International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM)
First Special Issue on "Computer Graphics and Geometric Modeling" (2008)
Selected Papers from CGGM'2007 Workshop

Bio-Inspired Metaheuristic Methods for Fitting Points in CAGD

Angel Cobo, Akemi Gálvez, Jaime Puig-Pey, Andrés Iglesias
Dept. of Applied Mathematics and Computational Sciences
University of Cantabria
E-39005, Santander, Spain
{acobo,galveza,puigpeyj,iglesias}@unican.es

Jesús Espinola
Faculty of Sciences
National University Santiago Antúnez de Mayolo
Huaraz, Peru
espinolj@gmail.com

Abstract- This paper deals with a classical optimization problem, fitting 3D data points by means of curve and surface models used in Computer-Aided Geometric Design (CAGD). Our approach is based on the idea of combining traditional techniques, namely best approximation by least-squares, with Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), both based on bio-inspired procedures emerging from the artificial intelligence world. In this work, we focus on fitting points through free-form parametric curves and surfaces. This issue plays an important role in real problems such as construction of car bodies, ship hulls, airplane fuselage, and other free-form objects. A typical example comes from reverse engineering where free-form curves and surfaces are extracted from clouds of data points. The performance of the proposed methods is analyzed by using some examples of Bézier curves and surfaces.

1 Introduction

Fitting points is a relevant problem in several areas; it is a major issue in regression analysis in Statistics [6], usually fitting an explicit polynomial function in one or several variables, plus a random noise term. Best approximation of points or functions is one of the main topics of Numerical Analysis, and is the source of most of the methods for performing practical fitting processes by computer [4, 5]. Most of the common models in Computer-Aided Geometric Design (CAGD) make use of polynomials [15]. In CAGD, data usually come from real measurements of an existing geometric entity. For instance, a typical geometric problem in Reverse Engineering is the process of converting dense clouds of data points captured from the surface of an object into a boundary representation CAGD model [9, 16, 17, 21, 23]. Models are often parametric representations of curves and surfaces, and best approximation methods make commonly use of least-squares techniques.

Genetic algorithms are increasingly used for geometric problems involving optimization processes with a great number of unknown variables [20]. The CAD (Computer-Aided Design) journal devoted a special issue in 2003 to genetic algorithms [22] and included one paper addressing data fitting with spline polynomials in explicit form [26]. A detailed description on genetic algorithms will be given in Section 3 while Sections 4 to 6 describe its application to data fitting.

Particle Swarm Optimization (PSO), another popular metaheuristic technique with biological inspiration, is also used in CAM (Computer-Aided Manufacturing) for dealing with optimization of milling processes [10]. The original PSO algorithm was first reported in 1995 by James Kennedy and Russell C. Eberhart in [18]. In [8] some developments are presented. These authors integrate their contributions in [19]. In general, the PSO techniques have evolved greatly, from the more intuitive initial idea to more formal convergence analysis within more general optimization frameworks [24]. In this paper, PSO is described in Section 7 while some examples for Bézier curves and surfaces are discussed in Section 8.

In this work we address the problem of fitting data points through parametric models, which are more relevant in CAGD than the explicit ones. In this case, an additional and important problem is to obtain a suitable parameterization of the data points. As remarked in [1, 2] the selection of an appropriate parameterization is essential for topology reconstruction and surface fitness. These parameter values are also unknowns which introduce nonlinearity in the fitting problem [25]. In this paper, we focus on fitting points with Bézier curves and surfaces, combining the search of parameter values by using GA or PSO techniques, with the least-squares minimization of quadratic errors. Some simple yet illustrative examples to show the good performance of the proposed methods are also briefly described.

2 The problem

As a simple explanatory example in simple linear regression, the process of fitting a set of given data points $\{(x_i, y_i)\}_{i=1,\ldots,n_p}$, to a straight line, $y = a + bx$, by applying the least-squares technique can readily be done: the coefficients a and b are the solutions of the linear system obtained by minimizing the error function, represented by the sum of squared error distances in the y-direction.

Let us suppose that we are dealing with a 3D parametric curve $\mathbf{C}(t) = \sum_{j=1}^{n_b} \mathbf{P}_j B_j(t)$, where $\mathbf{P}_j$ are vector coefficients, $B_j(t)$ are the *basis functions* (or *blending functions*) of the parametric curve $\mathbf{C}(t)$ and t is the parameter, usually

defined on a finite interval $[\alpha, \beta]$. Note that in this paper vectors are denoted in bold. Now we can compute, for each of the cartesian components (x, y, z), the minimization of the sum of squared errors referred to the data points, given by:

$$E_\gamma = \sum_{i=1}^{n_p} \left(\gamma_i - \sum_{j=0}^{n_b} c_{\gamma,j} B_j(t_i) \right)^2 \; ; \; \gamma = x, y, z \qquad (1)$$

but we need a parameter value t_i to be associated with each data point (x_i, y_i, z_i), $i = 1, \ldots, n_p$. Coefficients $c_{\gamma,j}$, $j = 0, \ldots, n_b$, have to be determined from the information given by the data points (x_i, y_i, z_i), $i = 1, \ldots, n_p$. We can see that performing the component-wise minimization of these errors is equivalent to minimizing the sum, over the set of data, of the Euclidean distances between data points and corresponding points given by the model in 3D space. Note that in addition to the coefficients of the basis functions, $c_{\gamma,j}$, parameter values t_i corresponding to the data points also appear as unknowns in (1). Due to the fact that the blending functions $B_j(t)$ are usually nonlinear in t, the least-squares minimization of the errors is a strongly nonlinear problem, with a high number of unknowns for large sets of data points, a case that happens very often in practice. On the other hand, if values are assigned to the t_i, the problem becomes a classical linear least-squares minimization for each component, with the coefficients $c_{\gamma,j}$ as unknowns.

Our strategy for solving the problem in the general case is a mixed one, combining genetic algorithms (GA) in one case, or particle swarm optimization (PSO) in the other, to determine suitable parameter values t_i for the data points, and calculating the best least-squares fitting coefficients $c_{\gamma,j}$ and the corresponding error for the set of parameter values provided by the GA/PSO method. The process is performed iteratively while the evolution of the parameters does not stabilize the minimization of the error, using GA or PSO techniques. As basis functions, we use polynomials $B_j(t)$ of low degree in order to prevent possible numerical instabilities and over-fitting of the data.

3 Genetic Algorithms

Genetic Algorithms (GA) [13] are search procedures based on principles of evolution and natural selection. They can be used in optimization problems where the search of optimal solutions is carried out in a space of coded solutions as finite-length strings. They were developed by John Holland at the University of Michigan [14] and are considered as a particular type of metaheuristic, a group of techniques that, according to the classification followed in [3], encompasses trajectory methods such as Tabu Search, Simulated Annealing or Iterated Local Search, and population-based methods such as Genetic Algorithms, Particle Swarm Optimization and Ant Colonies.

