Postprint: Improved Algorithm for B-spline Curve Fitting Based on Genetic Algorithm

Authors: Gao Maoting, Feng Li

Date: 2018-06-19T00:00:00+00:00

Abstract

B-spline curve fitting, applied to depict the variation trends of discrete data points and generally obtained through data approximation or iterative methods, constitutes an important component in image processing and reverse engineering. To address issues such as multi-peak features, cusp points, and discontinuities in the curve to be fitted, a B-spline curve fitting algorithm based on genetic algorithms is proposed. The approach first employs penalty functions to transform the constrained curve optimization problem into an unconstrained formulation, then utilizes an improved genetic algorithm to select an appropriate fitness function, and further combines a simulated annealing algorithm to adaptively adjust the number and position of knots, thereby finding the optimal knot vector during the optimization process through continuous iteration until a final high-quality reconstruction curve is generated. Experimental results demonstrate that the algorithm effectively improves accuracy and accelerates convergence speed.

Full Text

Preamble

Title: Improved B-spline Curve Fitting Algorithm Based on Genetic Algorithm

Authors: Gao Maoting, Feng Li
Affiliation: College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China

Abstract: B-spline curve fitting is widely used to depict the variation trends of discrete data points, typically achieved through data approximation or iterative methods, and represents a critical component in image processing and reverse engineering. To address challenges such as multi-peak values, cusp points, and discontinuities in target curves, this paper proposes a B-spline curve fitting algorithm based on genetic algorithms. The approach first employs penalty functions to transform the constrained curve optimization problem into an unconstrained

one, then utilizes an improved genetic algorithm to select an appropriate fitness function. By integrating simulated annealing, the algorithm adaptively adjusts both the number and positions of nodes, identifying the optimal node vector during the optimization process through continuous iteration until a final high-quality reconstruction curve is generated. Experimental results demonstrate that the algorithm effectively improves accuracy and accelerates convergence speed.

Keywords: curve fitting; penalty function; genetic algorithm; node vector

0 Introduction

Curve fitting is a data processing method that approximates or models the functional relationship between a set of discrete point coordinates using continuous curves, with broad applications in reverse engineering and image processing. Curve fitting algorithms fall into two main categories: parametric fitting and non-parametric fitting. Parametric fitting represents dependent and independent variables through parametric equations. The commonly used least squares method [**?**], which minimizes the sum of squared errors, serves as a fundamental parameter estimation technique for linear models and plays a vital role in both theoretical research and engineering applications, while also forming the basis for many more complex methods. Non-parametric fitting directly identifies the functional relationship between variables, typically using interpolation methods such as Newton' s interpolation polynomial, divided differences with Newton interpolation, and piecewise low-order interpolation.

Bezier curves are widely employed in engineering applications due to their desirable properties including symmetry, endpoint interpolation, convex hull property, geometric invariance, and variation diminishing. In practice, piecewise low-order Bezier curve segments are often stitched together, but this approach lacks flexibility—the curve degree strictly depends on the number of data points defining the curve, and local performance is poor; modifying a curve segment requires adjusting all control point positions. B-spline curves [**?**], as a generalization of Bezier curves, retain the advantages of Bezier curves while enhancing local modifiability, enabling easy solution of continuity problems and facilitating curve reconstruction.

For B-spline curve fitting problems involving multi-peak values, nonlinear characteristics, cusp points, and discontinuities, this paper proposes an improved genetic algorithm-based B-spline curve fitting method. The approach controls error through a least squares model and uses genetic algorithms to select optimal node vectors. During genetic optimization, simulated annealing [**?**] and improved genetic operators are combined to overcome premature convergence, slow convergence, and insufficient precision issues.

1 Related Research

B-spline theory was first proposed by Schoenberg in 1946. Subsequently, Boor and Cox independently defined the standard B-spline algorithm in 1972, after which Gordon and Riesenfeld applied B-splines to shape description, leading to the development of B-spline curves.

