Paper 4:

A genetic algorithm for constrained and multiobjective optimization

Eduardo Camponogara and **Sarosh N. Talukdar**
Engineering Design Research Center
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890, USA
phone (412)268-6411, fax (412)268-5229
E-mail: camponog@cs.cmu.edu, snt@edrc.cmu.edu

Abstract. Although several schemes for solving constrained optimization problems by genetic algorithms are available in the literature, none of them is dominant. They also require a significant effort in fine tuning parameters. We propose a scheme that calculates improvement directions from pareto sets defined by the objective function and constraint violations. This scheme outperformed genetic algorithms based on penalty methods. Furthermore, the scheme is conceptually elegant, simple, and applicable to multiobjective optimization problems.

Keywords: genetic algorithms, optimization /constraint, optimization /multiobjective, pareto sets

4.1 Introduction

Several optimization problems can be formalized as nonlinear programming problems. The design of electrical networks, mechanical components, and facility location are examples. Usually, a vector x represents a solution and it is sought a solution x that minimizes an objective function [48]. Formally, a nonlinear programming problem (NPP) P is stated as follows:

$$\begin{array}{lll} \texttt{Minimize} & f(x) & \\ \texttt{Subject to} & & \\ & g_i(x) \leq 0 & \texttt{for all } i = 1, \ldots, m \\ & x \in R^n & \end{array} \tag{4.1}$$

The objective is to find a feasible solution x^* that has the lowest objective evaluation among all feasible solutions. Such a solution is said to be a global optimum. NPPs can be categorized into differentiable and nondifferentiable problems. The first class can be tackled with algorithms that make use of gradient information like steepest descent and sequential quadratic programming [60]. The nondifferentiable class is often tackled with genetic algorithm [113], simulated annealing [1], and other optimization techniques.

Several methods for handling NPPs by genetic algorithms are available in the literature. Most of them penalize the constraint violations which are incorporated into the objective function. Such a hybrid objective function defines the fitness function for trial solutions in memory. These methods were extensively surveyed by Michalewicz [329] who discussed their strengths and weaknesses.

This paper proposes an alternative scheme for solving NPPs by genetic algorithms. The fitness function of trial solutions are not determined by a combined objective and penalty functions, but by the concept of dominance. In the proposed method, constraint violations define objective functions to be minimized as well as the objective function. The dominance relation of solutions in memory defines a partial order (acyclic graph) that can be represented by pareto sets. These pareto sets have two roles in the proposed scheme, denoted by pareto scheme (PS):

- **Evolution selection**. From one generation to the next, only nondominated solutions are kept.
- **Combination operator**. Solutions in distinct pareto sets are used to calculate improvement directions. Offspring of these solutions are created by searching through the line defined by them.

The next sections review some of the GAs already applied to NPPs, describe the pareto scheme, and discuss the results achieved in a set of test problems.

4.1.1 Review of GAs applied to NPPs

Most of the methods for solving nonlinear programming problems by genetic algorithms are based on penalty functions, which are combined with the objective to express the fitness function. A random initial population is evolved through crossover combinations until a stopping criterion is reached. In this section, $f_j(x)$ is defined as the violation of constraint j. For inequalities constraints, $g_i(x) <= 0$, $f_j(x) = \max(0, g_i(x))$. For equality constraints, $h_j(x) = 0$, $f_j(x) = |h_j(x)|$.

Method 1

The method 1 [217, 329] assumes that for each constraint $j \in \{1, \ldots, m\}$ a family of interval violations and appropriate penalties are given. R_{ij} denotes the penalty for constraint j when the violation is within interval i. Usually, the higher the violation, the higher the penalty. The fitness function is given by:

$$\text{fit}(x) \quad = \quad f(x) + \sum_{j=1}^{m} R_{ij} * f_j^2(x)$$

A drawback of this method is the large number of parameters.

