that, for problems with a very small $\kappa$ value, the number of solutions undergoing the repair may have to increase considerably. This section examines the effects of different repair probabilities on the solution quality as well as the computational efforts for acquiring the optimal solution. Six levels of repair intensity (0%, 10%, 20%, 50%, 75% and 100% of infeasible solutions) are examined. The quality of the solutions is defined by the deviation of the solution from the reference solution whereas the number of function evaluations required to obtain an optimal solution is used as a performance index. Please note that other constraint-handling methods to which the repair method is compared used 350,000 function evaluations (i.e., [4,9] used 5000 generations each of which consists of 70 solutions while [8] used 1750 generations with 200 solutions each). The experimental results are reported in Table 2. In addition to the two performance measures mentioned earlier, the average objective value, the standard deviation of objective value, the success rate, the feasibility rate, and the average number of generations spent to obtain the optimal solution, are also reported for each problem. The success rate is the percentage of obtaining the reference (or better) solution among 20 runs. The feasibility rate is the percentage of obtaining a feasible solution, but not necessarily the optimal solution at the final generation. If the reference solution could not be found, the numbers of generations and function evaluations spent until the final generation are used.

The experimental results suggest that, in general, a random repair of 10–20% of the infeasible solutions at each generation is sufficient to enhance the performance of genetic search and the quality of solutions. Although there is no guarantee that the optimal solutions are always found for all problems within this range of repair rates (e.g., see G2, G3, G7, G9, and G10), the quality of the solution could be considered fairly good already (i.e., the absolute deviation from the reference solution is less than 1% excepting G2). One of the general observations is that as the repair rate increases, the genetic search tends to find the optimal solution faster (i.e., lower number of generations). However, the average number of function evaluations is also increased since a higher effort is put toward repairing the infeasible solutions. As can be seen from Table 2, some of the problems, such as G2, G4, G6, G8, G9, and G11, may not need any constraint handling since the genetic search can easily locate the feasible region (i.e., the feasibility rate is 100%) as indicated by the results of the zero repair rate. This observation may be due to the fact that the sizes of such problems (i.e., number of decision variables) are quite small (G6, G8, and G11) or the sizes of the feasible region are quite large (G2 and G4). However, there are only two problems (G4 and G8) of which the reference solutions could be found almost all the times (i.e., the success rate is 100%) without the need to use the repair method. This observation shows that, in general, the proposed repair method indeed provides sufficiently good genetic information to the GA to improve the quality of solutions.

4.3. Comparison with other methods

In this section, the experimental results obtained by the repair method using a 10% repair rate are used to compare with the results obtained by the homomorphous mapping method [4], the stochastic ranking method [8], and the self-adaptive fitness formulation method [9]. It should be noted that Runasson and Yao [8] used the evolutionary strategy (ES) algorithm to solve these benchmark problems while the other studies [4,9] including ours used the GAs. Table 3 compares the results of these problems in
Table 2
Effects of repair probability on solution quality and performance of GA