Numerous scholars have conducted extensive research on B-spline curve fitting, proposing various methods. Piegl et al. [**?**] developed a B-spline curve interpolation algorithm with point-tangent constraints. Lin et al. proposed a geometric algorithm for cubic B-spline curves based on curvature conditions from tangents. Xiao et al. [**?**] introduced a method optimizing data point parameters using iterative closest points. Li et al. [**?**] presented a B-spline curve fitting algorithm using minimal data points under given error constraints. Duan et al. [**?**] combined curve interpolation with B-spline curve fitting, adding data points to ensure curve locality. Beyond traditional equation-based or geometric approaches, various artificial intelligence algorithms have been integrated for optimization. Yoshimoto et al. employed real-coded genetic algorithms for unconstrained fitting problems. Sun et al. [**?**] established a relationship between data parameters and node vectors to improve genetic algorithm fitness and optimize population evolution. Xu et al. [**?**] addressed discontinuous curves by combining chaotic ant swarm optimization to adjust free knot positions. Gálvez et al. [**?**] built upon genetic algorithm fitting, applying particle swarm optimization to dynamically determine vector length and node count for curve optimization.

While these algorithms achieve optimization to varying degrees, issues regarding precision, convergence speed, and premature convergence in genetic algorithms require further improvement. Most methods prioritize fitting accuracy as the optimization objective while considering convergence speed and the inherent limitations of artificial intelligence algorithms, ultimately transforming the problem into a multi-constrained nonlinear optimization problem.

2.1 Least Squares Model

Given a non-decreasing sequence $U : u_0 \leq u_1 \leq \cdots \leq u_{n+k+1}$, where U represents the node vector, the basis functions of the B-spline function are defined as in equations (1) and (2):

$$N_{i,0}(t) = \begin{cases} 1 & \text{if } u_i \leq t < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(t) = \frac{t - u_i}{u_{i+1} - u_i} N_{i,k-1}(t) + \frac{u_{i+k+1} - t}{u_{i+k+1} - u_{i+1}} N_{i+1,k-1}(t)$$

In equation (2), k denotes the degree of the B-spline curve. The parametric equation of the B-spline is:

$$p(t) = \sum_{i=0}^{n} d_i N_{i,k}(t)$$

where d_i represents the control point coordinate vector of the B-spline curve. If the data point set $P = \{p_j : j = 1, 2, \cdots, r\}$ lies on this curve, then point p_j satisfies:

$$p(t_j) = \sum_{i=0}^{n} d_i N_{i,k}(t_j)$$

Rewriting equation (4) in matrix form yields:

$$P = N \cdot D$$

where P denotes an $r \times 3$ matrix of r data points; N represents an $r \times n$ matrix of B-spline basis function coefficients; and D denotes a control point matrix containing $n \times 3$ control points. When $k < n < r$, equation (5) can be solved approximately using least squares as shown in equation (6):

$$D = (N^T N)^{-1} N^T P$$

The curve error is then obtained as:

$$E = \sum_{j=1}^{r} (D(t_j) - p_j)^2$$

$$E = D^T (I - NN^-) D$$

Equations (7) and (8) clearly demonstrate that the error magnitude depends directly on the parameter values t_j.

2.2 Problem Formulation and Basic Solution Approach

Pre-determining data parameters or node vectors is not optimal for curve fitting. Treating these two quantities as variables is the key to achieving the best fitting results. Considering their interrelationship, the node vector is dynamically set according to the least squares model to alter the size and approximation effect of the B-spline curve. Since the established nodes continuously change, $N^T N$ in equation (6) cannot be guaranteed to remain positive definite after genetic algorithm iterations. Moreover, least squares yields extreme values but cannot ensure they are global minima, and derivative-based function methods cannot

guarantee that every function has a derivative. Therefore, the concept of loss functions [**?**] is adopted to improve the traditional least squares solution.

