ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/323987673

Learnt Knot Placement in B-Spline Curve Approximation using Support Vector Machines

Article *in* Computer Aided Geometric Design · March 2018
DOI: 10.1016/j.cagd.2018.03.019

CITATIONS
0

READS
113

3 authors:

Pascal Laube
Hochschule Konstanz Technik, Wirtschaft und Gestaltung Konstanz
10 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Matthias O. Franz
Hochschule Konstanz Technik, Wirtschaft und Gestaltung Konstanz
92 PUBLICATIONS **2,489** CITATIONS
SEE PROFILE

Georg Umlauf
Hochschule Konstanz Technik, Wirtschaft und Gestaltung Konstanz
65 PUBLICATIONS **451** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Entwicklungen einer innovativen Anlagentechnik zur automatisierten und laserbasierten Reparatur strukturierter Formeinsätze – ToolRep
View project

Multifunktional-skalierbare generische InlineInspektion für flexible Fertigungsprozesse in vernetzten Produktionsanlagen (MultiFlexInspect)
View project

All content following this page was uploaded by Pascal Laube on 27 March 2018.

The user has requested enhancement of the downloaded file.

Learnt Knot Placement in B-Spline Curve Approximation using Support Vector Machines

Pascal Laube [1], Matthias O. Franz [1], Georg Umlauf [1]

[1] Institute for Optical Systems, University of Applied Sciences Konstanz, Germany

Abstract

Knot placement for curve approximation is a well known and yet open problem in geometric modeling. Selecting knot values that yield good approximations is a challenging task, based largely on heuristics and user experience. More advanced approaches range from parametric averaging to genetic algorithms.

In this paper, we propose to use Support Vector Machines (SVMs) to determine suitable knot vectors for B-spline curve approximation. The SVMs are trained to identify locations in a sequential point cloud where knot placement will improve the approximation error. After the training phase, the SVM can assign, to each point set location, a so-called score. This score is based on geometric and differential geometric features of points. It measures the quality of each location to be used as knots in the subsequent approximation. From these scores, the final knot vector can be constructed exploring the topography of the score-vector without the need for iteration or optimization in the approximation process. Knot vectors computed with our approach outperform state of the art methods and yield tighter approximations.

1 Introduction

Finding a good knot vector for curve approximation is a common problem in many applications such as geometric modeling, data fitting or reverse engineering. The reverse engineering process usually starts with a sampled physical object represented by a set of points. For this point set, a corresponding CAD representation needs to be reconstructed. After pre-processing and segmentation steps, a B-spline approximation of the point set, or parts of it, is computed. For this approximation, the data points, parameter values for the data points, the B-spline degree, and an appropriate knot vector are required. The goal is to find knot vectors consisting of as few knots as possible while minimizing the deviation of the approximating B-spline curve to the original dataset. Usually, an error bound has to be satisfied to guaranty a sufficient approximation quality. Since the number of control points is determined by the number of knots (and the degree), the number of knots is particularly important if the resulting curves will be further processed manually.

Because there are many unknowns in the approximation process, we propose to compute the knot vector using machine learning methods. We train support vector machines (SVMs) to classify locations along a set of points for their fitness as knots in the parameter domain. This fitness is measured in terms of the approximation error. For training and classification, we use a feature vector that concentrates information about a point cloud's geometric structure. After training, the SVMs assign a score to each point of a point set based on its local point neighborhood. This score assesses each point location's impact on the approximation error if used as an additional knot. The topography of this score along the point set yields the final knot vector. Since optimization is only done during the training of the SVM, our knot placement approach does not involve an optimization or iteration process during the actual approximation task. Compared to state of the art methods for knot vector computation our method generates tighter approximations.

The sections of the paper are arranged as follows. Section 2 gives an overview of the related works. The required preliminaries for the score-based knot placement are given in Section 3. In Section 4 we present our score-based knot placement approach. Results are discussed in Section 5.

2 Related works

Knot placement for curve approximation to a given error bound typically is an iterative process. This process starts with an empty or full knot vector and then iteratively adds or removes knots until a certain threshold is satisfied. Examples for knot removal are the methods by Lyche [17, 18].

Adding suitable knots is either done by instant computation of new knot positions or by using optimization methods. A method for instant knot vector computation that uses shape information to determine suitable knot values is [15]. Here Li et al. use a heuristic rule that is based on an angular measure for knot placement. They introduce new knots so that the curvature integration of resulting curve segments is balanced. Razdan [23] uses a slightly different approach by first selecting nodes and then interpolating these nodes by B-spline curves. Razdan also extensively analyzed the impact of different geometric features for knot vector computation. Another method for instant computation of knot vectors that does not consider geometric features is the method of Piegl and Tiller [20]. Here parameter values are averaged to generate the knot vectors which lead to a stable system of equations and an even distribution of knots along the curve.