<table>
<thead>
<tr>
<th>Problem</th>
<th>Performance</th>
<th>Repair probability (%)</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1 Minimization (-15,000)</td>
<td>Deviation (%)</td>
<td>N/A</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>Average obj. value</td>
<td>N/A</td>
<td>-15.0000</td>
<td>-15.0000</td>
<td>-15.0000</td>
<td>-15.0000</td>
<td>-15.0000</td>
<td>-15.0000</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>N/A</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>Success rate (%)</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Feasibility rate (%)</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Avg. generation</td>
<td>5000</td>
<td>884</td>
<td>963</td>
<td>588</td>
<td>524</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg.</td>
<td>320,000</td>
<td>95,512</td>
<td>150,128</td>
<td>188,236</td>
<td>244,496</td>
<td>277,856</td>
<td></td>
</tr>
<tr>
<td>G2 Maximization (0.803553)</td>
<td>Deviation (%)</td>
<td>-3.4123</td>
<td>-2.1690</td>
<td>-2.2388</td>
<td>-1.3014</td>
<td>-1.2443</td>
<td>-0.8413</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average obj. value</td>
<td>0.7761</td>
<td>0.7857</td>
<td>0.7856</td>
<td>0.7931</td>
<td>0.7936</td>
<td>0.7968</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>0.0135</td>
<td>0.0137</td>
<td>0.0126</td>
<td>0.0095</td>
<td>0.0118</td>
<td>0.0069</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Success rate (%)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feasibility rate (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg. generation</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>4929</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg.</td>
<td>320,000</td>
<td>331,972</td>
<td>344,923</td>
<td>387,658</td>
<td>423,729</td>
<td>415,263</td>
<td></td>
</tr>
<tr>
<td>G3 Maximization (1.0000)</td>
<td>Deviation (%)</td>
<td>-0.0080</td>
<td>-0.0003</td>
<td>-0.0001</td>
<td>-0.0001</td>
<td>-0.0001</td>
<td>-0.0001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average obj. value</td>
<td>0.9999</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>0.0001</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Success rate (%)</td>
<td>0</td>
<td>5</td>
<td>40</td>
<td>50</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feasibility rate (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg. generation</td>
<td>5000</td>
<td>5000</td>
<td>4975</td>
<td>4229</td>
<td>4050</td>
<td>3846</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg.</td>
<td>320,000</td>
<td>399,804</td>
<td>474,288</td>
<td>604,399</td>
<td>743,564</td>
<td>865,676</td>
<td></td>
</tr>
<tr>
<td>G4 Minimization (-30,665.5386)</td>
<td>Deviation (%)</td>
<td>-0.0001</td>
<td>-0.0001</td>
<td>-0.0001</td>
<td>-0.0001</td>
<td>-0.0001</td>
<td>-0.0001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Success rate (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feasibility rate (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg. generation</td>
<td>372</td>
<td>383</td>
<td>348</td>
<td>266</td>
<td>171</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg.</td>
<td>23,562</td>
<td>26,981</td>
<td>27,207</td>
<td>28,427</td>
<td>23,175</td>
<td>26,419</td>
<td></td>
</tr>
<tr>
<td>G5 Minimization (5126.4981)</td>
<td>Deviation (%)</td>
<td>N/A</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average obj. value</td>
<td>N/A</td>
<td>5126.4981</td>
<td>5126.4981</td>
<td>5126.4981</td>
<td>5126.4981</td>
<td>5126.4981</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>N/A</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Success rate (%)</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feasibility rate (%)</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg. generation</td>
<td>5000</td>
<td>476</td>
<td>390</td>
<td>137</td>
<td>79</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg.</td>
<td>320,000</td>
<td>39,459</td>
<td>38,508</td>
<td>21,137</td>
<td>15,867</td>
<td>15,701</td>
<td></td>
</tr>
<tr>
<td>G6 Minimization</td>
<td>Deviation (%)</td>
<td>2.2140</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average obj. value</td>
<td>-6807.6775</td>
<td>-6961.8139</td>
<td>-6961.8139</td>
<td>-6961.8139</td>
<td>-6961.8139</td>
<td>-6961.8139</td>
<td></td>
</tr>
</tbody>
</table>
Table 2 (continued)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Performance</th>
<th>Repair probability (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