Based on the parametric function of B-spline curves, if both the control vertices $\{P_i\}, i = 0, 1, \cdots, n$ and the node vector $\{u_j\}, j = 0, 1, \cdots, n+p+1$ are unknown variables, the B-spline curve approximation problem can be transformed into a nonlinear minimization optimization problem [**?**] as shown in equations (9) and (10):

$$Q = \min \left(\sum_{i=0}^{n} \|d_i - p(t_i)\|^2 \right)$$

$$Q = \min \left(\sum_{i=0}^{n} \left\| d_i - \sum_{j=0}^{n} N_{j,p}(t_i) P_j \right\|^2 \right)$$

Solving this minimization problem using traditional methods is difficult. Therefore, we improve upon conventional approaches by combining genetic algorithms with penalty functions to convert constrained optimization problems into unconstrained ones. In genetic algorithm optimization, an appropriate fitness function is selected, three basic operators are modified, and a simulated annealing operator is added to obtain the node vector. This vector is then input into the least squares model to derive optimal control points, ultimately generating the optimal curve fitting.

The flowchart of the curve fitting algorithm is shown in Figure 1 [Figure 1: see original paper].

3 Improved B-spline Curve Fitting Algorithm

To address insufficient handling of smooth, cusp, and discontinuous problems in curve fitting, we improve the genetic algorithm operators and propose an enhanced fitting algorithm. The algorithmic steps are as follows:

(a) Input the data points d_i to be fitted;

(b) Perform parameterization of data points and establish the corresponding fitness function;

(c) Initialize genetic algorithm parameters, including population size (popsize), crossover probability p_c, mutation probability p_m, and maximum iteration count;

(d) Initialize population $\delta(t)$ using Gray code encoding;

(e) Determine curve control points via least squares and calculate the fitness function value for each individual;

(f) Perform iterative calculations and finally output the optimal node vector and control points to reconstruct the curve; otherwise proceed to (g);

(g) Execute selection operation on $\delta(t)$ using the selection operator described above to obtain $\delta_1(t)$;

(h) Execute crossover operation on $\delta_1(t)$ using the crossover operator described above to obtain $\delta_2(t)$;

(i) Execute mutation operation on $\delta_2(t)$ using the mutation operator described above to obtain $\delta_3(t)$, then return to (e).

Based on these algorithmic steps, we focus on improving the least squares method and genetic operators to obtain the optimal node vector. The detailed process is described below.

3.1 Improvement of Traditional Least Squares Solution

In the least squares method, the final error expression can be understood as a loss function requiring representation using gradient vectors and Jacobian matrices. The Jacobian matrix is defined with elements as partial derivatives of errors with respect to parameters, as shown in equation (11):

$$J_{i,j}f(w) = \frac{\partial e_i}{\partial w_j} \quad (i = 1, 2, \cdots, m \text{ and } j = 1, 2, \cdots, n)$$

where m represents the number of data points and n represents the number of neural network parameters, making the Jacobian matrix an $m \times n$ matrix. The gradient vector of the loss function is given by equation (12):

$$\nabla f = 2J^T \cdot e$$

where e denotes the vector of all error values. The weight function update and optimization are performed as in equation (13):

$$w_{i+1} = w_i - (J_i^T \cdot J_i + \varepsilon_i I)^{-1} \cdot (2J_i^T \cdot e_i)$$

where ε represents a decay factor to ensure matrix positive definiteness.

3.2 Data Point Parameterization Settings

Based on least squares fitting, two factors influence curve shape: (a) the distribution of data point parameterization; and (b) the selection of the node vector. Different data point parameters and node vectors critically affect curve shape, and unreasonable selections lead to poor approximation.