Genetic Algorithms handle populations consisting of a set of potential solutions, i.e. the algorithm maintains a population of n individuals $Pop(k) = \{x_1(k), \ldots, x_n(k)\}$ for each iteration *k*, where each individual represents a potential solution of the problem. Normally the initial population is randomly selected, but some knowledge about the specific problem can be used to include in the initial population special potential solutions in order to improve the convergence speed. The size of this initial population is one of the most important aspects to be considered and may be critical in many applications. If the size is too small, the algorithm may converge too quickly, and if it is too large the algorithm may waste computational resources. The population size is usually chosen to be constant although GA with varying population size are also possible. A study about the optimal population size can be found in [12]. Each individual in the population, i.e. potential solution, must be represented using a genetic representation. Commonly, a binary representation is used, however other approaches are possible. Each one of the potential solutions must be evaluated by means of a fitness function; the result of this evaluation is a measure of individual adaptation.

The algorithm is an iterative process in which new populations are obtained using a selection process (reproduction) based on individual adaptation and some "genetic" operators (crossover and mutation). The individuals with the best adaptation measure have more chance of reproducing and generating new individuals by crossing and muting. The reproduction operator can be implemented as a biased roulette wheel with slots weighted in proportion to individual adaptation values. The selection process is repeated m times and the selected individuals form a tentative new population for further genetic operator actions.

After reproduction some of the members of the new tentative population undergo transformations. A *crossover* operator creates two new individuals (*offsprings*) by combining parts from two randomly selected individuals of the population. In a GA the crossover operator is randomly applied with a specific probability, p_c. A good GA performance requires the choice of a high crossover probability. *Mutation* is a unitary transformation which creates, with certain probability, p_m, a new individual by a small change in a single individual. In this case, a good algorithm performance requires the choice of a low mutation probability (inversely proportional to the population size). The mutation operator guarantees that all the search space has a nonzero probability of being explored.

In spite of their surprising simplicity, GA have been recognized as a powerful tool to solve optimization problems in various fields of applications; examples of such problems can be found in a great variety of domains such as transportation problems, wire routing, travelling salesman problem [11, 13, 22].

Parent 1	0.123	0.178	0.274	**0.456**	0.571	0.701	0.789	0.843	0.921	0.950
Parent 2	0.086	0.167	0.197	**0.271**	0.367	0.521	0.679	0.781	0.812	0.912
				cross point						
Offspring 1	0.123	0.178	**0.274**	**0.271**	0.367	0.521	0.679	0.781	0.812	0.912
	"	"	**0.271**	**0.274**	"	"	"	"	"	"
				chromosomes sorting						
Offspring 2	0.086	0.167	0.197	**0.456**	0.571	0.701	0.789	0.843	0.921	0.950

Table 1: Crossover operator of the genetic algorithm

begin
  k=0
  random initialization of *Pop(k)*
  fitness evaluation of *Pop(k)*
  while (not termination condition) **do**
    Select individuals from *Pop(k)*
    Apply crossover and mutation operator with probabilities p_c and p_m respectively
    Set $Pop(k+1)$
    $k = k + 1$
  end
end

Table 2: General structure of the genetic algorithm

4 Using Genetic Algorithms for Data Fitting

In order to use GA for fitting curves/surfaces to data points, several aspects must be previously considered. Firstly, a typical GA requires two elements to be defined prior to its use: the genetic representation of each potential solution of the problem and a measure of the quality of the solution (usually referred to as the *fitness function*). In our problem, we are interested on the assignment process of parameter values to data points, so we propose the use of a real-coded genetic algorithm in which the genetic representation of an individual will be a real n_p-dimensional vector, where each coordinate represents the parameter value assigned to a data point. The fitness function that allows measuring the quality of an assignment will be based on the error function of the fitting process.

As initial population we will consider a randomly generated set of parameter vectors (individuals). To widen the search area of the algorithm it is desirable that the population size be large; however the computation time increases as this parameter rises, so a trade-off between both considerations is actually required.

The algorithm uses three genetic operators to obtain new populations of individuals: selection, crossover and mutation. In our case, the selection operator is implemented as the classical biased roulette wheel with slots weighted in proportion to individual fitness values. We use an one-point crossover operator that randomly selects a crossover point within an individual, then swaps the two parent chromosomes to the left and to the right from this point and eventually sorts the obtained vectors to produce two new offsprings. This process is illustrated in Table 1.

As mutation method we propose to select the position s with worst fit error in the vector parameter of the solution and change the value of the selected parameter by the arithmetic mean of the previous and next parameters in the vector, that is, $t_s = \frac{t_{s-1} + t_{s+1}}{2}$. Using these genetic operators, the general structure of the algorithm is described in Table 2.

The termination condition consists of establishing a threshold limit for the number of consecutive iterations without improving the fitting error in the set of geometric data points.

5 Best Least-Squares Approximation

Let us consider a set of 3D data points $\mathbf{D}_i = (x_i, y_i, z_i)$, $i = 1, \ldots, n_p$. We describe the procedure in more detail for the x's coordinates (the extension to y's and z's is immediate). The goal is to calculate the coefficients c_j^x, $j = 0, \ldots, n_b$ of (1) which give the best fit in the discrete least-squares sense to the column vector $\mathbf{X} = (x_1, \ldots, x_{n_p})^T$ where $(.)^T$ means transposition, by using the model

$$x(t) = \sum_{j=0}^{n_b} c_j^x B_j(t), \tag{2}$$

supposing that t_i $(i = 1, \ldots, n_p)$ are parameter values assigned to the data points and the $B_j(t)$ are the known blending functions of the model. Considering the column vectors $\mathbf{B}_j = (B_j(t_1), \ldots, B_j(t_{n_p}))^T$, $j = 0, \ldots, n_b$ and solving the following system gives the coefficients c_j^x:

$$\begin{pmatrix} \mathbf{B}_0^T.\mathbf{B}_0 & \ldots & \mathbf{B}_{n_b}^T.\mathbf{B}_0 \\ \vdots & \vdots & \vdots \\ \mathbf{B}_0^T.\mathbf{B}_{n_b} & \ldots & \mathbf{B}_{n_b}^T.\mathbf{B}_{n_b} \end{pmatrix} \begin{pmatrix} c_0^x \\ \vdots \\ c_{n_b}^x \end{pmatrix} = \begin{pmatrix} \mathbf{X}^T.\mathbf{B}_0 \\ \vdots \\ \mathbf{X}^T.\mathbf{B}_{n_b} \end{pmatrix} \tag{3}$$

The elements of the coefficient matrix and the independent terms are calculated by performing a standard Euclidean scalar product between finite-dimensional vectors. This system (3) results from minimizing the sum of squared

