Using a New GA-based Multiobjective Optimization Technique for the Design of Robot Arms*

Carlos A. Coello Coello Alan D. Christiansen Arturo Hernández Aguirre

Department of Computer Science
Tulane University
New Orleans, LA 70118

Abstract: *This paper presents a hybrid approach to optimize the counterweight balancing of a robot arm. A new technique that combines the genetic algorithm (GA) and the weighted min-max multiobjective optimization method is proposed. This new approach is compared to several mathematical programming and GA-based techniques used for multiobjective optimization. These techniques are included in a system developed by the authors, called MOSES, which is intended to be used as a tool for engineering design optimization. The results presented here show how the new proposed technique can get better trade-off solutions and a more accurate Pareto front for this highly non-convex problem using an ad-hoc floating point representation and traditional genetic operators. Finally, a methodology to compute the ideal vector using a genetic algorithm is presented. It is shown how with a very simple dynamic approach to adjust the parameters of the GA, it is possible to obtain better results than those previously reported in the literature for this problem.*

1 Introduction

The use of industrial robots in different fields of technology is becoming more common every day, making it more important to be able to improve their efficiency in terms of energy consumption and working accuracy. The proper balancing of a robot manipulator is one way to improve such efficiency. There are two main methods of balancing a robot manipulator:[1] 1) by spring mechanisms, and 2) by counterweights. The second approach, which is the one selected for this work, has been frequently used in the literature for establishing better mass distributions of mechanisms and its use on robot manipulators involves the minimization of driving forces or torques as well as the support reactions at joints. Since these two criteria have to be satisfied at the same time, a multiobjective optimization approach has to be taken. The lengths and masses of balancing mechanisms of the robot arm are used as design variables, and several constraints derived from the allowable movements of the arm are imposed. The optimization model used for this work is based on the rigid-body dynamics of the PUMA-560 robot[2].[3] A hybrid approach was used to solve this problem, using a combination of a genetic algorithm with the min-max method to get the Pareto optimal set, which corresponds to several possible robot designs from which the decision maker has to choose the most appropriate. This set was obtained by varying the importance of each of the four objective functions derived from the optimization model—two torques and two reactions—. This new approach is compared to a more traditional min-max technique in which a combination of random and sequential search is used to generate the Pareto optimal solutions. This problem has a highly non-convex search space, which implies the presence of several local minima. On the other hand, the large amount of CPU time required to evaluate the different objectives arise some interesting issues on the use of genetic algorithms in this kind of application.

*This work was supported in part by EPSCoR grant: NSF/LEQSF (1992-93)-ADP-04.

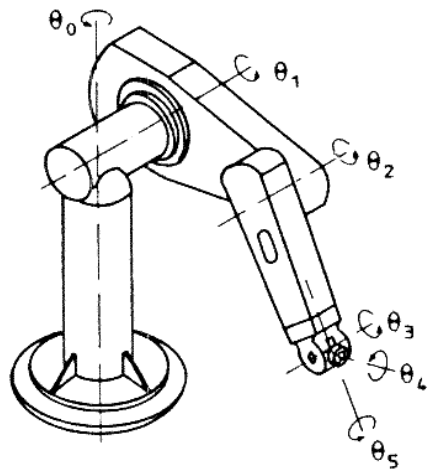

Figure 1: PUMA-560 robot arm and schematic representation of coordinate angles θ_i.

2 Statement of the Problem

Consider the PUMA-560 shown in Figure 1. Koski and Osyczka[1] present a multiobjective optimization model of such arm based on its rigid-body dynamics. By using angular coordinates for the PUMA-560 robot, it is possible to calculate the generalized torques at each joint applying the following equation:

$$M_{ti} = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) - \frac{\partial L}{\partial \theta_i} \tag{1}$$

where θ_i is the rotation at joint i and $\dot{\theta}_i$ is the corresponding angular velocity. The term

$$L = T - V \tag{2}$$

represents the Lagrangian function of the mechanical system. Here, T is the total kinetic energy of the system and V is the total potential energy. The application of Equation (1) to a fully articulated robot arm results in the following nonlinear second-order system of differential equations

$$A\ddot{\theta} + B\dot{\theta}^2 + c - m = 0 \tag{3}$$

Here, the vector of angular accelerations is given by

$$\ddot{\theta} = (\ddot{\theta}_1, \ddot{\theta}_2, \ldots, \ddot{\theta}_N)^T \tag{4}$$

and the vector of squared angular velocities by

$$\dot{\theta}^2 = (\dot{\theta}_1\dot{\theta}_1, \dot{\theta}_1\dot{\theta}_2, \ldots, \dot{\theta}_1\dot{\theta}_N | \dot{\theta}_2\dot{\theta}_1, \dot{\theta}_2\dot{\theta}_2, \ldots, \dot{\theta}_2\dot{\theta}_N | \ldots \dot{\theta}_k\dot{\theta}_1, \dot{\theta}_k\dot{\theta}_2, \ldots, \dot{\theta}_k\dot{\theta}_N | \ldots \dot{\theta}_N\dot{\theta}_1, \dot{\theta}_N\dot{\theta}_2, \ldots, \dot{\theta}_N\dot{\theta}_N)^T$$

where N is the number of joints. The corresponding matrices are

$$A = \begin{bmatrix} D_{11} & D_{12} & \ldots & D_{1N} \\ D_{21} & D_{22} & \ldots & D_{2N} \\ \vdots & & & \\ D_{N1} & D_{N2} & \ldots & D_{NN} \end{bmatrix} \tag{5}$$

and

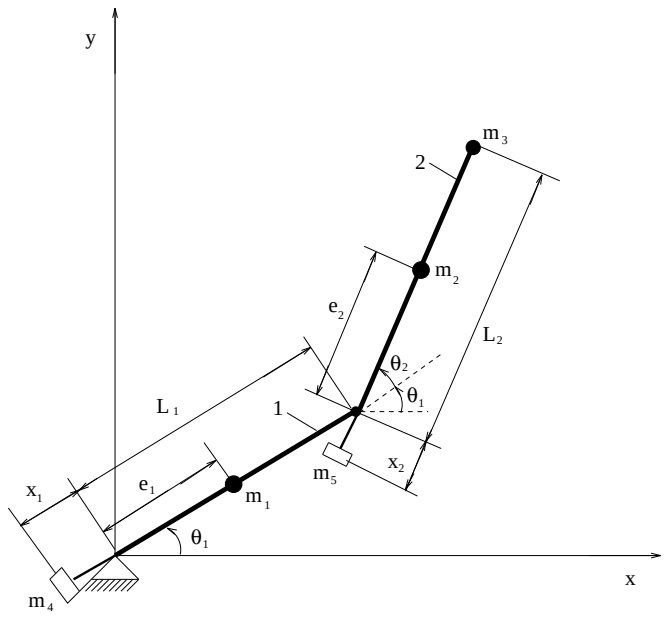

Figure 2: Mechanical model of the robot arm used for optimization.

$$B = \left[\begin{array}{cccc|cccc} D^1_{11} & D^1_{12} & \dots & D^1_{1N} & D^2_{11} & D^2_{12} & \dots & D^2_{1N} \\ D^1_{21} & D^1_{22} & \dots & D^1_{2N} & D^2_{21} & D^2_{22} & \dots & D^2_{2N} \\ & \vdots & & & & \vdots & & \\ D^1_{N1} & D^1_{N2} & \dots & D^1_{NN} & D^2_{N1} & D^2_{N2} & \dots & D^2_{NN} \\ & & & & D^N_{11} & D^N_{12} & \dots & D^N_{1N} \\ & & & & D^N_{21} & D^N_{22} & \dots & D^N_{2N} \\ & & & & & \vdots & & \\ & & & & D^N_{N1} & D^N_{N2} & \dots & D^N_{NN} \end{array}\right]$$

where it is assumed that each joint has one degree of freedom. The elements of matrix A are the inertia terms, and the elements of matrix B represent the centripetal and Coriolis terms. All these terms depend on the position of the arm—i.e., $D_{ij} = D_{ij}(\theta_i)$—. Vector $c = (D_1, D_2, \dots, D_N)^T$ includes the gravitational terms D_i and $m = (m_1, m_2, m_3, m_4, m_5)$ is the vector of torques. Kinetic Equations (1),(2) and (3) represent the rigid-body motion of the arm, and they are geometrically nonlinear because of large rotations of θ_i.

The manipulator is an isostatic structure, and thus it is possible to get explicit expressions for all forces and moments in the system. The friction in the joints as well as the flexibility of the arm are not included in the following design model. For the application of optimization methods, a two-member robot arm, which corresponds to the two links of the PUMA-560 robot in a plane motion, is considered. This arm is assumed to move in the xy-plane only (corresponding angular coordinates θ_i are shown in Figure 2). The masses of the members are m_1 and m_2. They are located as point masses at distances e_1 and e_2 from the joints. The external load is represented by the point mass m_3. In the model used by Koski and Osyczka, only the counterweight masses m_4 and m_5, as well as their distances from the joints x_1 and x_2 are treated as design variables, whereas all the other quantities are fixed.

The torques of this two-member link are obtained from the Equation (1) and are expressed as follows:

$$\begin{aligned} M_{t1} &= D_{11}\ddot{\theta}_1 + D_{12}\ddot{\theta}_2 + D^1_{11}\dot{\theta}_1^2 + D^2_{12}\dot{\theta}_2^2 + (D^1_{12} + D^2_{11})\dot{\theta}_1\dot{\theta}_2 + D_1 \\ M_{t2} &= D_{21}\ddot{\theta}_1 + D_{22}\ddot{\theta}_2 + D^1_{21}\dot{\theta}_1^2 + D^2_{22}\dot{\theta}_2^2 + (D^1_{22} + D^2_{21})\dot{\theta}_1\dot{\theta}_2 + D_2 \end{aligned} \tag{6}$$

The coefficients for torque M_{t1} are:

$$\begin{aligned} D_{11} = m_1 e_1^2 + m_4 x_1^2 + m_2 e_2^2 + m_3 L_2^2 + m_5 x_2^2 + \\ +(m_2 + m_3 + m_5)L_1^2 + 2m_2 e_2 L_1 \cos\theta_2 + 2m_3 L_1 L_2 \cos\theta_2 - 2m_5 x_2 L_1 \cos\theta_2 + J_1 + J_2 \end{aligned}$$

$$D_{12} = m_2 e_2^2 + m_3 L_2^2 + m_5 x_2^2 + m_2 e_2 L_1 \cos\theta_2 + m_3 L_1 L_2 \cos\theta_2 - m_5 x_2 L_1 \cos\theta_2 + J_2$$

$$D^1_{11} = -m_5 x_2 L_1[\sin(\theta_0 + \theta_1) + \sin\theta_2] + m_5 x_2^2 \sin 2\theta_0$$

$$D^2_{12} = -m_2 e_2 L_1 \sin\theta_2 - m_3 L_1 L_2 \sin\theta_2 - m_5 x_2 L_1 \sin(\theta_0 + \theta_1) + m_5 x_2^2 \sin 2\theta_0$$

$$D^1_{12} + D^2_{11} = -2m_2 e_2 L_1 \sin\theta_2 - 2m_3 L_1 L_2 \sin\theta_2 - 2m_5 x_2 L_1 \sin(\theta_0 + \theta_1) + 2m_5 x_2^2 \sin 2\theta_0$$

$$D_1 = m_1 g e_1 \cos\theta_1 - m_4 g x_1 \cos\theta_1 + m_2 g e_2 \cos\theta_0 + m_3 g L_2 \cos\theta_0 - m_5 g x_2 \cos\theta_0 + (m_2 + m_3 + m_5) g L_1 \cos\theta_1$$

Coefficients for torque M_{t2} are:

$$D_{21} = m_2(L_1 e_2 \cos\theta_2 + e_2^2) + m_3(L_1 L_2 \cos\theta_2 + L_2^2) - m_5(L_1 x_2 \cos\theta_2 - x_2^2) + J_2$$

$$D_{22} = m_2 e_2^2 + m_3 L_2^2 + m_5 x_2^2 + J_2$$

$$D^1_{21} = m_2 L_1 e_2 \sin\theta_2 + m_3 L_1 L_2 \sin\theta_2 + m_5(2x_2^2 \sin\theta_0 \cos\theta_0 - L_1 x_2 \sin\theta_2)$$

$$D^2_{22} = 2m_5 x_2^2 \sin\theta_0 \cos\theta_0 \tag{7}$$

$$D^1_{22} + D^2_{21} = 4m_5 x_2^2 \sin\theta_0 \cos\theta_0 \tag{8}$$

$$D_2 = m_2 g e_2 \cos\theta_0 + m_3 g L_2 \cos\theta_0 - m_5 g x_2 \cos\theta_0 \tag{9}$$

where J_1 and J_2 are the rotary inertias of members 1 and 2, respectively. Notation $\theta_0 = \theta_1 + \theta_2$ is used for convenience.