Numerous methods exist for determining data point parameterization, with common approaches including uniform parameterization, cumulative chord length parameterization, and centripetal chord length parameterization. Compared to other methods, centripetal chord length parameterization [**?**] effectively reflects the distribution characteristics of data points and provides accurate fitting results when data undergoes sudden changes. The parameter t_i determines the positional relationship between data points, as given by equation (14):

$$t_0 = 0$$

$$t_{i+1} = t_i + \frac{\sqrt{\|d_{i+1} - d_i\|}}{\sum_{j=0}^{m-1} \sqrt{\|d_{j+1} - d_j\|}}$$

3.3 Node Vector Settings

Multiple methods have been proposed for node vector setting, including uniform node vectors, average node vectors, and Piegl' s approximation node vectors. The first method completely ignores data point distribution and yields poor results. The second method, while considering data points, employs overly simple averaging that accounts for too few cases. Piegl' s approximation method works for small-scale problems but becomes computationally inefficient when data volume increases.

To address these limitations, this paper proposes a new selection approach with the following steps:

a) Interpolate control points at endpoints. For a non-decreasing node vector $u_0 \leq u_1 \leq \cdots \leq u_{n+p+1}$, set the endpoint knot multiplicity to the B-spline degree $p+1$: $u_0 = u_1 = \cdots = u_p = 0$, $u_{n+1} = u_{n+2} = \cdots = u_{n+p+1} = 1$.

b) Use the first and last data points as the first and last control points of the reconstructed B-spline curve: $d_0 = P_0$, $d_m = P_n$.

c) The remaining $n + p$ internal nodes (excluding endpoints) are unknown before fitting. Genetic algorithms iteratively perform adaptive setting until the optimal node vector is generated.

3.4 Genetic Algorithm Principles and Corresponding Improvements

Genetic algorithms [**?**], first proposed by Professor Holland in his book *Adaptation in Natural and Artificial Systems*, draw inspiration from nature' s "survival of the fittest" principle. As a global optimization adaptive probabilistic search algorithm, genetic algorithms rely on individual fitness function values to perform selection, crossover, and mutation operations during evolution, thereby finding optimal solutions to problems. The algorithm provides a general framework for solving complex system optimization problems and has important applications in biotechnology, chemical engineering, computer-aided design, and medical engineering.

3.4.2 Operator Improvements and Parameter Settings The simple genetic algorithm consists of five components: initial population selection, chromosome encoding, fitness function selection, genetic operation design, and algorithm control parameter setting. These are described in detail below.

a) Initial Population Selection

Common approaches randomly generate initial populations, but initial population setting significantly impacts both results and algorithm efficiency. Random generation leads to non-uniform individual distribution in the feasible solution space. Therefore, uniform design is adopted: first, the solution space is divided into Z intervals; then P chromosomes are selected using uniform arrays; finally, Q individuals with higher fitness values are chosen from $Z \times P$ as the initial population to ensure individuals are as dispersed as possible in the solution space.

b) Chromosome Encoding

Genetic algorithms commonly use binary or decimal encoding for chromosomes. While these offer high stability and large population diversity for numerical optimization, their randomness results in poor local search capability. For high-precision problems, large phenotypic changes after mutation can push solutions away from the optimum, preventing stability. Therefore, this paper adopts Gray code encoding, where encoded values for consecutive integers differ by only one bit.

The conversion formula from binary to Gray code is:

$$g_m = b_m$$

$$g_i = b_{i+1} \oplus b_i \quad (i = m-1, m-2, \cdots, 1)$$

c) Fitness Function Selection

The fitness function forms the foundation for other key steps in genetic algorithms, and selecting an appropriate fitness function is crucial for optimization.

Based on the least squares model proposed earlier, achieving excellent curve fitting results requires controlling error values while reducing the number of control points. The fitness function can thus be designed as in equation (17):

$$fitness = Q \cdot \varphi + \mu \cdot \frac{\max\{G_i\} + 1}{\max\{H_i\} + 1}$$

where φ represents a control factor and μ represents a correction factor. Since genetic algorithm iterations cannot guarantee node ordering (and unordered nodes cannot generate B-spline curves), a functional body is designed as follows: (a) sort all individuals by size; (b) apply the least squares method to the data points to solve for control points; (c) determine individual fitness values based on the fitness function. This functional approach ensures node vector ordering by placing each iteration' s results in sequence.