Method 2

This method [247, 329] defines a dynamic penalty scheme. Each trial solution x is evaluated by the following fitness function:

$$\text{fit}(x) \quad = \quad f(x) + (C.t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(x)$$

where C is a constant, t is the generation number, and α and β are parameters. The penalties for violations increase towards the end of the optimization. This method requires a lower number of parameters when compared to method 1.

Method 3

The third method [448, 329] gradually considers the constraints. It starts with a random population and one constraint (iteration $j = 1$). This population is evolved until a percentage threshold ϕ of the population satisfy the constraint j. When this threshold is achieved, another constraint is added and the process is repeated. At iteration j, all trial solutions that do not satisfy the constraints $1, \ldots, j-1$ are rejected.

Method 4

The fourth method [328, 329] is a combined genetic algorithm and simulated annealing. It also has a special treatment for linear constraints, which are always satisfied by applying closed operators. The method has a temperature σ that is gradually decreased. It starts with a single solution that satisfies the linear constraints and an initial temperature $\sigma = \sigma_0$. A population is evolved using the following fitness function:

$$\text{fit}(x) \quad = \quad f(x) + \frac{1}{2\sigma} \sum_{j=1}^{m} f_j^2(x)$$

After some generations, the temperature σ is decreased. The best solution is taken as the single solution for the next iteration and the process is repeated. When the temperature falls below a final temperature σ_f, the algorithm halts.

Method 5

The fifth method [388, 329] has a fitness function defined by

$$\text{fit}(x) \quad = \quad f(x) + r \sum_{j=1}^{m} f_j^2(x) + \lambda(t, x)$$

where r is a constant and $\lambda(t, x)$ is an iteration dependent function. The difference between this method and the classical penalty function lays on $\lambda(t, x)$ which guarantees that any feasible solution x_1 has $\text{fit}(x_1) \leq \text{fit}(x_2)$ for any infeasible trial solution x_2.

Method 6

The last method simply rejects all infeasible solutions [38, 149, 329].

4.2 The Pareto Scheme

Genetic algorithms rely on fitness functions to evaluate trial solutions in memory. Penalty functions can be seen as a means of expressing the constraints of nonlinear programming problems. Nevertheless, one may explicitly express these constrains through dominance relations. Moreover, these relations establish pareto sets that can be used to design operators which estimate search directions. Some definitions are given below.

- **Definition 1. Dominance.** Given $f_i()$, $i = 0, \ldots, k$ objective functions to be analyzed, if $f_i(x_1) \leq f_i(x_2)$ for every $i \in \{0, \ldots, k\}$ and there is j such that $f_j(x_1) < f_j(x_2)$, then x_1 dominates x_2.

- **Definition 2. Nondominated solutions (Pareto Set).** Given a set of trial solutions S, a solution $x_e \in S$ is a nondominated solution in S if there does not exist $x \in S$ such that x dominates x_e.

- **Definition 3. List of pareto sets.** A list $L = \{S_1, \ldots, S_t\}$ is a list of pareto sets for a set S if S_i is the pareto set for the set $S - \{\cup_{j=1}^{i-1} S_j\}$ for every $i = 1, \ldots, t$ and $S = \{\cup_{j=1}^{t} S_j\}$. See figure 4.2 for an example.

In the pareto scheme we convert the constrained optimization problem into an unconstrained multiobjective problem where the constraint violations are functions to be minimized. Let $F = \{f_0, \ldots, f_k\}$ be the set of functions to be minimized where $f_0(x) = f(x)$ (original objective function), and the remaining functions are constraint violations or combinations of them. A possibility is to have two objectives, where $f_0(x) = f(x)$ and $f_1(x) = \sum_{i=1}^{m} \max(0, g_i(x))$.