In addition to the torques, the joint forces are considered in the optimization process. In this application the most convenient way of solving them is to use the force equilibrium conditions in both coordinate directions x and y. For this purpose, the free-body diagrams of both members have been depicted in Figure 3. The

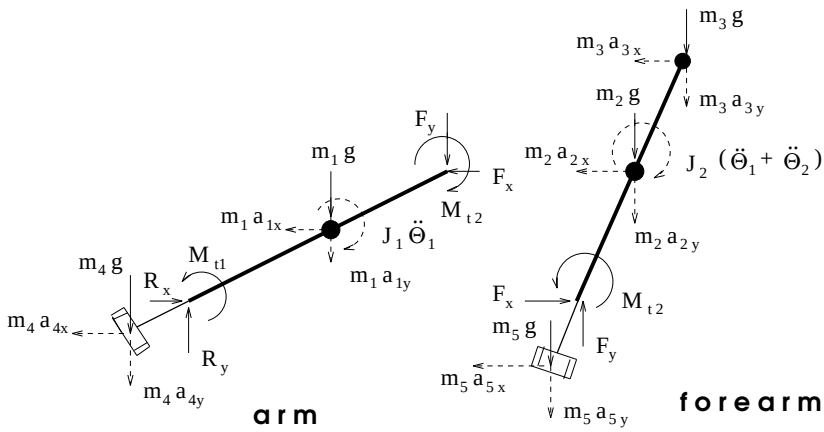

Figure 3: Free-body diagrams of the robot arm.

positive directions in this figure are associated with the global xy-axes, and the positive rotation direction is counterclockwise. By computing the accelerations from the well-known kinematic equation

$$a_p = a_Q + \alpha \times r_{p/Q} + \omega \times (\omega \times r_{p/Q}) \tag{10}$$

analytic expressions for $a_i(1, \ldots, 5)$ can be obtained. Here, a_Q is the acceleration vector of the comparison point, α the angular acceleration vector of the member, $r_{p/Q}$ the position vector from point Q to point P along the member, and ω the angular velocity of the member. For member 1, point Q is the support point and $\alpha = (0, 0, \ddot{\theta}_1)^T$, $\omega = (0, 0, \dot{\theta}_1)^T$. For member 2, point Q is at the joint and $\alpha = (0, 0, \ddot{\theta}_0)^T$, $\omega = (0, 0, \dot{\theta}_0)^T$ where $\theta_0 = \theta_1 + \theta_2$. Vector $r_{p/Q}$ depends on the 5 selected calculation points. Here, the detailed expressions for the accelerations are presented separately, and they also appear as part of the terms in Equation (6). The corresponding inertia forces $m_i a_i$ $(i = 1, \ldots, 5)$ and the moments $J_j \alpha_j$ $(j = 1, 2)$ with the complete free-body diagrams are shown in Figure 3.

The accelerations of the points at which the point masses are located have the following explicit expressions:

$$a_1 = \ddot{\theta}_1 e_1 \begin{bmatrix} -\sin\theta_1 \\ \cos\theta_1 \end{bmatrix} - \dot{\theta}_1^2 e_1 \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} \tag{11}$$

$$a_2 = \ddot{\theta}_1 L_1 \begin{bmatrix} -\sin\theta_1 \\ \cos\theta_1 \end{bmatrix} - \dot{\theta}_1^2 L_1 \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} + e_2 \ddot{\theta}_0 \begin{bmatrix} -\sin\theta_0 \\ \cos\theta_0 \end{bmatrix} + e_2 \dot{\theta}_0^2 \begin{bmatrix} -\cos\theta_0 \\ -\sin\theta_0 \end{bmatrix}$$

$$a_3 = \ddot{\theta}_1 L_1 \begin{bmatrix} -\sin\theta_1 \\ \cos\theta_1 \end{bmatrix} - \dot{\theta}_1^2 L_1 \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} + L_2 \ddot{\theta}_0 \begin{bmatrix} -\sin\theta_0 \\ \cos\theta_0 \end{bmatrix} + L_2 \dot{\theta}_0^2 \begin{bmatrix} -\cos\theta_0 \\ -\sin\theta_0 \end{bmatrix}$$

$$a_4 = \ddot{\theta}_1 x_1 \begin{bmatrix} \sin\theta_1 \\ -\cos\theta_1 \end{bmatrix} + \dot{\theta}_1^2 x_1 \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} \tag{12}$$

$$a_5 = \ddot{\theta}_1 L_1 \begin{bmatrix} -\sin\theta_1 \\ \cos\theta_1 \end{bmatrix} - \dot{\theta}_1^2 L_1 \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} + x_2 \ddot{\theta}_0 \begin{bmatrix} \sin\theta_0 \\ \cos\theta_0 \end{bmatrix} + x_2 \dot{\theta}_0^2 \begin{bmatrix} \cos\theta_0 \\ -\sin\theta_0 \end{bmatrix}$$

Here again the notations $\theta_0 = \theta_1 + \theta_2$, $\dot{\theta}_0 = \dot{\theta}_1 + \dot{\theta}_2$ and $\ddot{\theta}_0 = \ddot{\theta}_1 + \ddot{\theta}_2$ have been used. By applying the force equilibrium conditions in the coordinate directions, the following joint reactions (see Figure 3) to member 2 are obtained:

$$
\begin{aligned}
R_{2x} &= m_2 a_{2x} + m_3 a_{3x} + m_5 a_{5x} \\
R_{2y} &= m_2 a_{2y} + m_3 a_{3y} + m_5 a_{5y} + (m_2 + m_3 + m_5) g
\end{aligned} \tag{13}
$$

The torque M_{t2} at the joint can be calculated from the moment equilibrium condition. By applying the same routine to member 1, it is possible to derive expressions for the support reactions:

$$
\begin{aligned}
R_{1x} &= m_1 a_{1x} + m_2 a_{2x} + m_3 a_{3x} + m_4 a_{4x} + m_5 a_{5x} \\
R_{1y} &= m_1 a_{1y} + m_2 a_{2y} + m_3 a_{3y} + m_4 a_{4y} + m_5 a_{5y} + \\
&\quad (m_1 + m_2 + m_3 + m_4 + m_5) g
\end{aligned} \tag{14}
$$

The torque M_{t1} is obtained again from the moment equilibrium condition. Corresponding to the Lagrangian approach, the torques M_{t1} and M_{t2} must be the same as those computed from Equation (6). The resulting support reactions are:

$$
R_1 = (R_{1x}^2 + R_{1y}^2)^{\frac{1}{2}} \quad , \quad R_2 = (R_{2x}^2 + R_{2y}^2)^{\frac{1}{2}} \tag{15}
$$

The torques M_{ti} and the forces R_i are chosen as criteria in the optimization model. It is important to present the detailed expressions for M_{ti} and R_i because the choice of the design variables as well as the general complexity of the optimization problem are associated with these formulas.

Given the previous information, the optimization problem can now be formulated. The objective is to find such masses m_4 and m_5 for the counterweights and such joint distances x_1 and x_2 which will minimize the chosen four design criteria. Consequently, the design variable vector is:

$$
\bar{x} = (x_1, x_2, x_3, x_4)^T \tag{16}
$$

where the first two are the distances shown in Figure 2, $x_3 = m_4$ and $x_4 = m_5$. The upper and lower limits for all these four design variables can be given in the form

$$
x_i^l \leq x_i \leq x_i^u \quad , \quad i = 1, \ldots, 4 \tag{17}
$$

The torques M_{t1} and M_{t2} at the arm joints are chosen as the first two criteria of the vector objective function. Their minimization is important because it is then possible to use smaller motors, and the energy consumption is lower if the variation ranges of the torques are small.[1] In the explicit expressions of Equation (6), terms $m_4 x_1$, $m_4 x_1^2$, $m_5 x_2$, and $m_5 x_2^2$ appear, and thus it is reasonable to choose the design variables in the way presented.

The torques do not depend on the design variables alone, but also on the position of the robot arm (θ_1, θ_2). On the angular velocities $(\dot{\theta}_1, \dot{\theta}_2)$ and on the angular accelerations $(\ddot{\theta}_1, \ddot{\theta}_2)$. Usually, the working space of the robot arm is restricted, and thus constraints of the form:

$$
\theta_i^l \leq \theta_i \leq \theta_i^u \quad , \quad i = 1, 2 \tag{18}
$$

are needed. Here, θ_i^l and θ_i^u are the lower and the upper limits of the angles θ_i. In each position of the arm, the angular velocities and accelerations may be different. In order to optimize the performance of the robot, the torques should be as small as possible at all working positions and at all existing angular velocity acceleration combinations. Thus, the first two criteria are chosen as follows:

$$
\begin{aligned}
f_1(\bar{x}) &= \max_{\theta_1} \max_{\theta_2} \max_{\dot{\theta}_i, \ddot{\theta}_i} M_{t1} \\
f_2(\bar{x}) &= \max_{\theta_1} \max_{\theta_2} \max_{\dot{\theta}_i, \ddot{\theta}_i} M_{t2}
\end{aligned} \tag{19}
$$

where notation $\dot{\theta}_i$, $\ddot{\theta}_i$ is associated with the chosen angular velocity profile. This is shown in Figure 4 where a trapezoidal profile, typical of robot applications, has been depicted for both members. The corresponding angular accelerations are also presented in this Figure.