d) Genetic Operator Design

Selection Operator: This primarily embodies the survival-of-the-fittest principle. Individuals with larger fitness function values are retained while those with smaller values are discarded. Although roulette wheel selection is widely used in simple genetic algorithms, its excessive randomness can be problematic. Therefore, a simulated annealing selection operator based on competition index [**?**] is employed. The competition index J comprehensively considers individual fitness values (denoted as F) and coding difference values (denoted as C), as shown in equation (18):

$$J = m_1 F + m_2 C$$

where m_1 and m_2 are proportion parameters summing to 1. Individuals are sorted by competition index magnitude, with selection order consistent with positional order to reasonably maintain population quality.

Crossover Operator: Effective crossover operations enable information exchange of superior genes between individuals. During crossover, individuals with fitness values above the average are defined as superior and should be crossed with higher probability to preserve excellent genes. However, relying solely on average fitness values risks local optimization. Therefore, variance and entropy are incorporated into probability design.

Assuming population size N and individual fitness values f_i with average $\bar{f}$, variance is expressed as in equation (19):

$$G_i = \sum_{i=1}^{N} (f_i - \bar{f})^2$$

Variance represents individual distribution within the population. When variance is large (individuals are dispersed), crossover probability decreases; otherwise, it increases.

Entropy divides feasible solutions into A_i ranges, with population individuals falling in A_i numbered $|A_i|$. Entropy is given by equation (20):

$$H_i = -\sum_{i=1}^{N} p_i \log p_i$$

where $p_i = \frac{|A_i|}{N}$. Larger entropy values indicate greater population diversity. If the population has strong evolution capability, crossover probability should be set smaller; otherwise, it should be larger. The final crossover probability is given by equation (21):

$$p_c = e^{\frac{1}{\max\{G_i\}+1} \cdot \frac{1}{\max\{H_i\}+1}}$$

Since initial values are in increasing order and considering the encoding method, arithmetic crossover is adopted. For two individuals x_1 and x_2, the crossover formulas are:

$$x_1' = \omega_1 x_1 + \omega_2 x_2$$

$$x_2' = \omega_1 x_2 + \omega_2 x_1$$

where ω represents a random number between (0,1), and $\omega_1 + \omega_2 = 1$.

Mutation Operator: Mutation probability is a critical factor directly affecting genetic algorithm optimization results. The probability value should not severely disrupt superior genes and preferably can generate excellent individuals. During mutation, individuals with fitness values below average are defined as inferior and require higher mutation probability. Based on the above approach, a premature convergence judgment criterion is added, considering the difference θ between the current individual' s maximum fitness value and the average fitness value of individuals exceeding the average fitness. The mutation probability is defined as in equation (24):

$$p_m = \frac{1}{1 + e^{-(k \cdot \theta)}}$$

where k is a control factor greater than 0, and mutation probability changes synchronously with θ. When mutation conditions are met, a random number s is generated for each gene j_i. If this number is smaller than the mutation probability, single-point mutation is executed as in equation (25):

$$j_i = \begin{cases} u \cdot j_2 & \text{if } j_i = j_1 \\ u \cdot j_{i-1} + u \cdot (j_i - j_{i-1}) & \text{if } j_i = j_{i-1} \\ (j_i + u) & \text{otherwise} \end{cases}$$

where u represents a random number in (0,1), adhering to principles of optimal selection and worst elimination.

If execution continues as described above, although results converge quickly toward the optimum, individual similarity increases while population diversity gradually decreases, making local convergence likely. To solve this problem, combined with the premature convergence indicator above, similar individuals are filtered when the indicator threshold is reached. The specific approach is:

a) Sort population individuals by fitness value;

b) Calculate similarity based on encoding method and determine a β value;

c) Compare similarity results with β and delete similar individuals.

After filtering, new individuals must be added. Direct random generation is unsuitable as it hinders global search. Instead, individuals with relatively high fitness from the original population can be mutated to increase superior individuals that are more likely to produce excellent offspring with other original individuals.