At each iteration (generation) of the genetic algorithm, a list L of pareto sets is built for the trial solutions S in memory. The combination operator takes two pareto sets S_i and S_j, where $i < j$, and two solutions, $x_i \in S_i$ and $x_j \in S_j$, such that x_i dominates x_j. These solutions define a search direction $d = (x_i - x_j)/|x_i - x_j|$ which represents the willingness to simultaneously minimize all the objectives. d can be seen as an improvement direction. A line search is executed in this direction with the aim of identifying a solution x that dominates x_i and x_j. Figure 4.2 depicts an example. The mutation operator is essentially the

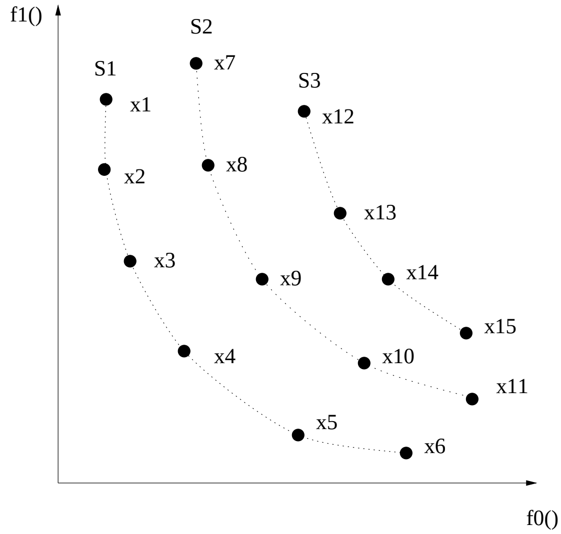

Figure 4.1: List of pareto sets $L = \{S_1, S_2, S_3\}$, where $S = \{x_1, \ldots, x_{15}\}$. The pareto set is defined for two objectives, $f_0()$ and $f_1()$.

combination operator, where the direction d is projected onto the axis of one variable j in the solution space.

After some offspring are generated by the combination operators, a new generation starts by discarding the less fitted trial solutions. In the pareto scheme, all solutions in S_j are discarded before any solution in S_i for any $i < j$. In other words, the selection rule discards the last pareto sets of the list $L = \{S_1, \ldots, S_l\}$. A pseudo-code for the proposed genetic algorithm is given below.

```
Let S be a random initial population
Let F = {f0(), ..., fk()} be a set of objectives to be minimized
While stopping criterion is not satisfied do
    Let L = {S1, ..., St} be the pareto list for S with respect to F
    If t = 1 then
        Replace half of the points in S by random points
        Rebuild the pareto list L
    Endif
    Let N = ∅ be the set of new points to be generated
    While |N| < m do
        Choose two pareto sets Si and Sj with i < j
        Choose two points xi ∈ Si and xj ∈ Sj such that xi dominates xj
        Let d = (xi − xj)/|xi − xj| be the search direction
        With probability α project d onto the axis
            of one variable j in the solution space
        Execute line search through the line defined by point xi and direction d
```

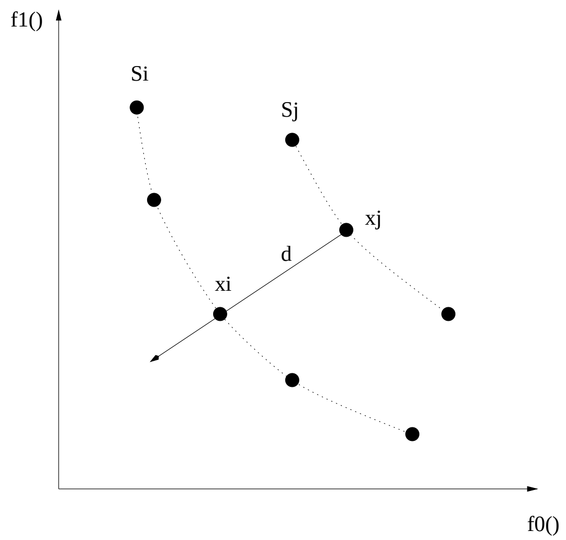

Figure 4.2: Example of searching direction obtained by points $x_i \in S_i$ and $x_j \in S_j$.