The construction of joints, especially with the choice of bearings, depends largely on the reaction forces at the joints. Thus, it seems reasonable to choose the maximum values of the joint forces as two additional criteria. By

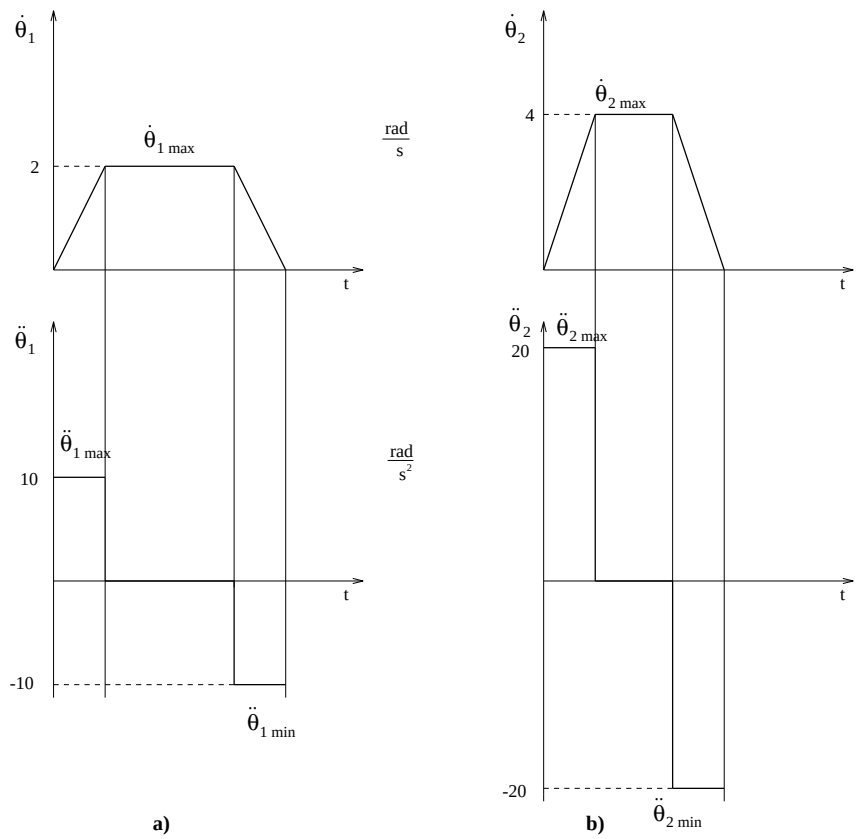

Figure 4: Angular velocities and corresponding angular accelerations of the robot arm.

Point	$\dot{\theta}_1$	$\dot{\theta}_2$	$\ddot{\theta}_1$	$\ddot{\theta}_2$
1	0	0	$\ddot{\theta}_{1\ max}$	$\ddot{\theta}_{2\ max}$
2	$\frac{1}{2}\ \dot{\theta}_{1\ max}$	$\frac{1}{2}\ \dot{\theta}_{2\ max}$	$\ddot{\theta}_{1\ max}$	$\ddot{\theta}_{2\ max}$
3	$\dot{\theta}_{1\ max}$	$\dot{\theta}_{2\ max}$	$\ddot{\theta}_{1\ max}$	$\ddot{\theta}_{2\ max}$
4	$\dot{\theta}_{1\ max}$	$\dot{\theta}_{2\ max}$	0	0
5	$\dot{\theta}_{1\ max}$	$\dot{\theta}_{2\ max}$	$-\ddot{\theta}_{1\ max}$	$-\ddot{\theta}_{2\ max}$
6	$\frac{1}{2}\ \dot{\theta}_{1\ max}$	$\frac{1}{2}\ \dot{\theta}_{2\ max}$	$-\ddot{\theta}_{1\ max}$	$-\ddot{\theta}_{2\ max}$
7	0	0	$-\ddot{\theta}_{1\ max}$	$-\ddot{\theta}_{2\ max}$

Table 1: Angular velocities and corresponding angular accelerations of the robot arm

using the fixed trapezoidal velocity profiles (see Figure 4) and every feasible position of the arm, these criteria can be expressed in the form

$$\begin{aligned} f_3(\bar{x}) &= \max_{\theta_1} \max_{\theta_2} \max_{\dot{\theta}_i, \ddot{\theta}_i} R_1 \\ f_4(\bar{x}) &= \max_{\theta_1} \max_{\theta_2} \max_{\dot{\theta}_i, \ddot{\theta}_i} R_2 \end{aligned} \tag{20}$$

The geometrical interpretation of all these four criteria is the following:[1] a small movement at every position (θ_1, θ_2) of the arms is performed by using the fixed trapezoidal profiles $(\dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2$ shown in Figure 4), and the maximum values of the torques and the joint forces during the movement are selected.

By using the design variables x_i given in Equation (3), the criteria presented in Equations (19) and (20), the side constraints of (17), and the state constraints of (18), it is now possible to formulate the multiobjective optimization problem:[1]

$$min \quad (f_1(\bar{x}), f_2(\bar{x}), f_3(\bar{x}), f_4(\bar{x}))^T \tag{21}$$

subject to

$$\begin{aligned} \theta_i^l \leq \theta_i \leq \theta_i^u \quad & i = 1, 2 \\ x_i^l \leq x_i \leq x_i^u \quad & i = 1, 2, 3, 4 \end{aligned} \tag{22}$$

The numerical design data for the design problem is given below.[1] These values are close to those for the first two links of the PUMA-560 robot shown in Figure 1.[3]

$$\begin{gathered} m_1 = 17\ kg, \quad m_2 = 6\ kg, \quad m_3 = 2\ kg, \\ L_1 = L_2 = 0.43\ m, \quad e_1 = 0.07\ m, \quad e_2 = 0.14\ m, \\ \theta_1^l = -40^\circ, \quad \theta_1^u = 220^\circ, \quad \theta_2^l = -140^\circ, \quad \theta_2^u = 140^\circ, \\ \dot{\theta}_{1\ max} = 2\ \frac{rad}{s}, \quad \dot{\theta}_{2\ max} = 4\ \frac{rad}{s}, \\ \ddot{\theta}_{1\ max} = 10\ \frac{rad}{s^2}, \quad \ddot{\theta}_{2\ max} = 20\ \frac{rad}{s^2}, \\ x_1^l = x_2^l = 0, \quad x_1^u = x_2^u = 0.2\ m, \quad x_3^l = x_4^l = 0, \\ x_3^u = 35\ kg, \quad x_4^u = 15\ kg, \\ J_1 = 0.2619\ kg - m^2, \quad J_2 = 0.0924\ kg - m^2 \end{gathered} \tag{23}$$

3 Solution Procedure

To obtain the term $\max_{\theta_1}(\cdot)$, the procedure given by Koski and Osyckza[1] was followed:

1. Compute the torques and joint forces at the positions $\theta_1^l, \theta_1^l + \Delta\theta_1, \theta_1^l + 2\Delta\theta_1, \ldots, \theta_1^u$, where the increment $\Delta\theta_1$ was chosen to be 20 degrees.

2. Select separately the maximum value for each criterion.

3. Perform the same calculations for $\underset{\theta_2}{max}(\cdot)$ with an increment $\Delta\theta_2$ (a value of 20 degrees was used).

4. The terms $\underset{\dot{\theta}_i \ddot{\theta}_i}{max}(\cdot)$ are computed using some chosen combinations of $\dot{\theta}_i$ and $\ddot{\theta}_i$ for given θ_1 and θ_2. Table 1 contains the seven points chosen for the calculations presented in this paper.

5. After calculating M_{ti} and R_i for all the rows of Table 1, the maximum values can be chosen.

Obviously, the smaller the value of $\Delta\theta_i$ the better the accuracy achieved, but also greater the computation time required. It was experimentally found that even an apparently large increment like the one used in this work, did not significantly affect the final result. However, in terms of time, this value made a great difference, particularly when considering that using either random search or the genetic algorithm, all these computations have to be performed a lot of times. To get an idea of the importance of this parameter, when an increment of one degree is used, the time required to get one set of results (i.e., the final values of the four objective functions) is of about 2 minutes and 20 seconds on a Sun Workstation with four 90 MHz HyperSparc CPUs. This time is reduced to only one second when using increments of 20 degrees, without any significant loss of precision (normally the differences were in the decimals).

3.1 The Classical Min-Max Method

In the classical min-max method, also known as the *Global Criterion method*,[4] an optimal solution is a vector of decision variables which minimizes some global criterion. A function describing this global criterion is a measurement of how close the decision maker can get to the ideal vector—i.e., the vector that contains the optimal solutions of every objective function assuming that these were treated independently—, which will be denoted by f^0. The most common form of this function is

$$f(x) = \sum_{i=1}^{k} \left(\frac{f_i^0 - f_i(x)}{f_i^0} \right)^p \tag{24}$$

where k is the number of objective functions.

For this formula Boychuck and Ovhinnikov[5] have suggested $p = 1$, and Salukvadze[6] has suggested $p = 2$, but other values of p can also be used. Another possible measurement of "the closeness to the ideal solution" is a family of the L_p-metrics defined as follows[7]

$$L_p(f) = \left(\sum_{i=1}^{k} |f_i^0 - f_i(x)|^p \right)^{1/p} \qquad 1 \leq p \leq \infty \tag{25}$$

Instead of deviations in the absolute sense, it is recommended to use in Equation (25) relative deviations[47] such as

$$\frac{f_i^0 - f_i(x)}{f_i^0} \tag{26}$$

which have a direct substantive meaning in any given context.

The name min-max method is given to the global criterion method with the $L_\infty(f)-$metric, because for this metric the optimum x^* is defined as

$$f(x^*) = min_x \; max_i \left| \frac{f_i^0 - f_i(x)}{f_i^0} \right| \qquad i = 1, \ldots, k \tag{27}$$

The solution to this optimization problem yields the best compromise solution, in which all criteria are considered equally important. The use of weighting coefficients has been introduced before[7] in conjunction with this method to rank the importance of the candidate criterion, so that the min-max problem can be restated as follows

$$f(x^*) = min_x \; max_i \; \omega_i \left| \frac{f_i^0 - f_i(x)}{f_i^0} \right| \quad i = 1, \ldots, k \tag{28}$$

where ω_i is the weighting coefficient representing the relative importance of the ith criterion. Koski and Osyczka[1] took this approach to solve the counterweight balancing problem presented in this paper, by using the Computer Aided Multicriteria Optimization System (CAMOS).[8] They used a method which combines random and sequential search to generate the Pareto-optima. First, they generated some points by the random search method, and the best of them were stored and used as the starting points for the sequential search procedure. Then, they minimized each objective separately, to obtain the set of optimal solutions, so that they could use the weighting min-max method described above for generating several Pareto-optimal solutions. The weights were chosen so that their sum were always equals to one. While seeking both, the ideal vector and the other Pareto-optima, they used the random search method in combination with the Nelder-Mead simplex method[9] with a penalty function.

4 Notions of Genetic Algorithms

The famous naturalist Charles Darwin defined *Natural Selection* or *Survival of the Fittest* in his book[10] as the *preservation of favorable individual differences and variations, and the destruction of those that are injurious.* In nature, individuals have to adapt to their environment in order to survive in a process called *evolution*, in which those features that make an individual more suited to compete are preserved when it reproduces, and those features that make it weaker are eliminated. Such features are controlled by units called **genes** which form sets called **chromosomes**. Over subsequent generations not only the fittest individuals survive, but also their fittest genes which are transmitted to their descendants during the sexual recombination process which is called **crossover**.