A well-known optimization based method was introduced by Park and Lee [19]. Their method for knot vector computation relies upon dominant points. These dominant points are points of special interest in the point set for approximation. Knot positions are then optimized to balance the inter-segment shape index distance. Techniques for iterative optimization of an initial knot vector also include methods from non-linear optimization [37, 25, 9]. In [12] Kang et al. treat knot placement as a convex optimization problem, which results in smaller computational complexity. In [36] Yuan et al. select knots extracting optimal subsets from a multi-resolution B-spline basis using the Lasso method. There exists also a solid body of work using genetic algorithms for knot vector optimization [35, 24, 33, 32, 31], meta-heuristics like a firefly algorithm driven approach [7], or elitist clonal selections [8].

The topic of machine learning has received a lot of attention over the last decade. There exist some insightful publications for learning methods in the context of reverse engineering and applications to problems in geometric modeling. Lin et al. [16] propose neural networks for surface reconstruction by 2d images. The outputs of the neural network are the surface normals which can then be used for illumination tasks. Surface approximation can also be treated as a regression problem. Steinke et al. [28]

use support vector regression for head reconstruction, hole filling, and outlier removal. They then adept the concept for surface deformation. Another aspect of reverse engineering is the classification of surface patches by geometric primitives. Arbeiter et al. [1] used SVMs with point features as well as curvature features to classify surfaces for a small set of geometric primitives including edges and corners. A similar concept for classification of different surface primitives, e.g. spheres or cylinders, by training support vector machines using different feature histograms, was proposed in [2].

Our method is related to methods [15, 23, 20], since the knot vector is computed instantly based on geometric features of a point set. Our method's advantage is that it is able to instantly recommend new knot positions that yield a near-optimal improvement of the approximation error, leading to tight approximations with fewer knots. To our knowledge, this is the first method introducing machine learning for knot vector computation.

3 Preliminaries

In this section, we give a short introduction to B-spline curve approximation and SVMs.

3.1 B-spline curve approximation

Suppose we have given a sequentially arranged set of points $\mathbf{p} = (p_0, ..., p_m)$ as in Figures 6a and 7a. Consider a B-spline curve

$$C(u) = \sum_{j=0}^{n} c_j \, N_j^k(u)$$

of degree k with control points c_j, B-spline functions $N_i^k(u)$, and a non-decreasing knot vector $\mathbf{u} = (u_0, \ldots, u_n)$ where the knots u_0 and u_n have multiplicity $k+1$ for end point interpolation. To compute the control points c_j of the B-spline curve C approximating $\mathbf{p}$, the least squares problem

$$\sum_{i=0}^{m} |p_i - C(t_i)|^2 \to \min$$

with precomputed parameters t_i and end points $C(t_0) = c_0 = p_0$ and $C(t_m) = c_n = p_m$ is solved. This yields the normal equation

$$(\mathbf{N}^T\mathbf{N})\mathbf{c} = \mathbf{q} \tag{1}$$

where $\mathbf{N}$ is the $(m-1) \times (n-1)$ matrix

$$\mathbf{N} = \begin{pmatrix} N_1^k(t_1) & \ldots & N_{n-1}^k(t_1) \\ \vdots & \ddots & \vdots \\ N_1^k(t_{m-1}) & \ldots & N_{n-1}^k(t_{m-1}) \end{pmatrix},$$

and $\mathbf{c}$ and $\mathbf{q}$ are the vectors defined as

$$\mathbf{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}, \mathbf{q} = \begin{pmatrix} \sum_{i=1}^{m-1} N_1^k(t_i) q_i \\ \vdots \\ \sum_{i=1}^{m-1} N_{n-1}^k(t_i) q_i \end{pmatrix}$$

and

$$q_i = p_i - N_0^k(t_i)p_0 - N_n^k(t_i)p_m$$

for $i = 1, ..., m-1$. The parameters t_i, $i = 0, \ldots, m$, can be precomputed using for example the centripetal parametrization, see e.g. [14]. They are combined in the *parameter vector* $\mathbf{t} = (t_0, \ldots, t_m)$. The control points c_j can be computed using (1), if

$$\sum_{l=1}^{m-1} N_i^k(t_l) N_j^k(t_l) \neq 0. \tag{2}$$

This is equivalent to the existence of a parameter $t_i \in [u_j, u_{j+1}]$ for $j = k, ..., n+1$, see e.g. [5]. For our experiments, we used $k = 3$.