Let $U = \{u_0, \ldots, u_k\}$ be the set of points obtained in the line search such that u_j is the best point with evaluation $f_j()$
Let $N = N \cup U$
Endwhile
Let $S = S \cup N$
Let $L = \{S_1, \ldots, S_l\}$ be the pareto list for S
Remove $|N|$ points from the last pareto sets in list L
Endwhile

4.3 Results

The proposed genetic algorithm was applied to the set of benchmark problems adopted by Michalewichz in his survey [329], which are available at the end of this paper. All test problems were converted to the formulation 4.1, where constraints are expressed by inequalities. In the test problem 4, for instance, each equality was represented by two inequalities. The set of objectives F has two objective functions: (1) $f_0(x) = f(x)$, the original objective function, and (2) $f_1(x) = \sum_{i=1}^{m} \max(0, g_i(x))$, the total infeasibility. For line search in the direction d, it is adopted uniform search in the segment defined by x_j and $x_j + 2(x_i - x_j)$. The size s of the population is 500 trial solutions. The probability for mutation is 10%. The number of probe points in the uniform search and the number of offspring is $s/2$. The stopping criterion is 8 hours CPU time in a Sun Sparc/20.

The proposed GA was applied three times to each of the test problems and the results are summarized in table 4.1. For each problem, the table has the worst, median, and best objective obtained in three independent runs. The ratio $r(x)$ of a solution x is defined by $r(x) = \frac{f(x)-f(x^*)}{f(x^*)}$, where x^* is the optimal solution. The table also shows the ratio of the best solution ($r(x_b)$), the number of generations (gen), and the number of times the pareto list L was reduced to a single pareto ($t = 1$).

test problem		results obtained by the proposed GA					
pb	optimum	worst (x_w)	median (x_m)	best (x_b)	$r(x_b)$	gen	t=1
1	-15	-9	-12.994586	-14.99999	0.000%	224	23
2	7049.3309	7051.6170	7149.1049	7130.5802	0.032%	1762	50
3	680.63005	680.63019	680.63109	680.63055	0.000%	2234	93
4	0.0539498	0.0578312	0.4553916	0.0558078	3.444%	1330	121
5	24.306209	24.346224	24.528300	24.528300	0.164%	1800	40

Table 4.1: Results obtained with the proposed genetic algorithm.

Figures 4.3, 4.3, 4.3 and 4.3 depict the evolution of the best solutions in memory over the time for each of the test problems. The y-axis is the natural logarithm of the ratio $r(x)$ given in percentage ($\ln(1 + r(x))$). This function is nonnegative and the optimum is 0, obtained when $r(x) = 0$. The x-axis is the number of generations for the first two figures. Figure 4.3 shows the problems whose convergence was reached in less than 300 generations, while figure 4.3 shows the problems whose convergence was reached in about 1800 generations. The last two figures illustrate the natural logarithm of the ratio $r(x)$ as a function of CPU time given in hours. Figure 4.3 depicts the problems whose convergence was reached in less than an hour, while figure 4.3 depicts the problems whose convergence was reached in about 8 hours.

It is interesting to observe that solutions with ratio under 1.45% were found for test problems 1, 3 and 5 in less than 200 generations, while solutions with ratio under 5.68% were found for test problems 2 and 4 in less than 200 generations. If CPU time is considered, then solutions with ratio under 1.45% were found for test problems 3 and 5 in less than one hour, while solutions with ratio under 5.68% were found for test problems 2 and 4 in less than the same amount of time. A solution with ratio under 0.22% was found for test problem 4 in about four hours.

The test problems were also benchmark problems for the survey [329] which compared the genetic algorithms described in section 4.1.1. All those genetic algorithm were applied 10 times to each of the test problems. The size of the population was 70 trial solutions and the number of generations was 5,000 for each run. The best solution found among all GAs and their runs are available in table 4.2.