John H. Holland became interested in the application of natural selection to machine learning, and in the late 60s, while working at the University of Michigan, he developed a technique that allowed computer programs to mimic the process of evolution. Originally, this technique was called **reproductive plans**, but the term **genetic algorithm** became popular after the publication of his book[11].[12]

In 1989, Goldberg published a book[13] that provided a solid scientific basis for this area, and cited no less than 73 successful applications of the genetic algorithm. In the last few years the growing interest on this technique is reflected in a larger number of conferences, a new international journal, and an increasing amount of software and literature devoted to this subject.

Koza[14] provides a good definition of a GA:

> The **genetic algorithm** is a highly parallel mathematical algorithm that transforms a set (**population** of individual mathematical objects (typically fixed-length character strings patterned after chromosome strings), each with an associated **fitness** value, into a new population (i.e., the next **generation**) using operations patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally occurring genetic operations (notably sexual recombination).

Actually, the genetic algorithm derives its behavior from a metaphor of one of the mechanisms of evolution in nature which is called **hard selection**.[15] Under this scheme, only the best available individuals are retained for generating descendants. This contrasts with **soft selection**, which offers a probabilistic mechanism for maintaining individuals to be parents of future progeny despite possessing relatively poorer objective values.

It has been argued[15] that the term **genetic algorithm** (GA) is misleading, since natural selection is only one of the mechanisms of evolution, and it would be more appropriate to call them **hard selection** (HS) algorithms to reflect the fact that they deal with only that particular selection scheme. However, the term is so common today, that a change does not seem feasible, at least in the near future.

A genetic algorithm for a particular problem must have the following five components:[16]

1. A representation for potential solutions to the problem.
2. A way to create an initial population of potential solutions.
3. An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness".
4. Genetic operators that alter the composition of children.
5. Values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

Some of the basic terminology used by the genetic algorithms (GAs) community is the following:[15]

- A **chromosome** is a data structure that holds a "string" of task parameters, or genes. This string may be stored, for example, as a binary bit-string (binary representation) or as an array of integers (floating point o real-coded representation) that represent a floating point number. This chromosome is analogous to the base-4 chromosomes present in our own DNA. Normally, in the GA community, the haploid model of a cell is assumed (one-chromosome individuals). However, diploids have also been used in the past.[13]
- A **gene** is a subsection of a chromosome that usually encodes the value of a single parameter.
- An **allele** is the value of a gene. For example, for a binary representation each gene may have an allele of 0 or 1, and for a floating point representation, each gene may have an allele from 0 to 9.
- If the solution of a problem can be represented by a set of N real-valued parameters, then the job of finding this solution can be thought of as a search in an N-dimensional space. This region is simply referred as the **search space** of the problem.
- The **fitness** of an individual is a value that reflects its performance (i.e., how well solves a certain task). A **fitness function** is a mapping of the chromosomes in a population to their corresponding fitness values. A **fitness landscape** is the hypersurface obtained by applying the fitness function to every point in the search space.
- **Exploitation** is the process of using information gathered from previously visited points in the search space to determine which places might be profitable to visit next. Hill climbing is an example of exploitation, because it investigates adjacent points in the search space, and moves in the direction giving the greatest increase in fitness. Exploitation techniques are good at finding local minima (or maxima). The GA uses crossover as an exploitation mechanism.
- **Exploration** is the process of visiting entirely new regions of a search space, to see if anything promising may be found there. Unlike exploitation, exploration involves leaps into unknown regions. Random search is an example of exploration. Problems which have many local minima (or maxima) can sometimes only be solved using exploration techniques such as random search. The GA uses mutation as an exploration mechanism.
- A **genotype** represents a potential solution to a problem, and is basically the string of values chosen by the user, also called chromosome.
- A **phenotype** is the meaning of a particular chromosome, defined externally by the user.
- Genetic drift is the name given to the changes in gene/allele frequencies in a population over many generations, resulting from chance rather than from selection. It occurs most rapidly in small populations and can lead to some alleles to become extinct, thus reducing the genetic variability in the population.
- A **niche** is a group of individuals which have similar fitness. Normally in multiobjective and multimodal optimization, a technique called **sharing** is used to reduce the fitness of those individuals who are in the same niche, in order to prevent the population to converge to a single solution, so that stable sub-populations can be formed, each one corresponding to a different objective or peak (in a multimodal optimization problem) of the function.

The basic operation of a Genetic Algorithm is illustrated in the following segment of pseudo-code:[17]

```
generate initial population, G(0);
evaluate G(0);
t:=0;
repeat
   t:=t+1;
   generate G(t) using G(t-1);
   evaluate G(t);
until a solution is found
```

First, an initial population is randomly generated. The individuals of this population will be a set of chromosomes or strings of characters (letters and/or numbers) that represent all the possible solutions to the problem. We apply a **fitness function** to each one of these chromosomes in order to measure the quality of the solution encoded by the chromosome. Knowing each chromosome's fitness, a **selection** process takes place to choose the individuals (presumably, the fittest) that will be the parents of the following generation. The most commonly used selection schemes are the following:[18]

- **Proportionate Reproduction:** This term is used generically to describe several selection schemes that choose individuals for birth according to their objective function values f. In these schemes, the probability of selection p of an individual from the ith class in the tth generation is calculated as

$$p_{i,t} = \frac{f_i}{\sum_{j=1}^{k} m_{j,t} f_j} \tag{29}$$

  where k classes exist and the total number of individuals sums to n. Several methods have been suggested for sampling this probability distribution, including Monte Carlo or **roulette wheel** selection,[19] **stochastic remainder** selection[20],[21] and **stochastic universal** selection[22].[23]

- **Ranking Selection:** In this scheme, proposed by Baker[24] the population is sorted from best to worst, and each individual is copied as many times as it can, according to a non-increasing assignment function, and then proportionate selection is performed according to that assignment.

- **Tournament Selection:** The population is shuffled and then is divided into groups of k elements from which the best individual (i.e., the fittest) will be chosen. This process has to be repeated k times because on each iteration only m parents are selected, where

$$m = \frac{population\ size}{k}$$

  For example, if we use binary tournament selection ($k = 2$), then we have to shuffle the population twice, since in each stage half of the parents required will be selected. The interesting property of this selection scheme is that we can guarantee multiple copies of the fittest individual among the parents of the next generation.

After being selected, **crossover** takes place. During this stage, the genetic material of a pair of individuals is exchanged in order to create the population of the next generation. The two main ways of performing crossover are called single-point and two-point crossover. When a **single-point** crossover scheme is used, a position of the chromosome is randomly selected as the crossover point as indicated in Figure 5. When a **two-point** crossover scheme is used, two positions of the chromosome are randomly selected as indicated in Figure 6.

Mutation is another important genetic operator that randomly changes a gene of a chromosome. If we use a binary representation, a mutation changes a 0 to 1 and viceversa. This operator allows the introduction of new chromosomic material to the population and, from the theoretical perspective, it assures that—given any population—the entire search space is connected.[17]

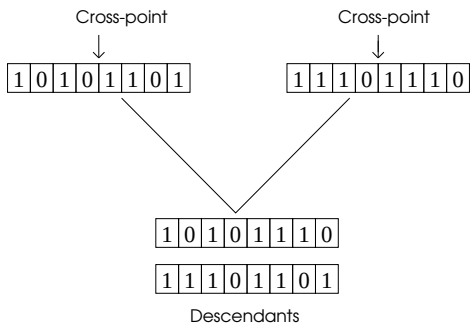

Figure 5: Use of a single-point crossover between two chromosomes. Notice that each pair of chromosomes produces two descendants for the next generation. The cross-point may be located at the string boundaries, in which case the crossover has no effect and the parents remain intact for the next generation.

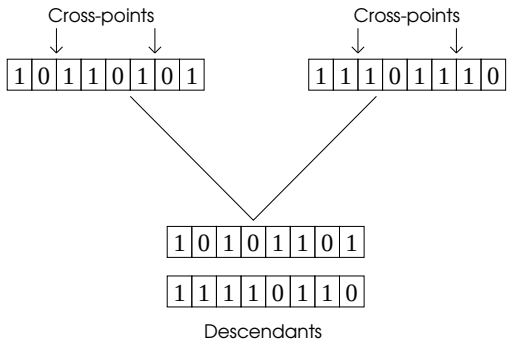

Figure 6: Use of a two-point crossover between two chromosomes. In this case the genes at the extremes are kept, and those in the middle part are exchanged. If one of the two cross-points happens to be at the string boundaries, a single-point crossover will be performed, and if both are at the string boundaries, the parents remain intact for the next generation.

1	0	0	0	1	1	0	1	0	1	0	0	0	0	1	0	1	0	1

Representation of the number 289.301 using binary encoding

2	8	9	3	0	1

Representation of the number 289.301 using floating point encoding

Figure 7: Representing the same number using binary and floating point encodings.

If we knew in advance the final solution, it would be trivial to determine how to stop a genetic algorithm. However, as this is not normally the case, we have to use one of the two following criteria to stop the GA: either give a fixed number of generations in advance, or verify when the population has stabilized (i.e., all or most of the individuals have the same fitness).

5 Using MOSES

To solve the multiobjective optimization problem presented in this paper, a system developed by the authors[25] called MOSES (Multiobjective Optimization of Systems in the Engineering Sciences), was used. Several Multiobjective Optimization approaches based on genetic algorithms are implemented in MOSES. The main body of the system is based on the Simple Genetic Algorithm (SGA) originally implemented by Goldberg[13] and then translated to C by R. E. Smith and modified by Jeff Erickson. However, this C implementation had to be modified to support both binary and floating point representations.

The traditional representation used by the genetic algorithms community is the binary scheme according to which a chromosome is a string the form $\langle b_1, b_2, \ldots, b_m \rangle$, where $b_1, b_2, \ldots, b_m$ are called *alleles* (either zeros or ones). Since the binary alphabet offers the maximum number of schemata per bit of information of any coding,[13] its use has became very popular among scientists. This coding also facilitates theoretical analysis of the technique and allows elegant genetic operators. However, since the "implicit parallelism" property of GAs does not depend on using bit strings[16] it is worthwhile to experiment with larger alphabets, and even with new genetic operators. In particular, for optimization problems in which the parameters to be adjusted are continuous, a floating point representation scheme seems a logical choice. According to this representation, a chromosome is a string of the form $\langle d_1, d_2, \ldots, d_m \rangle$, where $d_1, d_2, \ldots, d_m$ are digits (numbers between zero and nine). Consider the example shown in Figure 7, in which the same value is represented using binary and floating point encoding. Notice how in Figure 7 the binary value shown in the chromosome is not a typical binary representation of a floating point number in which some bits would represent the mantissa, another portion the exponent and the first bit the sign. The representation provided here is intended to be translated directly from binary to decimal, producing the value 289301 in this case, which will become the desired value (238.301) when we divide it by a certain amount (1000 in this example).

The term "floating" may seem misleading since the position of the implied decimal point is at a fixed position, and the term "fixed point representation" seems more appropriate. However, the reason that the term "floating point" is preferred is because in this representation each variable (representing a parameter to be optimized) may have the point at any position along the string. This means that even when the point is fixed for each gene, is not necessarily fixed along the chromosome. Therefore, some variables could have a precision of 3 decimal places,

while others are integers, and still they could all be represented with the same string. Nevertheless, the term *real-coded* GAs is also used in the Literature[26].[27] Floating point representation is faster and easier to implement, and provides a higher precision than its binary counterpart, particularly in large domains, where binary strings would be prohibitively long. One of the advantages of floating point representation is that it 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.[16] 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.