4 Experimental Results and Analysis

To evaluate the practical effectiveness of the proposed genetic algorithm-based B-spline curve fitting algorithm, test functions with smooth, discontinuous, and cusp characteristics were selected to determine appropriate nodes and obtain fitting results. Performance was compared against traditional simple genetic algorithms and fitting methods optimized with ant colony algorithms.

The experimental hardware environment consisted of an Intel(R) Core(TM) 2 Duo CPU with 4 GB RAM, running Windows 7, with MATLAB as the programming and experimental platform.

4.2 Test Function Selection

To test the multi-adaptability of curve fitting algorithms, three functions from literature [**?**] were adopted as test functions–commonly used functions for B-spline curve fitting. Their expressions and domains are shown in Table 1 .

Table 1: Test Functions

Function	Expression	Domain
$f_1(x)$	$1 + e^{-100(x-0.4)^2}$	$x \in [0, 1]$
$f_2(x)$	$\begin{cases} 0.01 + (x - 0.3)^2 & x \in [0, 0.6) \\ 0.015 + (x - 0.65)^2 & x \in [0.6, 1] \end{cases}$	$x \in [0, 1]$
$f_3(x)$	$e^{\lvert 10x-5 \rvert} + (10x - 5)^5$	$x \in [0, 1]$

4.3.1 Selection of Optimal Node Number

By analyzing the evolution trend of node numbers with increasing iteration counts for test functions $f_1(x)$, $f_2(x)$, and $f_3(x)$, optimal node numbers can be obtained to reconstruct function curve fitting diagrams. The functional relationships between iteration count and node number for the three test functions are shown in Figures 2 [Figure 2: see original paper] through 4 [Figure 4: see original paper].

Analysis of Figures 2–4 reveals that the optimal node number is 4 for $f_1(x)$, 8 for $f_2(x)$, and 5 for $f_3(x)$.

4.3.2 Test Function Fitting Effects

Using the obtained optimal nodes, curve fitting was performed. The fitting effects for the three test functions are shown in Figures 5 [Figure 5: see original paper] through 7 [Figure 7: see original paper].

Figures 5–7 illustrate the fitting results using the proposed method. Figure 5 shows the fitting effect for a curve with smooth regions, demonstrating good approximation in smooth areas with acceptable deviations. Figure 6 presents the fitting of a discontinuous curve, showing excellent handling at discontinuity points that closely approaches the original curve trend. Figure 7 depicts a curve with cusp points, clearly exhibiting extreme values and essentially preserving original characteristics. These results demonstrate that the improved genetic algorithm-based B-spline curve fitting algorithm handles smooth, discontinuous, and cusp-bearing cases effectively, with fitting effects nearly identical to actual curve trends, indicating excellent algorithm performance.

4.3.3 Comparison with Other Fitting Algorithms

To further evaluate the proposed algorithm' s rationality, three algorithms were compared using equation (8) to calculate results. Table 2 presents fitting errors for the proposed algorithm versus methods from literature [**?**] and literature [**?**].

Table 2: Error Comparison of Test Functions

Method	$f_1(x)$ Data Points	$f_2(x)$ Data Points	$f_3(x)$ Data Points
Proposed Algorithm	1.075×10^{-6}	4.062×10^{-6}	1.773×10^{-2}
Literature [?] Method	5.967×10^{0}	9.327×10^{-1}	7.762×10^{1}
Literature [?] Method	2.563×10^{-4}	3.743×10^{-4}	5.532×10^{-2}

Table 2 shows that for all three test functions, the proposed algorithm achieves relatively small errors and superior approximation effects.

5 Conclusion

This paper improves key genetic algorithm operators and combines them with a least squares model. Experimental results demonstrate that the improved genetic algorithm achieves better curve approximation with higher precision. However, achieving higher numerical precision with genetic algorithms requires improved convergence speed, and data parameterization needs enhancement. Additionally, the substantial randomness of genetic algorithms necessitates increased algorithm stability—an issue worthy of continued research. Future work will extend the current two-dimensional curve research to three-dimensional surfaces to keep pace with technological development.