In the literature there are classical algorithms for minimization of unconstrained nonlinear programming problems without derivatives. Among these methods, it was applied the cyclic coordinate method (CCM) and the method of

Hooke and Jeeves (MHJ) to the test problems [48]. The CCM starts with a random solution and executes line search through the canonical directions. The best point found is made the current solution and the exploratory search is repeated until no improvement is achieved. The MHJ is a CCM with an additional linear search that is done through the line segment define by two successive solutions. To apply CCM and MHJ the test problems were converted to unconstrained problems by introducing penalties for violations. The search is restricted to the n-cubes defined by the bounds on the variables. The penalties to violations are 10^6. The line searches executed in each direction are two-phase. The first phase searches through the entire n-cube, while the second searches in the neighborhood of the best solution found in the first phase. CCM and MHJ were repeatedly applied to all test problem with starting random solutions. They were given the same amount of time given to the proposed GA. All results are summarized in table 4.2.

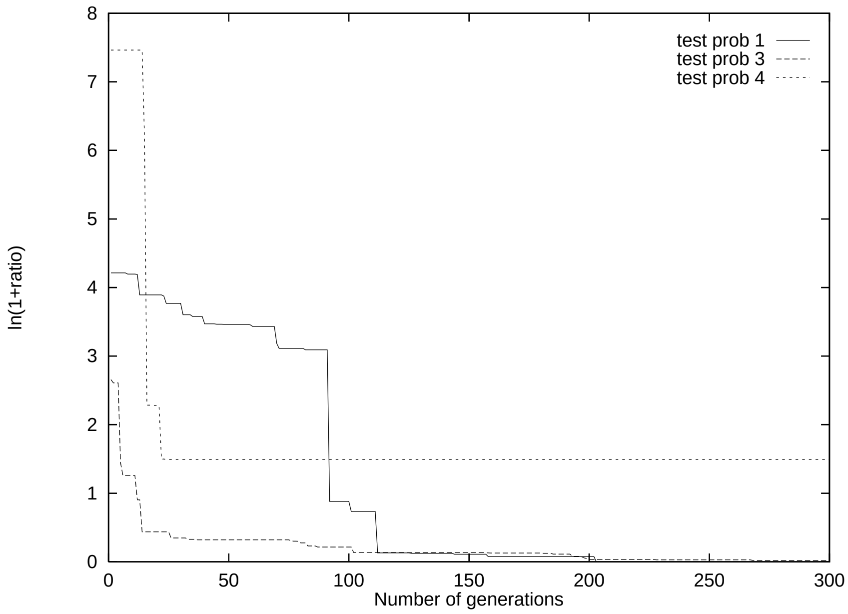

Figure 4.3: Best solution in memory over the time for test problems 1, 3, and 4. The x-axis is the number of generations and the y-axis is the natural logarithm of the ratio $r(x)$ given in percentage $(\ln(1 + r(x)))$.

Unfortunately, neither the number of objective funtion evaluations nor the CPU time spent by the GAs surveyed by Michalewicz are available. Therefore, the optimal solution found by the algorithm is the parameter adopted for comparison. Table 4.2 shows that the proposed GA achieved better results in all test problems, but test problem 4. Only one of them, method 4, found a solution for test problem 4 with better evaluation than the one found by the proposed GA. If we consider individually each of the genetic algorithm surveyed by Michalewicz, none of them outperformed the proposed GA when considering all test problems.