Goldberg[26] has presented a theory of convergence for real-coded or floating-point GAs, and also real numbers and other alphabets have been proposed,[27] particularly for numerical optimization, in a resemblance of the power of evolutionary strategies[28] in this domain. As Eshelman and Schaffer[29] point out, a lot of researchers in the GA community have agreed to use real-coded genetic algorithms for numerical optimization despite of the fact that there are theoretical arguments that seem to show that small alphabets should be more effective than large alphabets. Practitioners, on the other hand, have shown that real-coded genes work better in practice.[30] A few attempts have been made to develop a theoretical defense of this representation scheme, from which the recent work by Eshelman and Schaffer deserves special attention.[29] One of the main abilities of real-coded GAs is their capacity to exploit the gradualness of functions of continuous variables (where gradualness is taken to mean that small changes in the variables correspond to small changes in the function)[29].[27]

MOSES has an automatic encoding facility. The user can choose among three different types of variables: integer, discrete and real. The user has to provide the ranges of each variable (or the list of possible values if the type discrete is used), and MOSES will automatically compute the length of the chromosome. MOSES expects all the input from a file, and it needs another file to generate its output. The parameters of the GA (maximum number of generations, maximum number of runs, population size, crossover rate, mutation rate, maximum number of generations and random numbers seed) can be passed on the command line, but suitable defaults are included in the program. The system was designed in a modular fashion, so that the user only has to plug the particular decoding, report and fitness modules to start working. Everything else remains normally the same, except for the code used for selection, which can be changed according to the user needs. The fitness function used for this problem is the following:

$$t_1 = \left|\frac{f_1^0 - f_1}{f_1^0}\right| \quad t_2 = \left|\frac{f_2^0 - f_2}{f_2^0}\right| \quad t_3 = \left|\frac{f_3^0 - f_3}{f_3^0}\right| \quad t_4 = \left|\frac{f_4^0 - f_4}{f_4^0}\right|$$

$$t_5 = \left|\frac{f_1^0 - f_1}{f_1}\right| \quad t_6 = \left|\frac{f_2^0 - f_2}{f_2}\right| \quad t_7 = \left|\frac{f_3^0 - f_3}{f_3}\right| \quad t_8 = \left|\frac{f_4^0 - f_4}{f_4}\right|$$

$$\begin{array}{c} if\ (t_1 > t_5)\ z_1 = t_1\ else\ z_1 = t_5 \\ if\ (t_2 > t_6)\ z_2 = t_2\ else\ z_2 = t_6 \\ if\ (t_3 > t_7)\ z_3 = t_3\ else\ z_3 = t_7 \\ if\ (t_4 > t_8)\ z_4 = t_4\ else\ z_4 = t_8 \\ z = w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 \\ fitness = \frac{1}{z} \end{array} \tag{30}$$

The weights w_i were also chosen such that $w_1 + w_2 + w_3 + w_4 = 1$. Fifteen combinations of these four weights were used to generate the best overall result. For all tests, except for one method, binary tournament selection was used, together with double-point crossover, and a population size of 100 chromosomes. The methods included in MOSES are discussed in the following section.

6 Monte Carlo Methods

The two Monte Carlo methods used by Osyczka[31] to find the min-max optimum were also implemented in MOSES. These methods are called *exploratory* because a point is generated by means of a rule which disregards the results previously obtained. In particular, the Monte Carlo method picks up a certain number of points at random over the estimated range of all the variables of the problem. This is done formally by obtaining the randomly selected value for x_i from the following formula

$$x_i = x_i^a + \delta_i(x_i^b - x_i^a) \quad for\ i = 1, 2, \ldots, n \tag{31}$$

where x_i^a is the estimated or given lower limit for x_i, x_i^b is the estimated or given upper limit for x_i, and δ_i is a random number between zero and one. The same random number generator used by the genetic algorithm was employed to implement the FORTRAN function RANF of the original program.

If the values of variables for l^a points want to be generated, the random numbers δ_i for each point have to be generated first, and then equation (31) should be used to obtain the values of the variables x_i. After that, each generated point has to be tested for violation and discard it if it is not a feasible solution. If the point is in the feasible region, the objective function should be evaluated for that point. The best result is taken as the minimum, and a new set of random numbers is generated for each of l^a points.

The two Monte Carlo methods described by Osyczka[31] to find the min-max optimum are presented next.

6.1 Monte Carlo method 1

In this method, the space of variables is explored twice, first searching for the ideal vector $\bar{f}^0$ and then searching for the min-max optimum. The algorithm is the following:[31]

Do steps 1, 2, 3, 4, for $l = 1, 2, \ldots, l^a$
(1) Generate a random point $\bar{x}^{(l)}$.
(2) If the point $\bar{x}^{(l)}$ is not in the feasible region go to 1, otherwise go to 3.
(3) Evaluate $f_i(\bar{x}^{(l)})$ for $i = 1, 2, \ldots, k$.
(4) Replace f_i^0 by $f_i(\bar{x}^{(l)})$ for every i for which $f_i(\bar{x}^{(l)}) < f_i^0$.
Do steps 5, 6, 7, 8, for $l = 1, 2, \ldots, l^a$
(5) Generate a random point $\bar{x}^{(l)}$.
(6) If the point $x^{(l)}$ is not in the feasible region go to 5, otherwise go to 7.
(7) Evaluate $f_i(\bar{x}^{(l)})$ for $i = 1, 2, \ldots, k$.
(8) Check if the point $\bar{x}^{(l)}$ is the min-max optimum.

6.2 Monte Carlo method 2

Here, the space of variables is explored only once, and the Pareto set is generated while searching for the ideal vector $\bar{f}^0$. Then, this set analyzed to check which solution is the min-max optimum. The algorithm is the following:[31]

Do steps 1, 2, 3, 4, 5, for $l = 1, 2, \ldots, l^a$
(1) Generate a random point $\bar{x}^{(l)}$.
(2) If the point $\bar{x}^{(l)}$ is not in the feasible region go to 1, otherwise to 3.
(3) Evaluate $f_i(\bar{x}^{(l)})$ for $i = 1, 2, \ldots, k$.
(4) Replace f_i^0 by $f_i(\bar{x}^{(l)})$ for every i for which $f_i(\bar{x}^{(l)}) < f_i^0$.
(5) Check if the point $\bar{x}^{(l)}$ is Pareto optimal.
Do steps 6, 7 for $j = 1, 2, \ldots, j^a$
(6) Evaluate $f_i(\bar{x}_j^p)$ for $i = 1, 2, \ldots, k$.
(7) Check if the point $\bar{x}_j^p$ is the min-max optimum.

There are several trade-offs between these two methods. For example, the second method uses less CPU time than the first, because the space of variables is explored only once, but it also requires much more memory since the whole Pareto set has to be stored. Obviously, the designer normally wants to analyze the entire Pareto set in order to take a decision, but as I mentioned before, this set could be too large and the computational resources available could be insufficient for that sake. Osyczka recommends the reduction of this set by introducing constraints of the form $f_i(\bar{x}) \leq f_i^0$ for $i = 1, 2, \ldots, k$ where values of f_i^0 are chosen by the designer.

The second method should be preferred for problems with a large number of constraints and for discrete programming problems, because in those cases it is expected to have a small Pareto set. The main advantage of exploratory methods in general is their flexibility, since they can be applied both to linear and non-linear programming problems. However, they are normally recommended only for cases where a few decision variables are handled because otherwise, they could take too long to find a reasonable good solution.

7 Osyczka's Multicriterion Optimization System

This system was developed at the Technical University of Cracow, and its FORTRAN implementation is provided in Osyczka's book.[31] A C translation of that code was incorporated into MOSES, and its contents is explained next.

Osyczka's system contains several multiobjective optimization methods:

- **Min-max method** : The equation

$$\bigwedge_{i \in I} (z_i(\bar{x}) = max\ \{z_i^{'}(\bar{x}), z_i^{''}(\bar{x})\} \tag{32}$$

  is used to determine the elements of the vector $\bar{z}(\bar{x})$.

- **Global Criterion method** : The equation (24) is used as the global function.

- **Weighting min-max method** : This is a combination of the weighting method and the min-max approach that can find the Pareto set of solutions for both convex and non-convex problems. The equation

$$\bigwedge_{i \in I} (z_i(\bar{x}) = max\{w_i z_i^{'}(\bar{x}), w_i z_i^{''}(\bar{x})\}) \tag{33}$$

  is used to determine the elements of vector $\bar{z}(\bar{x})$.

- **Pure weighting method** : The equation

$$min \sum_{i=1}^{k} w_i f_i(\bar{x}) \tag{34}$$

  is used to determine a preferred solution. In this equation, $w_i \geq 0$ are the weighting coefficients representing the relative importance of the objectives.

- **Normalized weighting method** : $\bar{f}(\bar{x})$ is used in equation (34).

  Since all these methods require the ideal vector, the user is given the choice of providing it, or letting the system to find it automatically. For this purpose, the system includes two single criterion optimization techniques:

  1. The flexible tolerance (FT) method : Is a sequential method in which a point is established on the basis of the previously obtained results. Based on this information, the method will know where the minimum is likely to be so that the appropriate search direction may be established. Normally sequential methods, even when are more efficient and more highly developed than exploratory methods, tend to be designed to solve only continuous convex problems. However, this particular method can deal with non-linear models.[32]
  2. The direct and random search (DRS) method : It is a mixture of an exploratory and a sequential method. The direct search method[33] starts from the point chosen by the user and seeks a minimum. Then, a new starting point is generated at random and then the direct search method seeks a better solution. The procedure is repeated n times, and each time the direct search method starts from a new point where the value of n is given by the user. The best result from all searches is taken as the minimum.

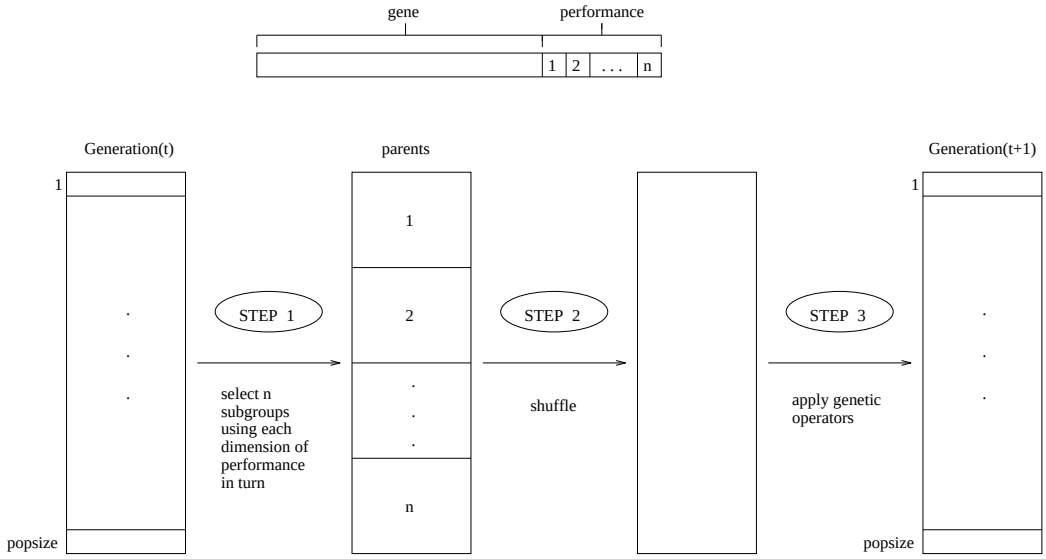

Figure 8: Schematic of VEGA selection.