|                  | Standard deviation | 133.0573 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
|                  | Success rate (%)   | 0       | 100   | 100   | 100   | 100   | 100   |
|                  | Feasibility rate (%) | 100    | 100   | 100   | 100   | 100   | 100   |
|                  | Avg. generation    | 5000    | 368   | 86    | 23    | 22    | 12    |
|                  | Avg. func_evaluation | 160,000 | 13,577 | 3874  | 1770  | 2221  | 1716  |
| G7 Minimization (24.3062) | Deviation (%) | 6.3745  | 0.6819 | 0.3087 | 0.0642 | 0.0155 | 0.0125 |
|                  | Standard deviation | N/A    | 0.1291 | 0.0643 | 0.0241 | 0.0037 | 0.0044 |
|                  | Success rate (%)   | 0      | 0     | 0     | 0     | 0     | 0     |
|                  | Feasibility rate (%) | 5      | 100   | 100   | 100   | 100   | 100   |
|                  | Avg. generation    | 5000    | 5000  | 5000  | 5000  | 5000  | 5000  |
|                  | Avg. func_evaluation | 320,000 | 428,314 | 547,532 | 940,834 | 1,268,823 | 1,594,469 |
| G8 Maximization (0.095825) | Deviation (%) | −3.4793  | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
|                  | Average obj. value | 0.092491 | 0.095825 | 0.095825 | 0.095825 | 0.095825 | 0.095825 |
|                  | Standard deviation | 0.0149 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
|                  | Success rate (%)   | 95     | 100   | 100   | 100   | 100   | 100   |
|                  | Feasibility rate (%) | 100    | 100   | 100   | 100   | 100   | 100   |
|                  | Avg. generation    | 1268    | 179   | 173   | 93    | 78    | 81    |
|                  | Avg. func_evaluation | 40,582  | 6217  | 6517  | 4486  | 4586  | 5550  |
| G9 Minimization (680.6301) | Deviation (%) | 0.0302  | 0.0012 | 0.0001 | 0.0000 | 0.0000 | 0.0000 |
|                  | Average obj. value | 680.8357 | 680.6381 | 680.6309 | 680.6301 | 680.6301 | 680.6301 |
|                  | Standard deviation | 0.2075 | 0.0066 | 0.0014 | 0.0000 | 0.0000 | 0.0000 |
|                  | Success rate (%)   | 0      | 0     | 20    | 90    | 85    | 95    |
|                  | Feasibility rate (%) | 100    | 100   | 100   | 100   | 100   | 100   |
|                  | Avg. generation    | 5000    | 5000  | 4785  | 3023  | 2709  | 2313  |
|                  | Avg. func_evaluation | 320000  | 388453 | 440255 | 409281 | 468869 | 488421 |
| G10 Minimization (7049.3309) | Deviation (%) | N/A  | 0.0033 | −0.0005 | −0.0005 | −0.0005 | −0.0004 | −0.0006 |
|                  | Average obj. value | N/A    | 7049.5659 | 7049.2932 | 7049.2955 | 7049.3036 | 7049.2851 |
|                  | Standard deviation | N/A    | 0.5699 | 0.0321 | 0.0247 | 0.0245 | 0.0281 |
|                  | Success rate (%)   | 0      | 65    | 95    | 100   | 100   | 100   |
|                  | Feasibility rate (%) | 0      | 100   | 100   | 100   | 100   | 100   |
|                  | Avg. generation    | 5000    | 3913  | 2148  | 1001  | 549   | 500   |
|                  | Avg. func_evaluation | 320000  | 572,629 | 530,019 | 525,334 | 407,038 | 474,531 |
| G11 Minimization (0.7500) | Deviation (%) | 33.3333 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
|                  | Average obj. value | 1.0000 | 0.7500 | 0.7500 | 0.7500 | 0.7500 | 0.7500 |
|                  | Standard deviation | 0      | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
|                  | Success rate (%)   | 100    | 100   | 100   | 100   | 100   | 100   |
|                  | Feasibility rate (%) | 100    | 100   | 100   | 100   | 100   | 100   |
|                  | Avg. generation    | 5000    | 182   | 103   | 45    | 32    | 29    |
|                  | Avg. func_evaluation | 160,000 | 7215  | 4901  | 3295  | 3056  | 3329  |
Table 3
Comparison of the best and the worst results for benchmark problems

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Best</td>
<td>Worst</td>
<td>Best</td>
<td>Worst</td>
<td>Best</td>
<td>Worst</td>
</tr>
<tr>
<td>G2a</td>
<td>0.803553</td>
<td>0.794860</td>
<td>0.782790</td>
<td>0.803515</td>
<td>0.726288</td>
<td>0.802970</td>
<td>0.760430</td>
</tr>
<tr>
<td>G3a</td>
<td>1.00000</td>
<td>0.99780</td>
<td>0.99600</td>
<td>1.00000</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.99970</td>
</tr>
<tr>
<td>G5</td>
<td>5126.4981</td>
<td>—</td>
<td>—</td>
<td>5126.4970</td>
<td>5142.4720</td>
<td>5126.9890</td>
<td>6089.4300</td>
</tr>
<tr>
<td>G6</td>
<td>-6961.8138</td>
<td>-6944.4000</td>
<td>-6390.6000</td>
<td>-6961.8140</td>
<td>-6350.2620</td>
<td>-6961.8000</td>
<td>-6961.8000</td>
</tr>
<tr>
<td>G8a</td>
<td>0.0958250</td>
<td>0.0958250</td>
<td>0.0958246</td>
<td>0.0958250</td>
<td>0.0958250</td>
<td>0.0958250</td>
<td>0.0958250</td>
</tr>
<tr>
<td>G10</td>
<td>7049.3309</td>
<td>7321.2000</td>
<td>7685.8000</td>
<td>7054.3160</td>
<td>8835.6550</td>
<td>7061.3400</td>
<td>8288.7900</td>
</tr>
<tr>
<td>G11</td>
<td>0.7500</td>
<td>0.7500</td>
<td>0.7500</td>
<td>0.7500</td>
<td>0.7500</td>
<td>0.7500</td>
<td>0.7500</td>
</tr>
</tbody>
</table>