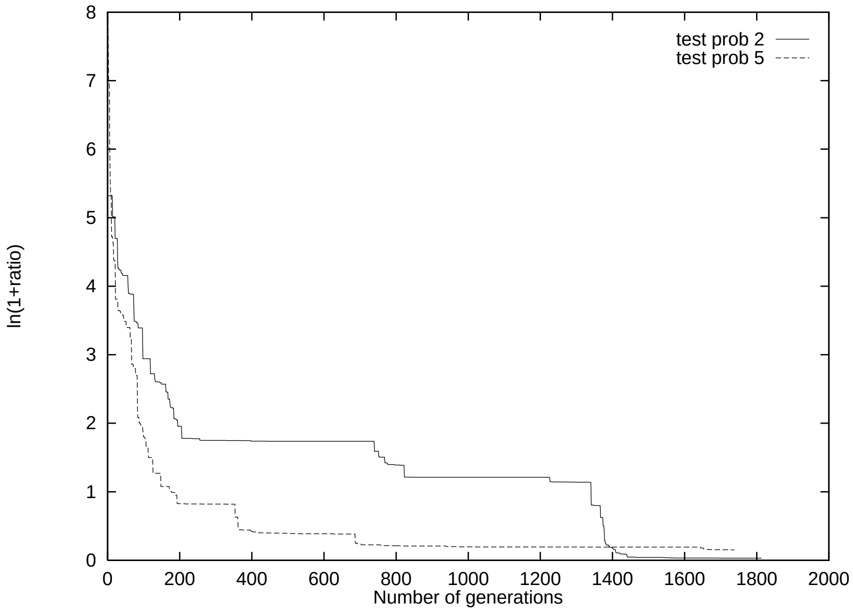

Figure 4.4: Best solution in memory over the time for test problems 2 and 5. The x-axis is the number of generations and the y-axis is the natural logarithm of the ratio $r(x)$ given in percentage $(\ln(1 + r(x)))$.

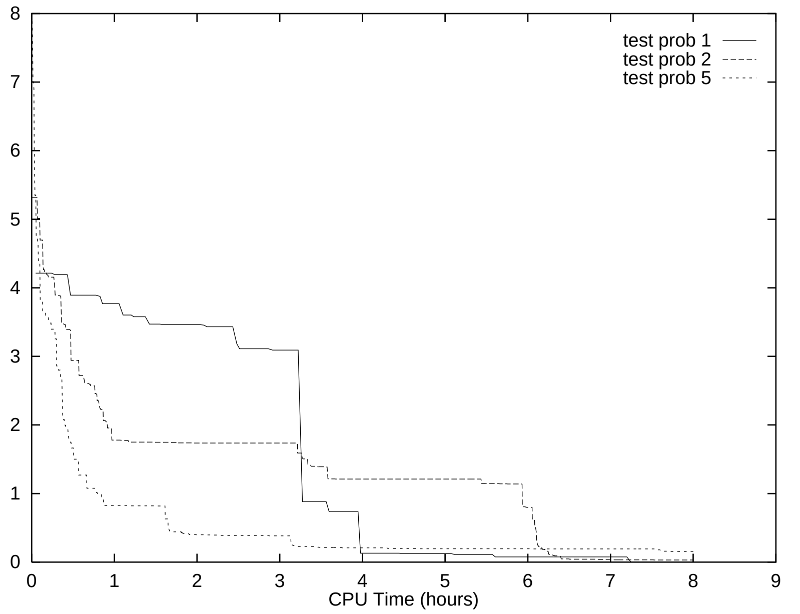

Figure 4.5: Best solution in memory over the time for test problems 1, 2, and 5. The x-axis is the CPU Time and the y-axis is the natural logarithm of the ratio $r(x)$ given in percentage $(\ln(1 + r(x)))$.