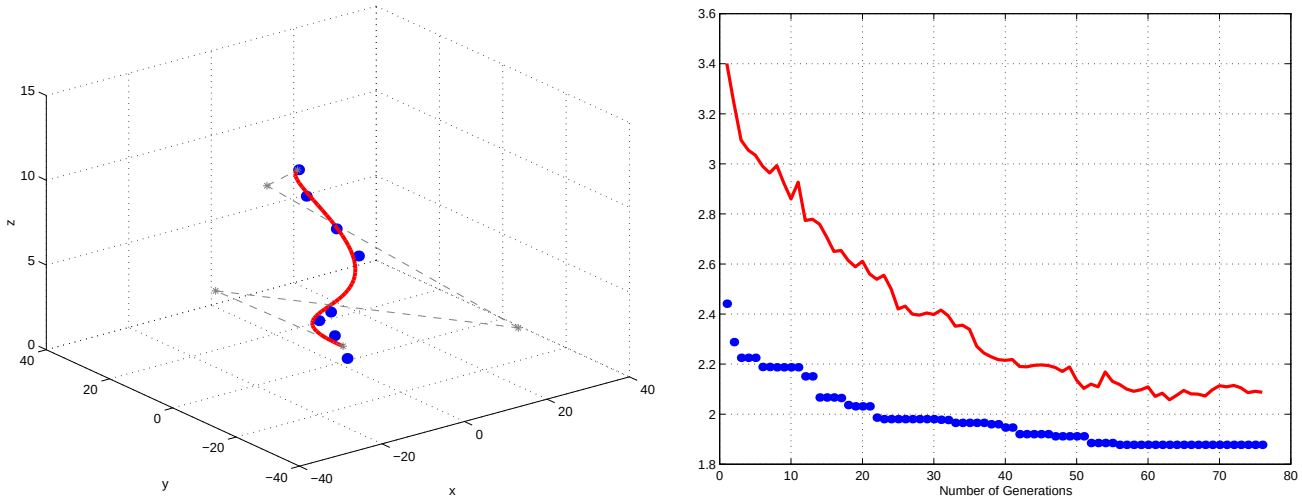

Figure 1: Bézier curve fitting through genetic algorithms: (left) Bézier curve, its control points (stars) and the data points (spheres); (right) evolution of the mean (solid line) and the best (dotted line) Euclidean errors along the generations

errors referred to the x_i coordinates of the data points, as indicated in Section 2. Considering all the x_i, y_i, z_i coordinates, the solution of the three linear systems with the same coefficient matrix provides the best least-squares approximation for the curve $\mathbf{C}(t) = \sum_{j=0}^{n_b} \mathbf{P}_j B_j(t)$, where the coefficients $\mathbf{P}_j = (c_j^x, c_j^y, c_j^z)$ represent 3D vectors.

For surfaces in parametric form, one uses the structure $\mathbf{S}(u,v) = \sum_{i=0}^{n_u}\sum_{j=0}^{n_v} \mathbf{P}_{i,j} B_i(u) B_j(v)$, which is a tensor-product surface, a very common model in CAGD. The coefficients $\mathbf{P}_{i,j}$ are the control points in 3D, arranged in a quadrilateral topology, and functions $B_i(u)$ and $B_j(v)$ are the same basis functions used for representing curves, for example Bernstein polynomials in this paper. The parameters u and v are valued on a rectangular domain $[u_m, u_M] \times [v_m, v_M]$, a Cartesian product of the respective domains for u and v. If $B_i(u)$ and $B_j(v)$ are Bézier basis functions, the $(n_u + 1).(n_v + 1)$ bivariate polynomials $B_{i,j}(u,v) = B_i(u).B_j(v)$, $i = 0, \ldots, n_u$, $j = 0, \ldots, n_v$ constitute a vector basis for a linear vector space of polynomials in u and v on the square domain $[0,1] \times [0,1]$. Given a cloud of points $(x_{l,k}, y_{l,k}, z_{l,k})$, in 3D, with a quadrilateral structure, $l = 1, \ldots, n_{p_u}$, $k = 1, \ldots, n_{p_v}$, and a set of parameter values (u_l, v_k) associated one-to-one with the data points in the cloud such that these points form a cartesian set in the parameter domain, a discrete formulation similar to that for fitting points to a curve can be made. The best least-squares tensor-product surface fitting the points can be obtained using the system (3), in which the role of the **B**'s is now assumed by the bivariate basis functions $B_{i,j}(u,v)$ described earlier.

6 Genetic Algorithm Examples

In this section two examples (a Bézier curve and a Bézier surface) aimed at showing the performance of the GA method are discussed.

6.1 Fitting a Bézier curve

As a first example we consider a Bézier curve of degree d whose parametric representation is given by Eq. (2) where the basis functions of degree d are defined as: $B_i^d(t) = \binom{d}{i} t^i (1-t)^{d-i}$, $i = 0, \ldots, d$ and $t \in [0,1]$. In this example, we have chosen a set of eigth 3D points to be fitted to a Bézier curve of degree $d = 4$. The unknowns are 23 scalar values: a vector of 8 parameter values associated with the 8 data points, plus 15 coefficients (3 for the coordinates of each of the 5 control points of the curve of degree 4). The data for the genetic algorithm are set as follows: we select an initial population of 100 parameter vectors, each having 8 elements generated randomly from a Uniform$[0,1]$ distribution and sorted in increasing order. Then, we apply the procedure shown in Table 2 to produce successive generations. In this example, the crossover and mutation operators are applied with probabilities $p_c = 0.80$ and $p_m = 0.20$, respectively. A typical output for a couple of parent chromosomes is shown in Table 1, yielding two new offsprings in the next generation.

Regarding our termination condition, these steps are repeated until the results no longer change for 20 successive iterations. In this example, the optimal fit is attained at the $76th$ generation with the following results: the error in the population (the maximum point error in the best fitted parameter vector) is 1.8774, while the mean error in the population is 2.0875. The number of crossovers

#	# iter.	Best error	Mean error	CPU time (secs)
1	**73**	**1.832**	**2.104**	**1.1718**
2	51	1.858	1.891	0.8281
3	40	1.904	2.116	0.6562
4	50	1.858	2.092	0.8281
5	33	1.870	2.242	0.5625
6	45	1.898	2.086	0.7656
7	64	1.874	2.041	1.0625
8	58	1.844	2.055	0.9531
9	37	1.842	2.017	0.6250
10	59	1.877	2.118	0.9531
11	59	1.986	2.150	0.9531
12	44	1.835	2.064	0.7500
13	56	1.837	2.017	0.9218
14	59	1.850	2.297	0.9531
15	61	1.897	2.085	1.0156
16	42	1.861	2.148	0.7031
17	37	1.968	2.166	0.6093
18	41	1.960	2.071	0.6875
19	56	1.880	2.045	0.9218
20	49	1.882	2.161	0.8281