a Maximization problem.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Standard deviation</td>
<td>Average</td>
<td>Standard deviation</td>
</tr>
<tr>
<td>G1</td>
<td></td>
<td>15.0000</td>
<td>0.00E+00</td>
<td>15.0000</td>
<td>0.00E+00</td>
</tr>
<tr>
<td>G2a</td>
<td></td>
<td>0.803553</td>
<td>2.00E-02</td>
<td>0.798100</td>
<td>1.20E-02</td>
</tr>
<tr>
<td>G3a</td>
<td></td>
<td>1.000000</td>
<td>1.90E-04</td>
<td>0.997000</td>
<td>7.50E-05</td>
</tr>
<tr>
<td>G4</td>
<td></td>
<td>−30,665.5000</td>
<td>2.00E-05</td>
<td>−30,653.1000</td>
<td>0.00E+00</td>
</tr>
<tr>
<td>G5</td>
<td></td>
<td>5126.4981</td>
<td>1.90E-02</td>
<td>5128.8810</td>
<td>3.50E+00</td>
</tr>
<tr>
<td>G6</td>
<td></td>
<td>6961.8138</td>
<td>1.60E+00</td>
<td>6961.8000</td>
<td>0.00E+00</td>
</tr>
<tr>
<td>G7</td>
<td></td>
<td>24.3062</td>
<td>6.60E-02</td>
<td>25.5450</td>
<td>1.14E+00</td>
</tr>
<tr>
<td>G8a</td>
<td></td>
<td>0.0958250</td>
<td>2.60E-17</td>
<td>0.0958250</td>
<td>0.00E+00</td>
</tr>
<tr>
<td>G9</td>
<td></td>
<td>680.6301</td>
<td>3.40E-02</td>
<td>682.5600</td>
<td>5.90E-03</td>
</tr>
<tr>
<td>G10</td>
<td></td>
<td>7049.3309</td>
<td>5.30E+02</td>
<td>7559.1920</td>
<td>5.70E+02</td>
</tr>
<tr>
<td>G11</td>
<td></td>
<td>0.7500</td>
<td>8.00E-05</td>
<td>0.7500</td>
<td>0.7500</td>
</tr>
</tbody>
</table>

aMaximization problem.
terms of the best and the worst solutions found by each method. Table 4 compares the stability of each method for solving these problems, which is indicated by the average and the standard deviation of the objective values aggregated over 20 runs. From the comparison presented in Table 3, the gradient-based repair can generally obtain competitive solutions, and, in some cases (G4, G5, G6, G7, G9, and G10—6 out of 11 problems), obtain better solutions compared to those reported in [4,9]. As noted by Koziel and Michalewicz [4], the homomorphous mapping method has difficulty in transforming the equality constraints of problem G5 to equivalent inequality constraints; therefore, the solutions for this problem were not reported in their study. On the other hand, Farmani and Wright [9] were able to solve this problem using the self-adaptive fitness formulation method, but none of the 20 runs could obtain the reference solution. With the method proposed in this study, we are able to obtain the reference solution of problem G5 in all 20 runs.

Among these four methods (see Table 3), the stochastic ranking [8] appears to be the most promising method. It can in general produce the better, if not optimal, solutions for most of the problems. However, the gradient-based repair can also produce very similar results. In addition, it performs slightly better for problems G6 and G10 in which the stochastic ranking method seems to have some troubles (i.e., it was unable to obtain the reference solutions, and the solutions obtained from different runs are rather different—high standard deviation). For problem G4, the stochastic ranking method and the repair method can obtain solutions slightly better than the reference solution provided in the literature. The solution vectors of problem G4 rounded off to 4-decimal precision are [78.0000, 33.0000, 29.9953, 45.0000, 36.7758]. For problem G10, only the gradient-based repair method can produce a solution better than the reference solution. The solution vector is [574.7858, 1,362.1506, 5,112.3244, 181.6389, 295.5070, 218.3611, 286.1319, 395.5070]. It should be noted that the factors contributing to the performance of the proposed constraint-handling method are the high precision of the real-coded GA and the capability of the gradient-based repair scheme to quickly locate the promising area of the search space. If the feasible region of the problem can be easily located (i.e., a large $\kappa$ value or only a few decision variables), such as problems G4 and G8, the quality of solutions would depend heavily on the performance of the genetic operators; otherwise, on the technique used to handle constraints.