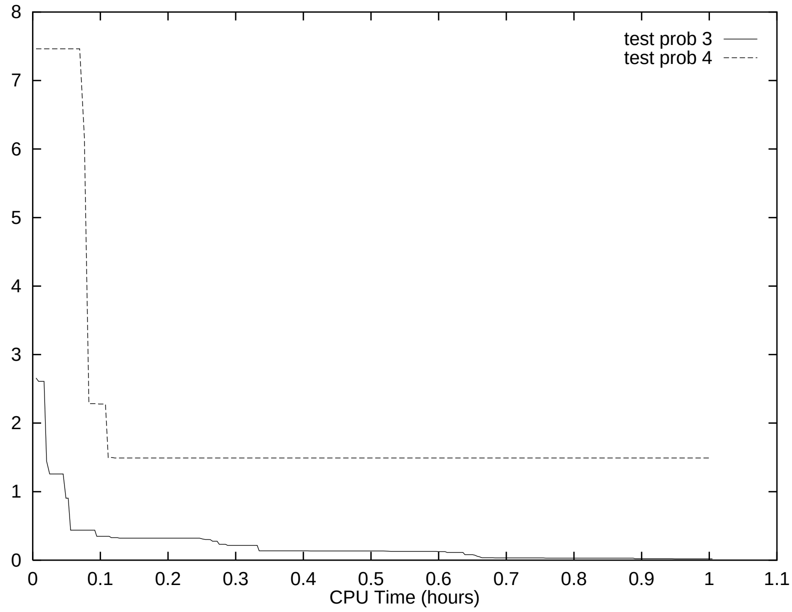

Figure 4.6: Best solution in memory over the time for test problems 3 and 4. The x-axis is the CPU Time and the y-axis is the natural logarithm of the ratio $r(x)$ given in percentage $(\ln(1 + r(x)))$.

	results				
problem	CCM	MHJ	reviewed GAs	proposed GA	best algorithm
1	0.000%	0.000%	0.000%	0.000%	all
2	22.080%	15.790%	4.66%	0.032%	proposed GA
3	0.509%	0.032%	0.001%	0.000%	proposed GA
4	1.259%	4.143%	0.092%	3.444%	other GA
5	16.670%	16.754%	4.853%	0.164%	proposed GA

Table 4.2: Results obtained by all algorithms. The results are given as ratio to the optimum.

The proposed GA relies on line search to generate offspring. It is interesting to find out whether or not line search methods can achieve comparable or better results. In other words, how much improvement does the Pareto Scheme provide? The results available in table 4.2 show that simple line search methods (CCM and MHJ) do not achieve results comparable to GAs. Nevertheless, line search achieved good results when embedded in the Pareto Scheme.

4.4 Conclusion and Future Research

We have demonstrated that the pareto scheme can achieve comparable if not better results than penalty methods and multidimensional searches when solving nonlinear programming problems. The pareto scheme also quickly found solutions of good quality. Besides the simplicity of the method, its extensibility to multiobjective problems is an interesting property that can be exploited. To solve a multiobjective problem one only needs to insert another objective function. The final solution provided by the algorithm is a pareto set (nondominated solutions) and not a single solution that maximizes the combined objective and penalty functions. A topic for future research is to consider not only the performance space to estimate improving directions, but the decision space as well. A possibility is to consider the dominance relation among trial solutions which belong to the same neighborhood.

Acknowledgements

We would like to thank Conselho Nacional de Pesquisa (CNPq/Brazil) for supporting this research under process of number 200047/96-5. We would also like to thank the anonymous referees for their comments and helpful suggestions.

4.5 Test Problems

4.5.1 Test Problem 1

`Minimize` $5x_1 + 5x_2 + 5x_3 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$

`Subject to`

$$
\begin{aligned}
&2x_1 + 2x_2 + x_{10} + x_{11} \leq 10\\
&2x_1 + 2x_3 + x_{10} + x_{12} \leq 10\\
&2x_2 + 2x_3 + x_{11} + x_{12} \leq 10\\
&-8x_1 + x_{10} \leq 0\\
&-8x_2 + x_{11} \leq 0\\
&-8x_3 + x_{12} \leq 0\\
&-2x_4 - x_5 + x_{10} \leq 0\\
&-2x_6 - x_7 + x_{11} \leq 0\\
&-2x_8 - x_9 + x_{12} \leq 0
\end{aligned}
$$

$$
\begin{array}{lllll}
0 & \leq & x_i & \leq & 1 \texttt{ for all } i = 1, \ldots, 9 \\
0 & \leq & x_i & \leq & 100 \texttt{ for all } i = 10, 11, 12 \\
0 & \leq & x_{13} & \leq & 1
\end{array}
$$

The global optimum is at $x^* = (1,1,1,1,1,1,1,1,1,3,3,3,1)$ and $f(x^*) = -15$ [145].