References

[1] Zhou Minghua, Wang Guozhao. Genetic algorithm-based least square fitting of B-spline and Bézier curves [J]. Journal of Computer Research & Development, 2005, 42 (1): 134-143.

[2] Lin Ao, Xiao Bing, Zhu Yi. An algorithm for Bayesian network structure learning based on simulated annealing with adaptive selection operator [J]. Lecture Notes in Electrical Engineering, 2014, 277: 525-532.

[3] Hu Liangchen, Shou Huahao. Optimization of B spline reconstruction curve nodes based on binary GA [J]. Journal of software, 2016, 27 (10): 2488-2498.

[4] Yoshimoto F, Harada T, Yoshimoto Y. Data fitting with a spline using a real-coded genetic algorithm [J]. Computer-Aided Design, 2003, 35 (8): 751-760.

[5] Zhang Jumei, Wang Honglun. B-spline curve fitting based on genetic algorithms and the simulated annealing algorithm [J]. Computer Engineering & Science, 2011, 33 (3): 191-193.

[6] Piegl LA, Tiller W. Least-squares B-spline curve approximation with arbitary end derivatives [J]. Engineering with Computers, 2000, 16 (2): 109-116.

[7] Xiao Yijun, Ding Mingyue, Peng Jiaxiong. ICP-based B-spline curve fitting [J]. Journal of Image & Graphics, 2000, 5 (7): 585-588.

[8] Li Jiyun, Geng Zhaofeng. A study of least point B-spline curve fitting algorithm with given error based on genetic algorithm [J]. Journal of Computer Aided Design & Computer Graphics, 2003, 15 (3): 334-337.

[9] Duan Zhenyun, Wang Ning, Yang Xu, et al. Research on an improved B-spline curve fitting algorithm [J]. Mechanical Design & Manufacture, 2016 (5): 17-19.

[10] Sun Yuehong, Wei Jianxiang, Xia Deishen. Parameter optimization for B-spline curve fitting based on adaptive genetic algorithm [J]. Journal of Computer Applications, 2010, 30 (7): 1878-1882.

[11] Xu Shanjian, Guo Youqiang, Qi Xiaoming, et al. Chaotic ant swarm optimization in solving curve fitting with free knot B-splines [J]. Computer Engineering & Applications, 2014, 50 (16): 177-182.

[12] Gálvez A, Iglesias A. Firefly algorithm for explicit B-spline curve fitting to data points [J]. Mathematical Problems in Engineering, 2013, 2013: 1-9.

[13] Picek S, Golub M, Jakobovic D. Evaluation of crossover operator performance in genetic algorithms with binary representation [J]. Lecture Notes in Computer Science, 2011, 6840: 223-230.

[14] Jin Wei, Liu Zhijie, Jing Fengxuan. B-spline curve interpolation based on least squares of approximation [J]. Journal of Guizhou Normal University, 2015, 33 (1): 98-102.

[15] Tsuchiya T. Global optimization of polynomial-expressed nonlinear optimal control problems with semidefinite programming relaxation [J]. Journal of Global Optimization, 2012, 54 (4): 831-854.

[16] Huang Weixian, Wang Guojin, Jin Congjian. Complex rational curves with chord length parameterization [J]. Journal of Computer-Aided Design & Computer Graphics, 2011, 23 (12): 1975-1980.

[17] Deng Chongyang, Lin Hongwei. Progressive and iterative approximation for least squares B-spline curve and surface fitting [J]. Computer-Aided Design, 2014, 47 (1): 32-44.

[18] Chen Hao, Cui Duwu, Yan Taishan, et al. Simulated annealing ranking selection operator based on competition index [J]. Electronic Journal, 2009, 37 (3): 586-590.

Note: Figure translations are in progress. See original paper for figures.