Table 3: 20 executions of a Bézier curve fitting through genetic algorithms

#	# iter.	Best error	Mean error	CPU time (secs)
1	127	1.271	2.287	17.5156
2	104	2.087	3.055	16.4687
3	80	1.259	2.080	12.9687
4	53	2.282	3.628	8.3750
5	86	1.524	2.337	12.9687
6	58	2.062	2.895	8.8125
7	188	1.623	2.180	27.8593
8	**157**	**1.027**	**1.723**	**24.4375**
9	160	1.409	1.913	26.0937
10	109	1.190	2.018	17.2656
11	101	1.083	2.028	14.9218
12	108	1.630	2.694	16.1718
13	100	1.238	2.068	14.4687
14	109	1.963	2.984	16.4687
15	150	1.123	1.775	22.3281
16	83	1.214	1.861	12.4843
17	38	1.245	2.763	5.7656
18	78	1.714	2.297	11.9218
19	149	1.457	1.863	21.3750
20	82	1.403	2.457	12.1562

Table 4: 20 executions of a Bézier surface fitting through genetic algorithms

and mutations for the last generation are 46 and 24, respectively. The optimum parameter vector obtained is $[0, 0.0131, 0.0583, 0.3556, 0.5384, 0.7138, 0.7899, 1]$. The computation time for this example (in Matlab version 7.0, running on a Pentium IV, 3 GHz and with 1GB RAM) has been 1.22 seconds.

Fig 1(left) shows the data points (represented as spheres), the fourth-degree 3D Bézier fitting curve and its 5 control points (represented as stars). Fig 1(right) shows the evolution of the mean (solid line) and the best (dotted line) Euclidean errors of the parameter vectors for each generation along the successive generations. Note that the best error becomes small very quickly at the beginning, the reduction rate getting slower for later generations.

To check how the randomness inherent in GA affects the results, we performed 20 executions on the previous example for a population size of 100 candidate solutions (generated as vectors of Uniform (0,1) random numbers sorted in increasing order) and probability values of 0.80 for the crossover and 0.20 for the mutation. The termination criteria is that of not improving solution after 20 consecutive iterations.

The obtained results are reported in Table 3. Columns of this table show the example number, number of iterations, the best and mean errors and the computation time (in seconds), respectively. For the sake of clarity, the execution with the best result has been boldfaced. Note that the best and mean errors take relatively close (although slightly different) values. Note also that the number of iterations of these examples for our choice of parameters ranges from 37 to 73 leading to computation times of about $0.7 \sim 1.2$ seconds for the configuration described above.

6.2 Fitting a Bézier surface

We consider now a parametric Bézier surface of degree n_u in u and n_v in v whose representation is given by:

$$\mathbf{S}(u,v) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} \mathbf{P}_{i,j} B_i^{n_u}(u) B_j^{n_v}(v) \qquad (4)$$

where the basis functions (the Bernstein polynomials) are defined as above and the coefficients $\mathbf{P}_{i,j}$ are the surface control points. For this example we consider an input of 256 data points generated from a Bézier surface as follows: for the u's and v's of data points, we choose two groups of 8 equidistant parameter values in the intervals $[0, 0.2]$ and $[0.8, 1]$. Our goal is to reconstruct the surface which the given points come from. To this purpose, we consider a bicubic Bézier surface, so the unknowns are $3 \times 16 = 48$ scalar coefficients (3 coordinates for each of 16 control points) and two parameter vectors for u and v (each of size 16) associated with the 256 data points. That makes a total of 80 scalar unknowns.

The input parameters for the procedure are as follows: population size: 200; $p_c = 0.95$; $p_m = 0.20$; termination criteria =no improvement after 20 consecutive generations. Initially, we have a population of 200 $\mathcal{U}$-vectors and 200 $\mathcal{V}$-vectors, each one constructed by assigning random parame-