8 GA-based Methods

Goldberg[13] indicates that the notion of genetic search in a multicriteria problem dates back to the late 60s, in which Rosenberg's[34] study contained a suggestion that would have led to multicriteria optimization if he had carried it out as presented. His suggestion was to use multiple **properties** (nearness to some specified chemical composition) in his simulation of the genetics and chemistry of a population of single-celled organisms. Since his actual implementation contained only one single property, the multiobjective approach could not be shown in his work, but it was a starting point for researchers interested in this topic.

Genetic algorithms require scalar fitness information to work, which means that when approaching multicriteria problems, it is necessary to perform a scalarization of the objective vectors. One problem is that it is not always possible to derive a global criterion based on the formulation of the problem. In the absence of information, objectives tend to be given equivalent importance, and when there is some understanding of the problem, it is possible to combine them according to the information available, probably assigning more importance to some objectives. Optimizing a combination of the objectives has the advantage of producing a single compromise solution, requiring no further interaction with the decision maker.[35] The problem is, that if the optimal solution cannot be accepted, either because the function used excluded aspects of the problem which were unknown prior to optimization or because an inappropriate setting of the coefficients of the combining function was chosen, additional runs may be required until a suitable solution is found.

8.1 VEGA

David Schaffer[36] extended Grefenstette's GENESIS program[37] to include multiple objective functions. Schaffer's approach was to use an extension of the Simple Genetic Algorithm (SGA) that he called the Vector Evaluated Genetic Algorithm (VEGA), and that differed of the first only in the way in which selection was performed. This operator was modified so that at each generation a number of sub-populations was generated by performing proportional selection according to each objective function in turn. Thus, for a problem with k objectives, k sub-populations of size N/k each would be generated, assuming a total population size of N. These sub-populations would be shuffled together to obtain a new population of size N, on which the GA would apply the crossover and mutation operators in the usual way. This process is illustrated in Figure 8 (taken from Schaffer[36]).

8.2 Lexicographic ordering

The basic idea of this technique is that the designer ranks the objectives in order of importance. The optimum solution is then found by minimizing the objective functions, starting with the most important one and proceeding according to the order of importance of the objectives.[38] Fourman[39] suggested a selection scheme based on lexicographic ordering. In a first version of his algorithm, objectives were assigned different priorities by the user and each pair of individuals were compared according to the objective with the highest priority. If this resulted in a tie, the objective with the second highest priority was used, and so on. A second version of this algorithm, reported to work surprisingly well,[35] consisted of randomly selecting the objective to be used in each comparison. As in VEGA, this corresponds to averaging fitness across fitness components, each component being weighted by the probability of each objective being chosen to decide each tournament.[35] However, the use of pairwise comparisons makes an important difference with respect to VEGA, since in this case scale information is ignored. Therefore, the population may be able to see as convex a concave trade-off surface, depending on its current distribution, and on the problem itself. This second version of the Lexicographic ordering algorithm was used in MOSES.

8.3 Multiple Objective Genetic Algorithm (MOGA)

Fonseca and Fleming[40] have proposed a scheme in which the rank of a certain individual corresponds to the number of chromosomes in the current population by which it is dominated. Consider, for example, an individual x_i at generation t, which is dominated by $p_i^{(t)}$ individuals in the current generation. Its current position in the individuals' rank can be given by:[40]

$$rank(x_i, t) = 1 + p_i^{(t)} \tag{35}$$

All non-dominated individuals are assigned rank 1, while dominated ones are penalized according to the population density of the corresponding region of the trade-off surface.

Fitness assignment is performed in the following way:[40]

1. Sort population according to rank.
2. Assign fitness to individuals by interpolating from the best (rank 1) to the worst (rank $n^* \leq N$) in the way proposed by Goldberg,[13] according to some function, usually linear, but not necessarily.
3. Average the fitnesses of individuals with the same rank, so that all of them will be sampled at the same rate. This procedure keeps the global population fitness constant while maintaining appropriate selective pressure, as defined by the function used.

8.4 Non-dominated Sorting Genetic Algorithm

The Non-dominated Sorting Genetic Algorithm (NSGA) was proposed by Srinivas and Deb,[41] and is based on several layers of classifications of the individuals. Before the selection is performed, the population is ranked on the basis of nondomination: all nondominated individuals are classified into one category (with a dummy fitness value, which is proportional to the population size, to provide an equal reproductive potential for these individuals). To maintain the diversity of the population, these classified individuals are shared with their dummy fitness values. Then this group of classified individuals is ignored and another layer of nondominated individuals is considered. The process continues until all individuals in the population are classified. A stochastic remainder proportionate selection was used for this approach. Since individuals in the first front have the maximum fitness value, they always get more copies than the rest of the population. This allows to search for nondominated regions, and results in quick convergence of the population toward such regions. Sharing, by its part, helps to distribute it over this region. The efficiency of NSGA lies in the way multiple objectives are reduced to a dummy fitness function using a nondominated sorting procedure. With this approach, any number of objectives can be solved,[42] and both maximimization and minimization problems can be handled. Figure 9 (taken from Srinivas and Deb[42]) shows the general flow chart of this approach. This method is the only one implemented in MOSES that does not use tournament selection, but uses the stochastic remainder method instead.

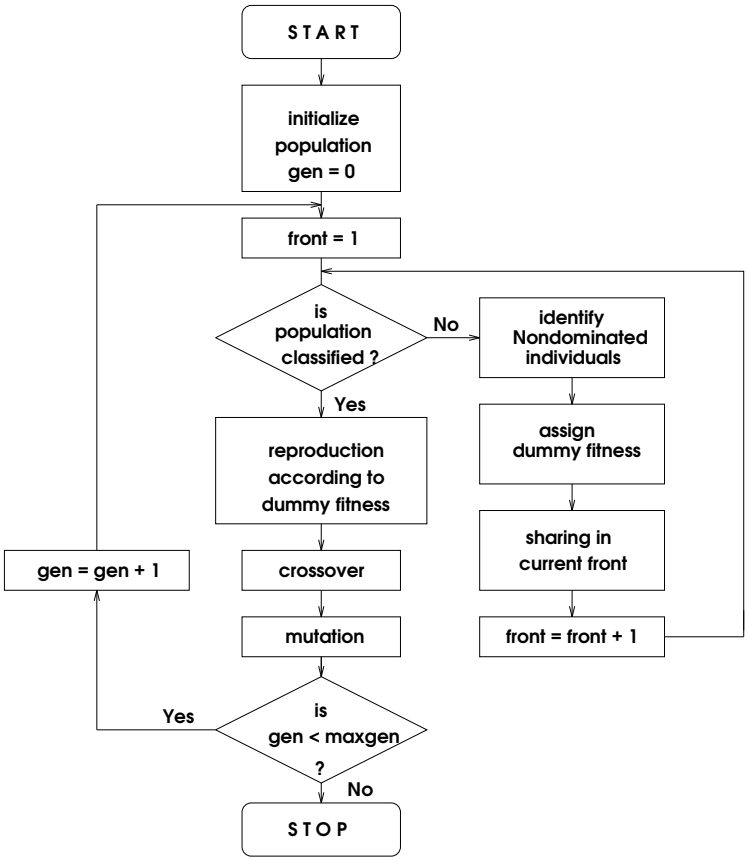

Figure 9: Flowchart of the Nondominated Sorting Genetic Algorithm (NSGA).

8.5 Niched Pareto GA

Horn and Nafpliotis[43] proposed a tournament selection scheme based on Pareto dominance. Instead of limiting the comparison to two individuals, a number of other individuals in the population was used to help determine dominance. When both competitors were either dominated or non-dominated (i.e., there was a tie), the result of the tournament was decided through fitness sharing.[44] Population sizes considerably larger than usual were used so that the noise of the selection method could be tolerated by the emerging niches in the population.[35]

The pseudocode for Pareto domination tournaments assuming that all of the objectives are to be maximized is presented below.[43] S is an array of the N individuals in the current population, *random_pop_index* is an array holding the N indices of S, in a random order, and t_{dom} is the size of the comparison set.

```
function selection /* Returns an individual from the current population S */
begin
    shuffle(random_pop_index); /* Re-randomize random index array */
    candidate_1 = random_pop_index[1];
    candidate_2 = random_pop_index[2];
    candidate_1_dominated = false;
    candidate_2_dominated = false;
    for comparison_set_index = 3 to t_dom + 3 do /* Select t_dom individuals randomly from S */
        begin
            comparison_individual = random_pop_index[comparison_set_index];
            if S[comparison_individual] dominates S[candidate_1]
                then candidate_1_dominated = true;
            if S[comparison_individual] dominates S[candidate_2]
                then candidate_2_dominated = true;
        end /* end for loop */
    if ( candidate_1_dominated AND ¬ candidate_2_dominated )
        then return candidate_2;
    else if ( ¬ candidate_1_dominated AND candidate_2_dominated )
        then return candidate_1;
    else
        do sharing;
end
```

8.6 Hajela's Approach

Hajela and Lin[45] proposed the use of a utility function of the form:

$$\bar{U} = \sum_{i=1}^{l} W_i \frac{F_i}{F_i^*} \tag{36}$$

where F_i^* are the scaling parameters for the objective criterion, l is the number of objective functions, and W_i are the weighting factors for each objective function F_i. In the current implementation, a min-max approach was used to determine the utility function, so that the scaling factor was the ideal vector.

Hajela's approach also uses a sharing function of the form:

$$\phi(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{sh}}\right)^{\alpha}, & d_{ij} < \sigma_{sh} \\ 0, & \text{otherwise} \end{cases} \tag{37}$$

where $\alpha = 1$ for this work, d_{ij} is a metric indicative of the distance between designs i and j, and σ_{sh} is the sharing parameter, which is typically chosen between 0.01 and 0.1. The fitness of a design i is then modified as:

$$f_{s_i} = \frac{f_i}{\sum_{j=1}^{M} \phi(d_{ij})} \tag{38}$$

where M is the number of designs located in vicinity of the i-th design.

Hajela incorporates weight combinations into the chromosomic string, so MOSES's implementation was extended to accomodate the additional genes required, according to the number of weight combinations provided by the user. Under Hajela's representation, a single number represents not the weight itself, but a combination of weights. For example, the number 4 (under floating point representation) could represent the vector $X_w = (0.4, 0.6)$ for a problem with two objective functions. Then, sharing is done on the weights.

Finally, a mating restriction mechanism was imposed, to avoid members within a radius σ_{mat} to cross. The value of $\sigma_{mat} = 0.15$ used by Hajela was adopted in MOSES's implementation.

8.7 An Approach Based on a Weighted Min-Max Strategy

This is really a variant of Hajela's idea, in which a few changes were introduced by the authors:

1. The initial population is generated in such a way that all their individuals constitute feasible solutions. This can be ensured by checking that none of the constraints is violated by the solution vector encoded by the corresponding chromosome.