5. Conclusion

This study has proposed a constraint handling technique that can effectively repair the infeasible solutions based on the gradient of the constraint set. Such gradient information can be derived directly from the constraints or indirectly by the finite difference scheme.Coupled with a real-coded GA, experimental results clearly illustrate the attractiveness of the method for handling several types of constraint. It can produce competitive, if not better, solutions compared to the stochastic ranking method, which appears to be the most promising constraint-handling technique reported thus far in the literature. In addition, as indicated by the results of several test runs, the method proposed here is quite robust; similar solutions are always obtained (i.e., indicated by a small standard deviation of the objective value). Experiments were also conducted to examine the effects of repair probability, which is the only parameter in the proposed method, on the computational requirements and solution quality. Based on the results, we found that a random repair of 10–20% of the infeasible solutions is sufficient to enhance the performance of genetic search and the quality of solutions.
Acknowledgements

This research was supported by the NSF CAREER Grant: CMS-0134161.

Appendix A. Formulation of benchmark problems [4]

Problem G1:

\[
\text{Min } 5 \sum_{i=1}^{4} x_i - 5 \sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i \\
\text{S.t.}
\begin{align*}
2x_1 + 2x_2 + x_{10} + x_{11} & \leq 10, \\
2x_1 + 2x_3 + x_{10} + x_{12} & \leq 10, \\
2x_2 + 2x_3 + x_{11} + x_{12} & \leq 10, \\
-8x_1 + x_{10} & \leq 0, \\
-8x_2 + x_{11} & \leq 0, \\
-8x_3 + x_{12} & \leq 0, \\
-2x_4 - x_5 + x_{10} & \leq 0, \\
-2x_6 - x_7 + x_{11} & \leq 0, \\
-2x_8 - x_9 + x_{12} & \leq 0, \\
0 \leq x_i & \leq 1, \ i = 1, 2, \ldots, 9, 13, \\
0 \leq x_i & \leq 100, \ i = 10, 11, 12.
\end{align*}
\]

Problem G2:

\[
\text{Max } \left| \frac{\sum_{i=1}^{n} \cos^4 x_i - 2 \prod_{i=1}^{n} \cos^2 x_i}{\sqrt{\sum_{i=1}^{n} ix_i^2}} \right|
\text{S.t.}
\begin{align*}
\prod_{i=1}^{n} x_i & \geq 0.75, \\
\sum_{i=1}^{n} x_i & \leq 7.5n, \\
0 \leq x_i & \leq 10, \ i = 1, 2, \ldots, n.
\end{align*}
\]

Problem G3:

\[
\text{Max } (\sqrt{n})^n \prod_{i=1}^{n} x_i \\
\text{S.t.}
\begin{align*}
\sum_{i=1}^{n} x_i^2 & = 1, \\
0 \leq x_i & \leq 1, \ i = 1, 2, \ldots, n.
\end{align*}
\]
Problem G4:
\[
\begin{align*}
\text{Min} & \quad 5.3578547 x_3^2 + 0.8356891 x_1 x_5 - 37.293239 x_1 - 40792.141 \\
\text{S.t.} & \quad 0 \leq 85.334407 + 0.0056858 x_2 x_5 + 0.0006262 x_1 x_4 - 0.0022053 x_3 x_5 \leq 92, \\
& \quad 90 \leq 80.51249 + 0.0071317 x_2 x_5 + 0.0029955 x_1 x_2 + 0.0021813 x_2^2 \leq 110, \\
& \quad 20 \leq 9.300961 + 0.0047026 x_3 x_5 + 0.0012547 x_1 x_3 + 0.0019085 x_3 x_4 \leq 25, \\
& \quad 78 \leq x_1 \leq 102, \\
& \quad 33 \leq x_2 \leq 45, \\
& \quad 27 \leq x_i \leq 45, \quad i = 3, 4, 5.
\end{align*}
\]

Problem G5:
\[
\begin{align*}
\text{Min} & \quad 3 x_1 + 0.000001 x_1^3 + (0.000002/3) x_2^3 \\
\text{S.t.} & \quad x_4 - x_3 + 0.55 \geq 0, \\
& \quad x_3 - x_4 + 0.55 \geq 0, \\
& \quad 1000.0 \sin(-x_3 - 0.25) + 1000.0 \sin(-x_4 - 0.25) + 894.8 - x_1 = 0, \\
& \quad 1000.0 \sin(x_3 - 0.25) + 1000.0 \sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0, \\
& \quad 1000.0 \sin(x_4 - 0.25) + 1000.0 \sin(x_4 - x_3 - 0.25) + 1294.8 = 0, \\
& \quad 0 \leq x_i \leq 1200, \quad i = 1, 2, \\
& \quad -0.55 \leq x_i \leq 0.55, \quad i = 3, 4.
\end{align*}
\]