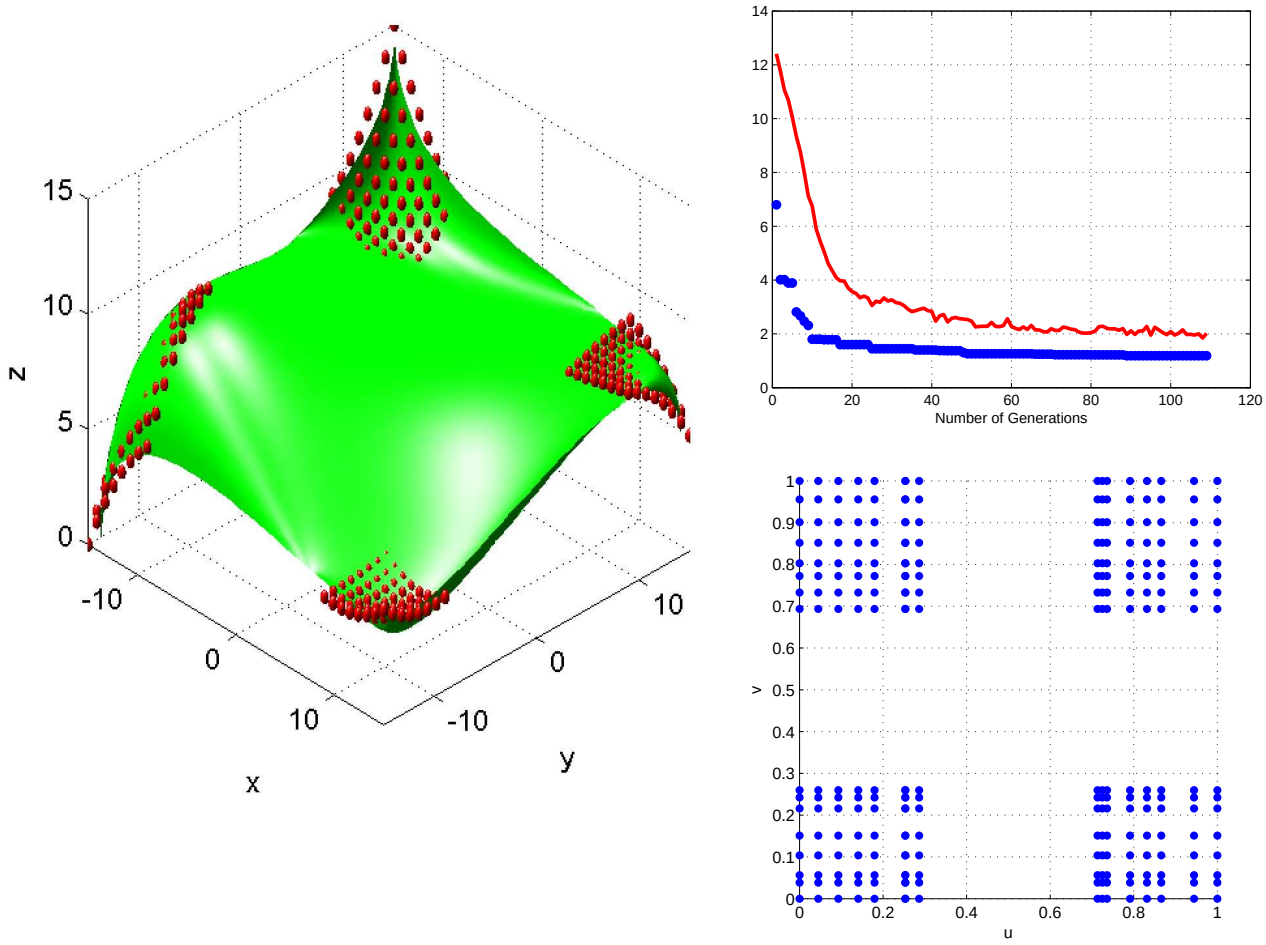

Figure 2: Bézier surface fitting through genetic algorithms: (left) bicubic Bézier surface and data points; (right-top): evolution of the mean (solid line) and the best (dotted line) Euclidean errors along the generations; (right-bottom): optimum parameter values for u and v on the parametric domain

ter values with Uniform$[0, 1]$ distribution, and sorting them within each vector. The best solution is attained at generation 108 with the following results: best error in the fit: 1.1922; mean error: 1.9831; number of crossovers (resp. mutations) for the last generation: 56 (resp. 31); computation time (Pentium IV, 3 GHz, 1GB RAM and running Matlab v7.0): 16.28 seconds.

Fig. 2(left) shows the data points and the bicubic Bézier fitting surface. In Fig. 2(right-top) we display the evolution of mean error (solid line) and best (dotted line) distance error for each generation along the iterations. The optimum parameter values for u and v are depicted in Fig. 2(right-bottom) where one can see how the fitting process grasps the distribution of parameter values assigned to the data points. It is worthwhile to mention the tendency of the obtained parameter values, initially uniformly distributed on the unit square, to concentrate at the corners of such unit square parameter domain, thus adjusting well the input information.

Similarly to the case of curves, we carried out 20 executions on the surface example for a population size of 200 candidate solutions, crossover probability $p_c = 0.95$, mutation probability $p_m = 0.20$ and a limit termination criteria of 20 consecutive iterations with no change. The corresponding results are listed in Table 4 with the same meaning for the columns as in Table 3.

7 Particle Swarm Optimization

Particle Swarm Optimization (PSO) techniques come from a model to imitate the behaviour of a flock of birds, for instance, when moving all together following a common tendency in their displacements. They incorporate both a global tendency for the movement of the set of individuals and local influences from neighbors [7, 18]. Similarly to Genetic Algorithms, PSO procedures start by choosing a population (swarm) of random candidate solutions, called *particles*. But they are displaced throughout their domain looking for an optimum taking into account global and local influences, the latest coming form the neighborhood of each particle.

The dynamics of the particle swarm is considered along successive iterations, like time instances. Each particle modifies its position P_i along the iterations, keeping track of its best position in the variables domain implied in the problem. This is made by storing for each particle the coordinates P_i^b associated with the best solution (fitness) it has achieved so far along with the corresponding fitness value, f_i^b. These values account for the *memory* of the best particle position. In addition, members of a swarm can communicate good positions to each other, so they can adjust their own position and velocity according to this information. To this purpose, we also collect the best fitness value among all the particles in the population, f_g^b, and its position P_g^b from the initial iteration. This is a global information for modifying the position of each particle.

Alternatively, an additional information can be used, consisting of the best fitness value, lf_i^b, and position, lP_i^b, attained within a local group of neighbors of particle i, being the number of neighbors a control parameter for the process. This alternative allows parallel exploration of the search space while reducing the probability of the PSO to fall into local minima, at the price of slow convergence speed. In general, smaller neighborhoods lead to slower convergence while larger neighborhoods yield faster convergence. Because of this reason, most PSO methods consider the global approach (i.e. the entire swarm) instead of a local approach (the neighborhood of each particle). In such a case, the evolution for each particle i is given by:

$$\begin{cases} V_i(k+1) = & w\,V_i(k) + C_1R_1(P_g^b(k) - P_i(k)) + \\ & C_2R_2(P_i^b(k) - P_i(k)) \\ P_i(k+1) = & P_i(k) + V_i(k) \end{cases} \tag{5}$$

where $P_i(k)$ and $V_i(k)$ are the position and the velocity of particle i at time k, w is called *inertia weight* and decide how much the old velocity will affected the new one and coefficients C_1 and C_2 are constant values called *learning factors*, which decide the degree of affection of P_g^b and P_i^b. In particular, C_1 is a weight that accounts for the "social" component, while C_2 represents the "cognitive" component, accounting for the memory of an individual particle along the time. Two random numbers, R_1 and R_2, with uniform distribution on $[0, 1]$ are included to enrich the searching space. Finally, a fitness function (similar to that for genetic algorithms) must be given to evaluate the quality of a position. The termination condition is also the same used for the genetic algorithm. This final PSO procedure is briefly sketched in Table 5.

```
begin
    k=0
    random initialization of individual positions P_i and
      velocities V_i in Pop(k)
    fitness evaluation of Pop(k)
    while (not termination condition) do
        Calculate best fitness particle P_g^b
        for each particle i in Pop(k) do
            Calculate particle position lP_i^b with best fitness
              in the neighborhood of particle i
            Calculate velocity V_i for particle i according
              to first equation of (5)
            while not feasible P_i + V_i do
              Apply scale factor to V_i
            end
            Update position P_i according to second
              equation of (5)
        end
        k = k + 1
    end
end
```

Table 5: General structure of the particle swarm optimization algorithm

8 Particle Swarm Optimization Examples

In this section the two previous examples of Section 6 are analyzed from the new perspective of Particle Swarm Optimization.

8.1 Fitting a Bézier curve

We consider the same set of 8 data points used in Section 6.1 and a fourth-degree Bézier fitting curve. The parameter values for the PSO algorithm are: population size: 100 individuals or particles, where each one is a vector with 8 components initially taken as an increasing sequence of random uniform numbers on [0,1]; inertia coefficient $w = 1$; coefficient for the global influence $C_1 = 0.2$; coefficient for the neighbors local influence $C_2 = 0.8$; number of neighbors locally influencing each particle: 5; and limit for not improving error iterations: 10. With these parameters, we obtain a curve with: best fitting error: 1.8104; mean error: 1.8106; number of iterations: 192; computation time (Pentium IV, 3 GHz, 1 GB. RAM, running Matlab v7.0): 2.6562 seconds. For these values, the optimum parameter vector is: [0.0028,0.0420, 0.0841, 0.2756, 0.4866, 0.7155, 0.8056, 0.9978]. The resulting curve is very similar to that in Fig. 1 and hence it is not displayed here.