2. The user should provide a vector of weights, which are used to spawn as many processes as weight combinations are provided (normally this number will be reasonably small). Each process is really a separate genetic algorithm in which the given weight combination is used in conjunction with a min-max approach to generate a single solution. It should be noticed that in this case the weights do not have to be encoded in the chromosome as in Hajela's approach.

3. After the n processes are terminated (n=number of weight combinations provided by the user), a final file is generated containing the Pareto set, which is formed by picking up the best solution from each of the processes spawned in the previous step.

4. Since this approach requires to know the ideal vector, the user is given the choice to provide such values directly (in case he/she knows them) or to use another genetic algorithm to generate it. This additional program works in a similar manner, spawning k processes (k=number of objective functions), where each process corresponds to a genetic algorithm responsible for a single objective function. When all the processes terminate, there will be a file containing the ideal vector, which turns out to be simply the best values produced by each one of the spawned processes.

5. The crossover and mutation operators were modified to ensure that they produced only feasible solutions. Whenever a child encodes an unfeasible solution, it is replaced by one of its parents.

6. Notice that the Pareto solutions produced by this method are guaranteed to be feasible, as opposed to the other GA-based methods in which there could be convergence towards a non-feasible solution.

9 The GA optimizer for single-objective problems

Using the GA itself as an optimizer for single-objective problems is a controversial topic, mainly because the difficulties found to adjust its parameters (i.e., population size, maximum number of generations, mutation and crossover rate).[46] Since once of the goals of this work was to be able to produce a reliable design optimization system, this is a natural problem to face. In practice, GA parameters are empirically adjusted in a trial and error process that could take quite a long time in some cases.

During several months, a very simple methodology, explained below, has been tried with different engineering design optimization problems, and the results obtained so far[25] led to think that it was a reasonable choice to use in MOSES. The method is the following:

- Choose a certain value for the random numbers seed and make it a constant.

- Make constants also the population size and the maximum number of generations (100 chromosomes and 50 generations, respectively are normally used).

Method	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4
Monte Carlo 1	0.19247	0.19511	34.9006	2.4059	**103.11**	39.30	708.72	236.49
Monte Carlo 1	0.02417	0.09112	18.9996	14.5462	214.30	**29.85**	795.79	434.79
Monte Carlo 1	0.04311	0.10200	0.00755	0.70932	135.1197	41.1169	**385.88**	205.76
Monte Carlo 1	0.01713	0.12098	29.5132	0.02916	127.45	41.90	658.27	**194.94**
Min-Max (OS)	0.19247	0.19511	34.9006	2.40593	**103.11**	39.30	708.72	236.49
Min-Max (OS)	0.02417	0.09112	18.9996	14.5462	214.30	**29.85**	795.79	434.79
Min-Max (OS)	0.04311	0.10200	0.07550	0.70932	135.12	41.12	**385.88**	205.76
Min-Max (OS)	0.01713	0.12098	29.5132	0.02916	127.45	41.90	658.27	**194.94**
GA (Binary)	0.1565	0.2000	35.0000	0.4095	**92.82**	41.25	679.78	201.59
GA (Binary)	0.1621	0.0963	22.4752	15.00	194.43	**29.60**	805.76	444.92
GA (Binary)	0.2000	0.1972	0.00340	0.0000	132.20	41.94	**373.85**	194.52
GA (Binary)	0.1419	0.0965	1.8294	0.0066	130.06	41.93	389.06	**194.61**
GA (FP)	0.1557	0.2000	35.00	0.9869	**91.99**	40.29	684.44	211.95
GA (FP)	0.2000	0.0930	35.00	15.00	168.03	**29.59**	890.92	443.34
GA (FP)	0.2000	0.2000	0.0000	0.0004	132.21	41.94	**373.83**	194.52
GA (FP)	0.2000	0.0096	35.00	0.0010	105.22	41.94	692.73	**194.52**
Literature	0.199	0.199	34.98	5.77	**112.75**	39.06	750.60	303.09
Literature	0.175	0.114	10.24	14.86	216.76	**30.21**	713.05	452.31
Literature	0.198	0.140	0.001	0.002	133.11	41.94	**374.82**	195.23
Literature	0.191	0.198	14.30	0.001	111.99	41.94	485.66	**195.21**

Table 2: Comparison of results computing the ideal vector for the design of a robot arm. For each method the best results for optimum f_1, f_2, f_3 and f_4 are shown in **boldface**. OS stands for Osyczka's Multiobjective Optimization System.

- Loop the mutation and crossover rates from 0.1 to 0.9 at increments of 0.1 (this is actually a nested loop). This implies that 81 runs are necessary. In each step of the loop, the population is not reinitialized.
- For each run, update 2 files. One contains only the final costs, and the other has a summary that includes, besides the cost, the corresponding values of the design parameters and the mutation and crossover rates used.
- When the whole process ends up, the file with the costs is sorted in ascending order, and the smallest value is searched in the other file, returning the corresponding design parameters as the final answer.

10 Comparison of Results

First, the ideal vector was generated using each of the single objective optimization techniques included in MOSES. The results are presented in Table 2. As can be seen from these results, the GA with floating point representation was able to find the complete ideal vector, obtaining even better results than those previously reported in the literature.[1] It is interesting to notice that the results reported by Koski and Osyczka[1] using CAMOS for computing the ideal vector are not any better than those obtained with Osyczka's Multiobjective Optimization System,[31] which is an older program. Probably the reason for that is that the software used to compute the ideal vector of this problem could had employed less digits of precision than MOSES's implementation of Osyczka's Multiobjective Optimization System, producing a discrepancy in their results. CAMOS is not part of MOSES, so the results shown in Table 2 were taken directly from the literature.[1] Nevertheless, CAMOS achieves a much better overall solution than any of the methods included in Osyczka's Multiobjective Optimization System (see Table 3).

Method	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	$L_p(f)$
Ideal Vector					91.99	29.59	373.83	194.52	0.000000
Monte Carlo 1	0.18738	0.18074	13.47721	1.5555	117.68	39.74	505.73	221.24	1.112581
Monte Carlo 2	0.12276	0.17042	9.26852	0.33446	123.43	41.41	458.17	200.12	0.995563
Min-max (OS)	0.12438	0.09609	29.9961	6.9961	135.62	35.06	740.01	306.76	2.215680
GCM (OS)	0.12438	0.09609	29.9961	6.9961	135.62	35.06	740.01	306.76	2.215680
WMM (OS)	0.12438	0.09609	29.9961	6.9961	135.62	35.06	740.01	306.76	2.215680
PMM (OS)	0.12438	0.09609	29.9961	6.9961	135.62	35.06	740.01	306.76	2.215680
NMM (OS)	0.12438	0.09609	29.9961	6.9961	135.62	35.06	740.01	306.76	2.215680
GALC (B)	0.20000	0.20000	20.2738	0.0132	102.18	41.94	529.01	194.52	0.943299
GALC (FP)	0.20000	0.06450	35.0000	15.000	161.89	30.85	878.20	430.70	3.365909
Lexicographic (B)	0.04690	0.01970	10.5668	0.4671	129.19	41.29	414.91	202.44	0.950512
Lexicographic (FP)	0.20000	0.02720	35.0000	0.8189	105.00	41.74	694.10	197.74	1.425351
VEGA (B)	0.18010	0.08990	15.3888	0.4392	129.71	41.90	393.13	194.90	0.879695
VEGA (FP)	0.20000	0.01320	35.0000	0.0188	105.20	41.94	692.83	194.79	1.415545
NSGA (B)	0.13680	0.06090	20.9519	0.4369	115.61	41.69	465.15	197.78	0.926816
NSGA (FP)	0.14100	0.20000	35.0000	0.2221	93.16	41.48	676.66	199.22	1.248742
MOGA (B)	0.17370	0.00630	35.0000	9.1130	144.92	34.49	637.22	316.49	2.072441
MOGA (FP)	0.20000	0.08090	35.0000	15.000	165.01	29.73	885.55	438.00	3.418972
NPGA (B)	0.08520	0.04760	35.0000	0.6984	109.84	41.51	695.58	204.99	1.511325
NPGA (FP)	0.20000	0.20000	0.0009	15.000	227.94	29.75	631.28	449.13	3.480962
Hajela (B)	0.19990	0.11240	0.0391	0.1537	132.75	41.75	376.58	196.89	0.873619
Hajela (FP)	0.20000	0.20000	35.000	10.0624	166.52	47.46	841.71	394.29	3.692761
GAminmax1 (B)	0.16430	0.20000	0.0059	0.0006	132.20	41.94	373.87	194.52	0.172920
GAminmax1 (FP)	0.20000	0.20000	0.0300	0.0440	133.16	41.87	375.72	195.91	**0.091128**
Literature	0.10300	0.11400	0.1380	2.0800	141.63	39.46	408.89	228.29	0.194282

Table 3: Comparison of the best overall solution found by each one of the methods included in MOSES. GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective).

Method	w_1	w_2	w_3	w_4	f_1	f_2	f_3	f_4	x_1	x_2	x_3	x_4	$L_p(f)$
Koski	0.25	0.25	0.25	0.25	138.88	38.93	510.18	268.92	0.186	0.198	7.95	4.06	0.3809
GA	0.25	0.25	0.25	0.25	133.16	41.87	375.73	195.92	0.200	0.200	0.029	0.045	0.2170
Koski	0.3	0.3	0.2	0.2	139.91	37.98	612.36	298.39	0.171	0.184	16.9	5.66	0.4737
GA	0.3	0.3	0.2	0.2	102.45	41.87	532.12	195.92	0.200	0.200	20.46	0.045	0.2431
Koski	0.35	0.35	0.15	0.15	152.99	37.74	667.45	336.62	0.194	0.182	19.6	7.59	0.5540
GA	0.35	0.35	0.15	0.15	96.99	40.70	581.09	209.56	0.200	0.200	25.055	0.853	0.2438
Koski	0.4	0.4	0.1	0.1	152.76	38.85	800.85	344.61	0.130	0.193	32.9	7.84	0.5793
GA	0.4	0.4	0.1	0.1	94.71	40.20	615.27	215.86	0.200	0.1778	27.689	1.237	0.2298
Koski	0.2	0.2	0.3	0.3	136.76	38.91	505.85	264.17	0.190	0.197	8.05	3.82	0.3711
GA	0.2	0.2	0.3	0.3	133.15	41.87	375.76	195.92	0.200	0.200	0.033	0.045	0.1742
Koski	0.15	0.15	0.35	0.35	139.62	38.63	457.88	245.80	0.200	0.200	0.039	0.044	0.2917
GA	0.15	0.15	0.35	0.35	133.14	41.87	375.79	195.91	0.200	0.200	0.039	0.044	0.1315
Koski	0.1	0.1	0.4	0.4	141.63	39.46	408.89	228.29	0.103	0.114	0.138	2.08	0.1943
GA	0.1	0.1	0.4	0.4	133.16	41.87	375.72	195.91	0.200	0.200	0.03	0.044	0.0911
Koski	0.5	0.1	0.2	0.2	99.44	41.46	592.53	202.09	0.172	0.093	26.5	0.45	0.2036
GA	0.5	0.1	0.2	0.2	98.91	41.88	553.41	195.84	0.200	0.200	23.244	0.04	0.1749
Koski	0.1	0.5	0.2	0.2	153.03	35.75	645.41	335.46	0.198	0.157	17.0	7.84	0.4584
GA	0.1	0.5	0.2	0.2	133.25	41.84	375.83	196.25	0.200	0.200	0.0	0.065	0.2533
Koski	0.4	0.2	0.2	0.2	121.99	38.42	606.99	258.65	0.148	0.182	20.6	3.6	0.3788
GA	0.4	0.2	0.2	0.2	98.91	41.87	553.44	195.91	0.200	0.200	23.243	0.044	0.2090
Koski	0.2	0.4	0.2	0.2	162.68	39.11	583.60	319.94	0.152	0.198	10.3	6.6	0.5215
GA	0.2	0.4	0.2	0.2	133.16	41.87	375.74	195.92	0.200	0.200	0.031	0.045	0.2566

Table 4: Pareto-optimal solutions for the robot arm whose mechanical model is shown in Figure 1.