Problem G6:
\[
\begin{align*}
\text{Min} & \quad (x_1 - 10)^3 + (x_2 - 20)^3 \\
\text{S.t.} & \quad (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geq 0, \\
& \quad -(x_1 - 6)^2 - (x_2 - 5)^2 + 82.81 \geq 0, \\
& \quad 13 \leq x_1 \leq 100, \\
& \quad 0 \leq x_2 \leq 100.
\end{align*}
\]

Problem G7:
\[
\begin{align*}
\text{Min} & \quad x_1^2 + x_2^2 + x_1 x_2 - 14 x_1 - 16 x_2 + (x_3 - 10)^2 \\
& \quad + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5 x_7^2 \\
& \quad + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \\
\text{S.t.} & \quad 105 - 4 x_1 - 5 x_2 + 3 x_7 - 9 x_9 \geq 0, \\
& \quad 8 x_1 - 2 x_2 - 5 x_9 + 2 x_{10} + 12 \geq 0, \\
& \quad -4 x_1^2 - x_2^2 + 3 x_1 x_2 - 2 x_3^2 - 5 x_6 + 11 x_7 \geq 0, \\
& \quad -10 \leq x_i \leq 10, \quad i = 1, 2, \ldots, 7, \\
& \quad 3 x_1 - 6 x_2 - 12(x_9 - 8)^2 + 7 x_{10} \geq 0, \\
& \quad -3(x_i - 2)^2 - 4(x_2 - 3)^2 - 2 x_3^2 + 7 x_4 + 120 \geq 0, \\
& \quad -10 x_1 + 8 x_2 + 17 x_7 - 2 x_8 \geq 0, \\
& \quad -x_1^2 - 2(x_2 - 2)^2 + 2 x_1 x_2 - 14 x_5 + 6 x_6 \geq 0, \\
& \quad -5 x_1^2 - 8 x_2 - (x_3 - 6)^2 + 2 x_4 + 40 \geq 0, \\
& \quad -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3 x_3^2 + x_6 + 30 \geq 0, \\
& \quad -10 \leq x_i \leq 10, \quad i = 1, 2, \ldots, 10.
\end{align*}
\]
Problem G8:
\[
\begin{align*}
\text{Max} & \quad \frac{\sin^3(2\pi x_1) \sin(2\pi x_2)}{x_1^3(x_1 + x_2)} \\
\text{S.t.} & \quad x_1^2 - x_2 + 1 \leq 0, \\
& \quad 1 - x_1 + (x_2 - 4)^2 \leq 0, \\
& \quad 0 \leq x_i \leq 10, \quad i = 1, 2.
\end{align*}
\]

Problem G9:
\[
\begin{align*}
\text{Min} & \quad (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 \\
& \quad + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \\
\text{S.t.} & \quad 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0, \\
& \quad 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0, \\
& \quad 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0.
\end{align*}
\]

Problem G10:
\[
\begin{align*}
\text{Min} & \quad x_1 + x_2 + x_3 \\
\text{S.t.} & \quad 1 - 0.0025(x_4 + x_6) \geq 0, \\
& \quad 1 - 0.0025(x_5 + x_7 - x_4) \geq 0, \\
& \quad 1 - 0.01(x_8 - x_5) \geq 0, \\
& \quad x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333 \geq 0, \\
& \quad x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \geq 0, \\
& \quad x_3x_8 - 125000 - x_3x_5 + 2500x_5 \geq 0, \\
& \quad 100 \leq x_1 \leq 10000, \\
& \quad 1000 \leq x_i \leq 10000, \quad i = 2, 3, \\
& \quad 10 \leq x_i \leq 1000, \quad i = 4, 5, \ldots , 8.
\end{align*}
\]

Problem G11:
\[
\begin{align*}
\text{Min} & \quad x_1^2 + (x_2 - 1)^2 \\
\text{S.t.} & \quad x_2 - x_1^2 = 0, \\
& \quad -1 \leq x_i \leq 1, \quad i = 1, 2.
\end{align*}
\]

References