Figure 3 shows the evolution of the mean and best Euclidean errors along the successive iterations. In comparison with the results for GA reported in Section 6.1, we can see that best and mean error are closer each other for PSO than for GA. This is because the particles or individuals in PSO tend to move all together to a closer position where the optimum is attained, while in GA the population individuals maintain a greater dispersion in their spatial distribution.

Table 6 shows 20 executions of the PSO algorithm for this example and the same parameters used in the previous paragraphs. Comparing it with Table 3 we can notice that

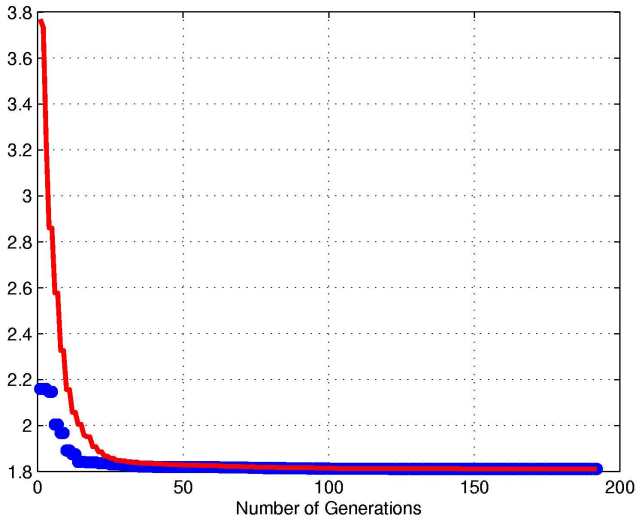

Figure 3: Mean (solid line) and best (dotted line) errors along the generations of a Bézier curve with the particle swarm optimization algorithm

#	# iter.	Best error	Mean error	CPU time (secs)
1	53	1.823	1.833	0.9687
2	**198**	**1.810**	**1.810**	**2.8281**
3	20	1.830	1.877	0.5156
4	42	1.815	1.838	0.8125
5	**172**	**1.810**	**1.810**	**2.1718**
6	46	1.813	1.826	0.7968
7	78	1.811	1.812	1.2187
8	**140**	**1.810**	**1.811**	**2.0468**
9	**187**	**1.810**	**1.810**	**2.4687**
10	65	1.813	1.820	1.0156
11	**104**	**1.810**	**1.810**	**1.5312**
12	200	1.811	1.813	2.6875
13	**146**	**1.810**	**1.810**	**2.0937**
14	**137**	**1.810**	**1.810**	**1.8437**
15	105	1.813	1.817	1.5122
16	34	1.820	1.853	0.6718
17	45	1.814	1.829	0.8125
18	**76**	**1.810**	**1.811**	**1.1406**
19	152	1.811	1.811	2.2031
20	89	1.811	1.813	1.3906

Table 6: 20 executions of a Bézier curve fitting through particle swarm optimization

PSO improves a little bit GA. Although in both cases the errors are large, they are smaller for PSO. Furthermore, from Table 6 we can see that there are many examples reaching the same best value, meaning that the obtained result is likely quite close to the real one. This implies that PSO is able to yield a very good approximation of the optimal solution. Such good results are obtained for a larger number of iterations in the PSO executions thus leading to larger computation times as well. One observation is that PSO introduces more variation in the values of the variables, and that increases the time required for the stabilization of the fitting error value.

8.2 Fitting a Bézier surface

For comparative purposes, we consider the same set of 256 point data used in Section 6.2, and try to fit a bicubic Bézier surface. The input parameter values for the PSO algorithm are: population size: 200 individuals or particles, where each particle is represented by two vectors, U and V, each with 16 components initialized with random uniform values on [0, 1] sorted in increasing order; inertia coefficient $w = 1$; coefficient for the global influence: $C_1 = 0.2$; coefficient for the neighbors local influence $C_2 = 0.8$; number of neighbours influencing locally to each particle: 20 and threshold limit for not improving iterations: 10.

Figure 4(left) shows the obtained surface and the data points. In Fig. 4(right-top) we display the evolution of mean error (solid line) and best (dotted line) distance error for each generation along the iterations, while the optimum parameter values for u and v are shown in Fig. 4(right-bottom). The convergence in this example is attained at iteration 473 at which the best and mean errors are 0.287 and 0.289 respectively. The computation time for this example is 48.7 seconds.

Visual comparison of Figures 2 and 4 shows that the PSO algorithm outperforms substantially GA for the given example. Points at the corners (that sometimes fall outside the surface with GA) are now very well fitted and the surface is much closer to the cloud of points as evidenced by the smaller error in comparison with GA. Further, the surface obtained with GA exhibit a more intricate structure at the boundaries (occasionally leading to self-intersections at the corners for some executions). By contrast, the surfaces obtained with PSO do reflect much better the real geometry of the data while preserving better the smoothness at the boundaries. Another remarkable issue concerns the distance between the best and the mean error for each iteration. In our PSO example, they converge to roughly the same value, as opposed to the GA example (see Figs. 2-4(right-top) for comparison). Finally, the distribution of points, (Figs. 2-4(right-bottom)) is more uniform for the PSO example and fits better into the intervals $[0, 0.2]$ and $[0.8, 1]$ for both u and v (although the obtained intervals are still slightly larger). One reason to explain this is that the number of iterations until reaching the prescribed convergence is larger for PSO. In short, better quality is achieved at the price of larger computation times.

Table 7 reports the results of 20 executions on the surface example for a population size of 200 candidate solutions, inertia coefficient $w = 1$; coefficient for the global influence: $C_1 = 0.2$; coefficient for the neighbors local in-