The best trade-off results obtained by each one of the methods included in MOSES for multiobjective optimization were compared against each other, producing the results shown in Table 3. To evaluate these results, the maximum deviation from the optimum was used as a parameter. This maximum deviation is defined by

$$L_p(f) = \sum_{i=1}^{4} w_i \left| \frac{f_i^0 - f_i(x)}{\rho_i} \right| \tag{39}$$

where $\rho_i = f_i^0$,or $f_i(x)$, depending on which gives the maximum value for $L_p(f)$.

It should be mentioned that this expression will favor mathematical programming techniques and approaches such as Hajela's and the new algorithm presented before, in which the emphasis is on obtaining the best overall result. Further studies[25] have shown that techniques such as MOGA[35] are very successful at keeping the population of a GA from converging to a single solution, and can also obtain some reasonable overall results under certain conditions not met by this problem. However, the goal of this work was to show that it was possible to develop a GA-based technique that could compete with any mathematical programming technique in finding the best overall solution to a complex multiobjective optimization problem, while at the same time avoiding total convergence of the population.

Table 3 shows the comparison of results using all the techniques implemented in MOSES and CAMOS,[1] including the new method based on the min-max algorithm. As can be seen, this new method found the best trade-off solution using both representations, surpassing even the mathematical programming techniques employed. Floating point representation provided the best result of all using this new method, with a significantly low total deviation, showing the suitability of this representation for numerical optimization problems.

Also, the Pareto front was generated, using the eleven weights proposed by Koski et al.[1] using the new method based on the min-max optimum, and the comparison of results is presented in Table 4. As can be seen, the new technique consistently finds better results than Koski's algorithm, proving its efficiency in this domain. Only floating point representation was used in this case with the new algorithm, since it has consistently provided with better results in all the experiments performed so far.

The two main drawbacks of this new technique when compared to similar GA-based approaches are that the user has to decide what are the weights to be employed and that the ideal vector has to be known. With respect to the first drawback, it can be said that small sets (of a maximum of about 20 vectors of values) have proved to be sufficient in practice, even when dealing with problems with higher search spaces.[25] With respect to the second problem, it should be said that the ideal vector does not have to be known beforehand, since this algorithm works with a utility function. This means that any set of values that are considered suitable can be employed, even if they underestimate or overestimate the optima. Nevertheless, a module to compute the ideal vector using either mathematical programming or GA-based single objective optimization techniques has also be included in MOSES.

11 Future Work

Because of the intensive CPU time-consuming nature of this problem, it would be desirable to explore the use of other techniques that can reduce the number of function evaluations, such as the approximation of functions by low order polynomials over some small region.[47] In this case a computationally expensive function is evaluated at a sufficient number of points to construct a low order polynomial approximation. Then, an iterative optimization algorithm is used for finding the minimum of the approximate function. At the point obtained the optimization model is replaced by a new approximate model, and the process continues until the improvement in the objective function can not be distinguished.

Another interesting path of research is to explore other possible alternatives to use genetic algorithms to solve multiobjective optimization problems. In that respect, there has been some experimenting with another technique also based on the min-max optimum but that does not require to have the ideal vector or any other set of target values to compute the Pareto set.[25] This approach uses sharing to keep the GA from converging to a single solution, but has still some difficulties to generate good trade-offs in certain domains, and more work has to be done in that respect.

12 Conclusions

A GA-based min-max approach has been proposed for a complex multiobjective optimization problem: a robot arm balancing. Also, MOSES, a multiobjective design optimization system developed by the authors was introduced as a powerful tool to apply different mathematical programming and GA-based techniques to numerical optimization problems. Its modularity makes it easy to expand it in the future to include new approaches as required by the user's needs.

The problem analyzed in this paper has four objective functions to be minimized, and is highly non-convex. Furthermore, the complex calculations involved consume a lot of CPU time, and make necessary the development of heuristic techniques that need the least possible number of function evaluations. The great variation of the results obtained show that this problem would be very difficult to solve with pure random search, or with brute-force techniques. Also, to find a reasonable heuristics seems a difficult task given the factors previously mentioned, and the possible presence of local minima. The GA has showed to be very consistent in this application, finding better compromise solutions for all the instances of the problem under consideration.

Finally, some other GA-based approaches seem suitable for this application, especially those in which a Pareto-based selection is applied. However, time remains to be an issue to be considered in further applications of the GA to this problem, and it would be desirable to explore techniques for reducing the number of function evaluations. Nevertheless, the use of such a powerful heuristic should bring benefits to the robotics industry, and this work should be seen as a small module of a larger system whose goal is to optimize the entire design process of a robot arm.

References

1. J. Koski and A. Osyczka. Optimal counterweight balancing of robot arms using multicriteria approach. In Hans Eschenauer, Juhani Koski, and Andrzej Osyczka, editors, *Multicriteria Design Optimization. Procedures and Applications*, chapter 5, pages 151–167. Springer-Verlag, Berlin, Germany, 1990.

2. Charles P. Neuman and John J. Murray. The complete dynamic model and customized algorithms for the puma robot. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(4):635–644, jul 1987.

3. B. Armstrong, O. Khatib, and J. Burdick. The explicit dynamic model and inertial parameters of the PUMA 560 arm. In *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, pages 510–518, San Francisco, California, apr 1986.

4. Andrzej Osyczka. Multicriteria optimization for engineering design. In John S. Gero, editor, *Design Optimization*, pages 193–227. Academic Press, 1985.

5. L. M. Boychuck and V. O. Ovchinnikov. Principal methods for solution of multi-criteria optimization problems (survey). *Soviet Automatic Control*, 6(3):1–4, 1973.

6. M. E. Salukvadze. On the existence of solution in problems of optimization under vector valued criteria. *Journal of Optimization Theory and Applications*, 12(2):203–217, 1974.

7. P. Hajela. Genetic search–an approach to the nonconvex optimization problem. *AIAA Journal*, 28(7):1205–1210, 1990.

8. H. A. Eschenauer, A. Osyczka, and E. Schäfer. Interactive multicriteria optimization in design process. In Hans Eschenauer, Juhani Koski, and Andrzej Osyczka, editors, *Multicriteria Design Optimization. Procedures and Applications*, chapter 3, pages 71–114. Springer-Verlag, Berlin, 1990.

9. J. A. Nelder and R. A. Mead. Simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

10. Charles Darwin. *The Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life*. The Book League of America, 1929. Originally published in 1859.

11. John H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Harbor : University of Michigan Press, 1975.

12. John H. Holland. *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, Massachusetts, 1992.

13. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, Mass. : Addison-Wesley Publishing Co., 1989.

14. John R. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

15. Joerg Heitkoetter and David Beasley. The hitch-hiker's guide to evolutionary computation (faq in comp.ai.genetic). USENET, sep 1995. (Version 3.3).

16. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, second edition, 1992.

17. Bill P. Buckles and Frederick E. Petry. *Genetic Algorithms.* Technology Series. IEEE Computer Society Press, 1992.

18. David E. Goldberg and Kalyanmoy Deb. A comparison of selection schemes used in genetic algorithms. In G.J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, California, 1991.

19. A. K. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975.

20. L. B. Booker. Intelligent behavior as an adaptation to the task environment. Technical Report 243, University of Michigan at Ann Arbor, Ann Arbor, Michigan, 1982.

21. A. Brindle. *Genetic Algorithms for Function Optimization.* PhD thesis, Department of Computer Science of the University of Alberta, Alberta, Canada, 1981.

22. J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In John Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, Hillsdale, New Jersey, 1987. Lawrence Erlbaum Associates.

23. J. J. Grefenstette and J. E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 20–27, San Mateo, California, jun 1989. George Mason University, Morgan Kaufmann Publishers.

24. J. E. Baker. Adaptive selection methods for genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 100–111, Hillsdale, New Jersey, 1985. Lawrence Erlbaum.

25. Carlos Artemio Coello Coello. *Multiobjective Engineering Design Optimization using Genetic Algorithms.* PhD thesis, Department of Computer Science, Tulane University, May 1996. (to be published).

26. David E. Goldberg. Real-coded genetic algorithms, virtual alphabets and blocking. Technical Report 90001, University of Illinois at Urbana-Champaign, Urbana, Illinois, sep 1990.

27. Alden H. Wright. Genetic algorithms for real parameter optimization. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann Publishers, San Mateo, California, 1991.

28. H. P. Schwefel. *Numerical Optimization of Computer Models.* John Wiley and sons, Great Britain, 1981.

29. Larry J. Eshelman and J. Davis Schaffer. Real-coded genetic algorithms and interval-schemata. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann Publishers, San Mateo, California, 1993.

30. Lawrence Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, New York, 1991.

31. Andrzej Osyczka. *Multicriterion Optimization in Engineering with FORTRAN programs.* Ellis Horwood Limited, 1984.

32. David M. Himmelblau. *Applied Nonlinear Programming.* McGraw-Hill Book Company, New York, 1972.

33. R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the ACM*, 8:221–230, 1961.

34. R. S. Rosenberg. *Simulation of genetic populations with biochemical properties.* PhD thesis, University of Michigan, Ann Harbor, Michigan, 1967.

35. Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. Technical report, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, U. K., 1994.

36. J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.

37. J. J. Grefenstette. GENESIS: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165, 1984.

38. S. S. Rao. Multiobjective optimization in structural design with uncertain parameters and stochastic processes. *AIAA Journal*, 22(11):1670–1678, nov 1984.

39. M. P. Fourman. Compaction of symbolic layout using genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 141–153. Lawrence Erlbaum, 1985.

40. Carlos M. Fonseca and Peter J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

41. N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. Technical report, Department of Mechanical Engineering, Indian Institute of Technology, Kanput, India, 1993.

42. N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, fall 1994.

43. J. Horn and N. Nafpliotis. Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

44. David E. Goldberg and J. Richardson. Genetic algorithm with sharing for multimodal function optimization. In J. J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum, 1987.

45. P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4:99–107, 1992.

46. J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

47. A. Osyczka and J. Zajac. Multicriteria optimization of computationally expensive functions and its applications to robot spring balancing mechanism design. In Hans Eschenauer, Juhani Koski, and Andrzej Osyczka, editors, *Multicriteria Design Optimization. Procedures and Applications*, chapter 5.2, pages 168–183. Springer-Verlag, Berlin, 1990.