4.5.2 Test Problem 2

$$
\begin{array}{ll}
\texttt{Minimize} & x_1 + x_2 + x_3 \\
\texttt{Subject to} & \\
& 1 - 0.0025(x_4 + x_6) \geq 0 \\
& 1 - 0.0025(x_5 + x_7 - x_4) \geq 0 \\
& 1 - 0.01(x_8 - x_5) \geq 0 \\
& x_1 x_6 - 8333.33252 x_4 - 100 x_1 + 83333.333 \geq 0 \\
& x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0 \\
& x_3 x_8 - 1250000 - x_3 x_5 + 2500 x_5 \geq 0
\end{array}
$$

$$
\begin{array}{lllll}
100 & \leq & x_1 & \leq & 10000 \\
1000 & \leq & x_i & \leq & 10000 \texttt{ for all } i = 2, 3 \\
10 & \leq & x_i & \leq & 1000 \texttt{ for all } i = 4, \ldots, 8
\end{array}
$$

The global optimum is at

$$
\begin{aligned}
x^* = \; & (579.3167, 1359.943, 5110.071, 182.0174, \\
& 295.5985, 217.9799, 286.4162, 395.5979)
\end{aligned}
$$

and $f(x^*) = 7049.330923$ [209].

4.5.3 Test Problem 3

$$
\begin{array}{ll}
\texttt{Minimize} & (x_1 - 10)^2 + 5(x_1 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + \\
& 10 x_5^6 + 7 x_6^2 + x_7^4 - 4 x_6 x_7 - 10 x_6 - 8 x_7 \\
\texttt{Subject to} & \\
& 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0 \\
& 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0 \\
& 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0 \\
& -4x_1^2 - x_2^2 + 3x_1 x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0
\end{array}
$$

$$
-10 \leq x_i \leq 10 \texttt{ for all } i = 1, \ldots, 7
$$

The global optimum is at

$$
\begin{aligned}
x^* = \; & (2.330499, 1.951372, -0.4775414, \\
& 4.365726, -0.6244870, 1.038131, 1.594227)
\end{aligned}
$$

and $f(x^*) = 680.6300573$ [209].

4.5.4 Test Problem 4

$$\begin{array}{ll} \texttt{Minimize} & e^{x_1 x_2 x_3 x_4 x_5} \\ \texttt{Subject to} & \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10 \\ & x_2 x_3 - 5 x_4 x_5 = 0 \\ & x_1^3 + x_2^3 = -1 \end{array}$$

$$\begin{array}{rcccl} -2.3 & \leq & x_i & \leq & 2.3 \texttt{ for all } i = 1,2 \\ -3.2 & \leq & x_i & \leq & 3.2 \texttt{ for all } i = 3,4,5 \end{array}$$

The global optimum is at

$$\begin{array}{rcl} x^* & = & (-1.717143, 1.595709, 1.827247, \\ & & -0.736413, -0.7636450) \end{array}$$

and $f(x^*) = 0.0539498478$ [209].

4.5.5 Test Problem 5

$$\begin{array}{ll} \texttt{Minimize} & x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 \\ & 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + \\ & 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \\ \texttt{Subject to} & \\ & 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0 \\ & -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0 \\ & 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0 \\ & -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0 \\ & -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0 \\ & -x_1^2 - 2(x_2 - 2)^2 + 2x_1 x_2 - 14x_5 + 6x_6 \geq 0 \\ & -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0 \\ & 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0 \end{array}$$

$$-10.0 \quad \leq \quad x_i \quad \leq \quad 10.0 \texttt{ for all } i = 1, \ldots, 10$$

The global optimum is at

$$\begin{array}{rcl} x^* & = & (2.171996, 2.363683, 8.773926, 5.095984, \\ & & 0.9906548, 1.430574, 1.321644, 9.828726, \\ & & 8.280092, 8.375927) \end{array}$$

and $f(x^*) = 24.3062091$ [209].

4.6 Bibliography

See chapter bibliography at the end of this proceedings.