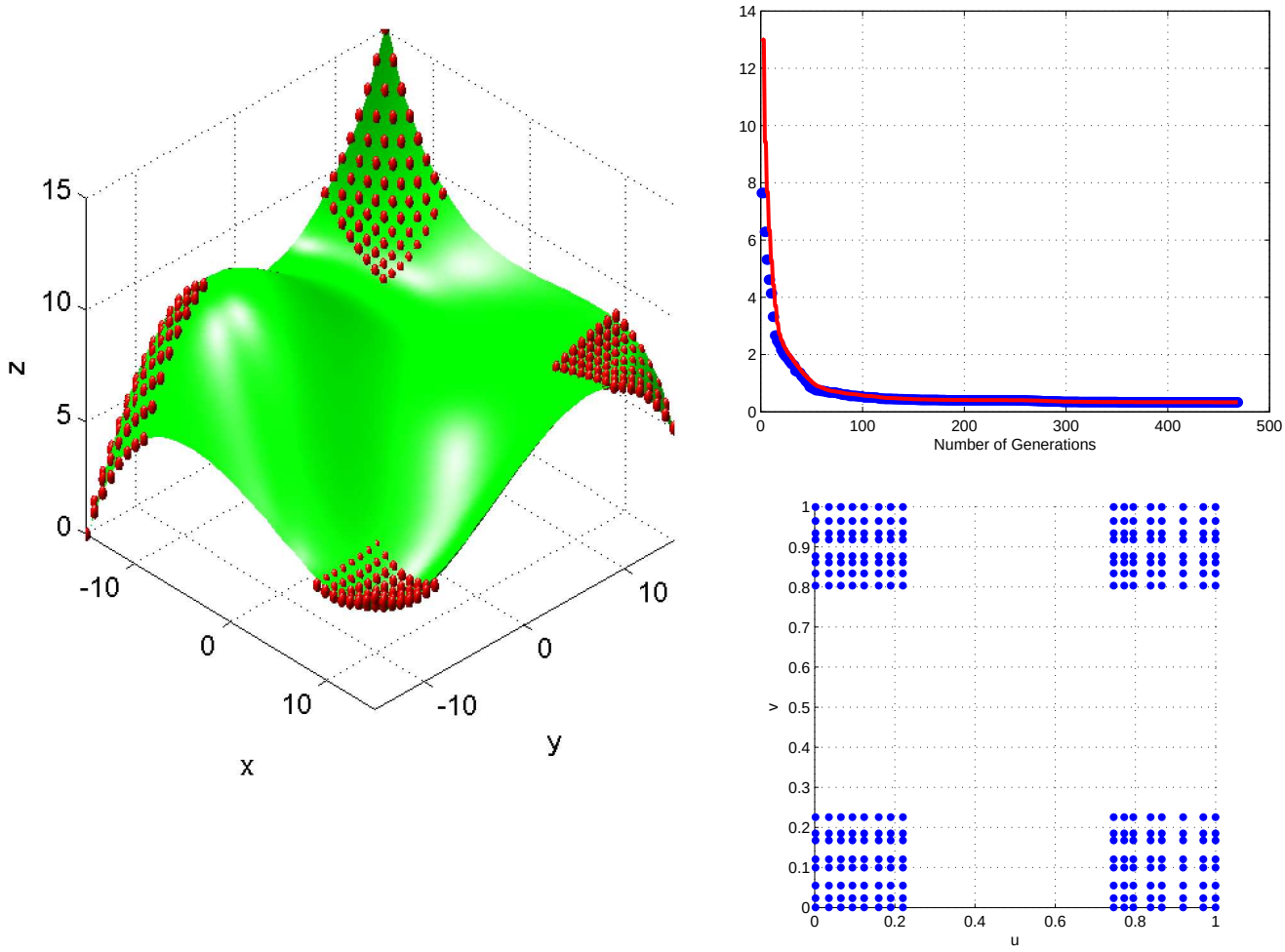

Figure 4: Bézier surface fitting through particle swarm optimization: (left) bicubic Bézier surface and data points; (right-top): evolution of the mean (solid line) and the best (dotted line) Euclidean errors along the generations; (right-bottom): optimum parameter values for u and v on the parametric domain

fluence $C_2 = 0.8$; number of neighbours influencing locally to each particle: 10 and threshold limit for not improving iterations: 10. In general, these results are in good agreement with our previous assertions.

9 Conclusions and Future Work

In this paper we address the problem of fitting 3D data points by means of Bézier curves and surfaces by following a novel approach. Basically, it combines the error minimization by least-squares with artificial intelligence methods of biological inspiration: Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). In our opinion, both methods can readily be implemented and provide the user with good quality solutions at reasonable execution times. In our trials, PSO generally outperforms GA. We speculate that the dual weighted contribution to the evolution of every particle given, on one hand, by the distance from such a particle to the global best of the population (or alternative, its neighborhood) and, on the other hand, by the local influence of its best over the time, seems to lead the swarm of particles towards the optimum in a very natural way. However, further research is still needed in order to elucidate the rationale of this behavior at full extent.

It is worthwhile to mention that in our experiments neighborhood is based on index proximity of particles. In sorted points along a curve this strategy reveals to be adequate. For surfaces whose points are arranged in a quadrilateral topology this approach could also be suitable, since the proximity of indices would imply Euclidean distance proximity between particles in the domain space of the problem. If the cloud of points to be fitted by a surface has not a quadrilateral structure, one feasible alternative is to obtain a quadrilateral data mesh from the given data, by applying some criteria like point proximity, for example. If no quadrilateral structure can eventually be adapted, the process described above could also be applied, but problems might appear with the condition of the resulting system of

#	# iter.	Best error	Mean error	CPU time (secs)
1	185	0.448	0.466	19.2031
2	488	0.274	0.275	50.6406
3	368	0.350	0.351	37.7968
4	441	0.259	0.260	45.2812
5	372	0.889	0.890	35.1875
6	242	0.390	0.397	24.2343
7	264	0.420	0.420	26.1562
8	261	0.685	0.692	26.5625
9	143	0.633	0.659	14.0625
10	242	0.505	0.510	24.9843
11	382	0.544	0.550	38.2812
12	135	0.633	0.646	15.4843
13	136	0.552	0.559	13.1718
14	109	0.634	0.654	10.8281
15	106	0.378	0.434	11.5000
16	143	0.845	0.853	15.6562
17	289	0.368	0.369	26.9687
18	**344**	**0.207**	**0.209**	**35.0156**
19	289	0.421	0.422	28.9375
20	413	0.319	0.322	39.6562

Table 7: 20 executions of a Bézier surface fitting through particle swarm optimization

equations.

Of course, other families of piecewise polynomial models such as B-spline or NURBS can be used to fit the data, with some changes in the computational process to handle the knot vectors, which are additional parameters in these models. In this case, the distance error function for fitting data to the different models could exhibit multiple relative minima. This makes more difficult attaining a global optimum. Some ideas on how to improve globally the search process are desirable.

Obviously, all results reported in this paper are affected by a certain level of randomness according to the probabilistic and random factors inherent to GA and PSO schemes. This means that our results should be regarded as an average behavior, rather than as deterministic rules. From this point of view, an additional issue is to determine the optimal values for the parameters of the GA/PSO models in order to achieve the best performance. Although we have already done some preliminary experiments, further research is still needed. We are currently working on these issues. Our ongoing results will be published elsewhere.

Acknowledgments

The authors would like to thank the financial support from the SistIng-Alfa project, Ref: ALFA II-0321-FA of the European Union and from the Spanish Ministry of Education and Science, Projects MTM2005-00287 (Mathematics National Program) and TIN2006-13615 (Computer Science National Program). Financial support from the University of Cantabria is also kindly acknowledged.

Bibliography

[1] Barhak, J., Fischer, A.: Parameterization and reconstruction from 3D scattered points based on neural network and PDE techniques. IEEE Trans. on Visualization and Computer Graphics, **7**(1) (2001) 1-16.

[2] Bradley, C., Vickers, G.W.: Free-form surface reconstruction for machine vision rapid prototyping. Optical Engineering, **32**(9) (1993) 2191-2200.

[3] Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys, **35**(3) (2003) 268-308.

[4] Dahlquist, G., Bjrck, A.: Numerical Methods. Prentice Hall (1974).

[5] de Boor, C. A.: Practical Guide to Splines. Springer-Verlag (2001).

[6] Draper, N. R., Smith, H.: Applied Regression Analysis, 3rd ed. Wiley-Interscience (1998).

[7] Eberhart, R. C., Kennedy, J.: A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan (1995) 39-43.

[8] Eberhart R.C., Shi Y.: Particle swarm optimization: developments, applications and resources Proceedings of the 2001 Congress on Evolutionary Computation (2001) 81-86.

[9] Echevarría, G., Iglesias, A., Gálvez, A.: Extending neural networks for B-spline surface reconstruction. Lectures Notes in Computer Science, **2330** (2002) 305-314.

[10] El-Mounayri H., Kishawy H., Tandon V.: Optimized CNC end-milling: a practical approach. International Journal of Computer Integrated Manufacturing, **15**(5) (2002) 453-470.

[11] Gálvez, A., Iglesias, A., Cobo, A., Puig-Pey, J., Espinola, J.: Bézier curve and surface fitting of 3D point clouds through genetic algorithms, functional networks and least-squares approximation. Lectures Notes in Computer Science, **4706** (2007) 680-693.

[12] Goldberg, D.E.: Optimal Initial Population Size for Binary-Coded Genetic Algorithms, TCGA Report No.85001. University of Alabama (1985).

[13] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989).

[14] Holland, J.H.: Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press (1975).

[15] Hoschek, J., Lasser, D.: Fundamentals of Computer Aided Geometric Design. A.K. Peters (1993).

[16] Iglesias, A., Gálvez, A.: A new artificial intelligence paradigm for computer aided geometric design. Lectures Notes in Artificial Intelligence, **1930** (2001) 200-213.

[17] Iglesias, A., Echevarría, G., Gálvez, A.: Functional networks for B-spline surface reconstruction. Future Generation Computer Systems, **20**(8) (2004) 1337-1353.

[18] Kennedy, J., Eberhart, R. C.: Particle swarm optimization. IEEE International Conference on Neural Networks, Perth, Australia (1995) 1942-1948.

[19] Kennedy J., Eberhart R. C., Shi, Y.: Swarm Intelligence, San Francisco: Morgan Kaufmann Publishers (2001).

[20] Marinov M., Kobbelt, K.: A Robust Two-Step Procedure for Quad-Dominant Remeshing. Computer Graphics Forum, **25**(3), (2006) 537-546.

[21] Pottmann, H., Leopoldseder, S. Hofer, M., Steiner, T., Wang, W.: Industrial geometry: recent advances and applications in CAD. Computer-Aided Design **37** (2005) 751-766.

[22] Renner, G. Ekárt, A.: Genetic algorithms in computer aided design. Computer-Aided Design **35** (2003) 709-726.

[23] Varady, T., Martin, R.: Reverse Engineering. In: Farin, G., Hoschek, J., Kim, M. (eds.): Handbook of Computer Aided Geometric Design. Elsevier Science (2002).

[24] Vaz I.F., Vicente L.N.: A particle swarm pattern search method for bound constrained global optimization. Journal of Global Optimization, **39** (2007) 197-219.

[25] Weiss, V., Andor, L., Renner, G., Varady, T.: Advanced surface fitting techniques. Computer Aided Geometric Design, **19** (2002) 19-42.

[26] Yoshimoto F., Harada T., Yoshimoto Y.: Data fitting with a spline using a real-coded algorithm. Computer Aided Design, **35** (2003) 751-760.

About the authors

ANGEL COBO is an Associate Professor at the Faculty of Economics of the University of Cantabria. He obtained a B.Sc. and a Ph.D. (1993) in Mathematics at the University of Cantabria, Spain. He has authored or coauthored 24 scientific articles and books in Applied Mathematics and Computer Science. His research topics are operations research, metaheuristics and application of bio-inspired techniques in information management.

AKEMI GALVEZ TOMIDA is a lecturer at the Department of Applied Mathematics and Computational Sciences of the University of Cantabria (Spain). She holds a B.Sc. degree in Chemical Engineering at the National University of Trujillo (Peru), a M.Sc. and a Ph.D. degrees in Computational Sciences at the University of Cantabria (Spain). She has published several papers on geometric processing, surface reconstruction and symbolic computation and participated in national and international projects on geometric processing and its applications to the automotive industry. Her fields of interest also include Chemical Engineering, numerical/symbolic computation and industrial applications.

JAIME PUIG-PEY is a Full Professor of Applied Mathematics and Computing at the University of Cantabria (Spain). He obtained a M.Sc. (1975) and a Ph.D. (1977) degrees in Civil Engineering, and a M.Sc. in Mathematics and Statistics in 1981. He currently teaches Numerical Methods to Civil Engineering students and postgraduate courses on CAGD. His research interests include Computer Aided Geometric Design (CAGD) from the point of view of numerical calculations, in particular curves on surfaces (intersection and other characteristic lines), and reconstruction of CAD curves and surfaces from clouds of points.

ANDRES IGLESIAS is an Associate Professor at the Department of Applied Mathematics and Computational Sciences of the University of Cantabria (Spain). He holds a B.Sc. degree in Mathematics (1992) and a Ph.D. in Applied Mathematics (1995). He has been the chairman and organizer of some international conferences in the fields of computer graphics, geometric modeling and symbolic computation, such as the CGGM (2002-08), TSCG (2003-08) and CASA (2003-08) annual conference series. In addition, he has served as program committee and/or steering committee member of 65 international conferences such as 3IA, CGA, CGIV, CIT, CyberWorlds, GMAG, GMAI, GMVAG, Graphicon, GRAPP, ICCS, ICCSA, ICICS, ICCIT, ICM, ICMS, IMS, ISVD, MMM and WSCG, and reviewer of 70 conferences and 13 journals. He has been guest editor of some special issues of international journals about computer graphics and symbolic computation. He is the author of over 90 international papers and four books. For more information, see: *http://personales.unican.es/iglesias*

JESUS ESPINOLA currently teaches Mathematics at UNASAM University, Huaraz (Peru). He got a B.Sc. in Mathematics from the University of Trujillo (Peru) and a Ph.D. from the University of Cantabria (Spain). He also worked as a researcher, developing mathematical algorithms at Candemat (a company that makes dies for automotive industry) and also at the University of Cantabria. His research interests include curve and surface modeling and computer graphics.