3.2 Support vector machines

Supervised learning methods are designed to classify new data with respect to a large body of pre-classified training data. The training data are represented as feature vectors which are chosen to separate the different classes from each other. These features live in a high-dimensional feature space F. In supervised learning, the objective is to generate separating hyperplanes or more general hypersurfaces between the classes in feature space. These separating hypersurfaces are then used to classify new input data by deciding on which side of the separating hypersurfaces the data reside. In the case of SVMs, the additional objective is to maximize the margin between two classes in relation to their separating hypersurface. Figure 1 illustrates this concept in case of separating hyperplanes in a two-dimensional feature space. For details on SVMs see [4].

Given a set of n training examples x_i with class labels $y_i \in \{-1, 1\}$ maximizing the *margin* between two classes results in finding the normal vector ω and a bias value b such that

$$\Phi(\omega) = \frac{1}{2}\omega^t\omega \tag{3}$$

is minimized with respect to the constraints

$$y_i(\omega^t\mathbf{x}_i + b) \geq 1, i = 1, \ldots, n. \tag{4}$$

For noisy data requiring linear separability might be too restrictive. Thus, *slack variables* ξ_i are used to allow for some data inside or beyond the margin, see Fig. 1. The objective function to minimize now additionally penalizes excessive slack variables

$$\Phi(\omega) = \frac{1}{2}\omega^t\omega + C\sum_{i=1}^{n}\xi_i \tag{5}$$

with respect to the constraints

$$y_i(\omega^t\mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, \ldots, n. \tag{6}$$

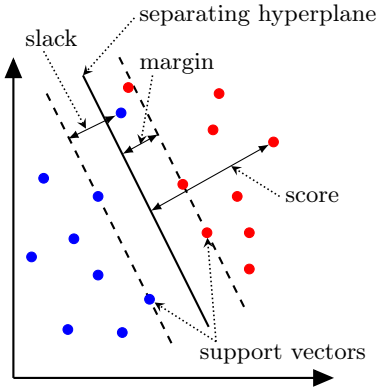

Figure 1: Separating hyperplane, margin, score, support vectors, and slack for a two dimensional feature space.

In (5) C is called the soft margin parameter and ξ the kernel parameter. The minimum of (5) subject to (6) is usually computed via its dual form using Lagrange multipliers [10]. Then, the Karush-Kuhn-Tucker-conditions imply that the weight vector ω can be represented in terms of the Lagrange multipliers $\alpha = (\alpha_1, \ldots, \alpha_n)$ and the training data as $\omega = \sum_{i=1}^{n} \alpha_i \, y_i \, \mathbf{x}_i$. So, instead of minimizing (5) the resulting Lagrangian

$$\Phi^*(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \, y_i y_j \, \mathbf{x}_i^t \mathbf{x}_j. \tag{7}$$

is maximized with respect to the constraints

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \qquad 0 \le \alpha_i \le C, \qquad i = 1, \ldots, n.$$

If in the solution $\alpha_i \neq 0$, the corresponding $\mathbf{x}_i$ is called *support vector*.

For some problems, the classes cannot be separated by hyperplanes, but only by non-linear hypersurfaces. For this purpose, the scalar product $\mathbf{x}_i^t \mathbf{x}_j$ is replaced by a so-called kernel $K(\mathbf{x}_i, \mathbf{x}_j)$. The kernel has to be chosen such that it corresponds to a scalar product in a high-dimensional feature space. We use the Gaussian RBF kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2).$$

With this kernel the classifier for new data $\mathbf{x} \in F$ is given by $\operatorname{sign}(m_{\alpha,b}(\mathbf{x}))$ with the *data margin*

$$m_{\alpha,b}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b. \tag{8}$$

The *score* for predicting $\mathbf{x}$ in the positive class is $m_{\alpha,b}(\mathbf{x})$, while the score for prediction it in the negative class is $-m_{\alpha,b}(\mathbf{x})$.

Finding parameters C and γ is referred to as model selection. This optimization is usually based on a non-uniform grid search in the two-dimensional (C, γ)-space. Since model selection requires training data, its description is deferred until Section 4.3. Thus, a trained SVM is completely determined by the SVM parameters (ω, b, C, γ).

4 SVM based knot placement for curve approximation

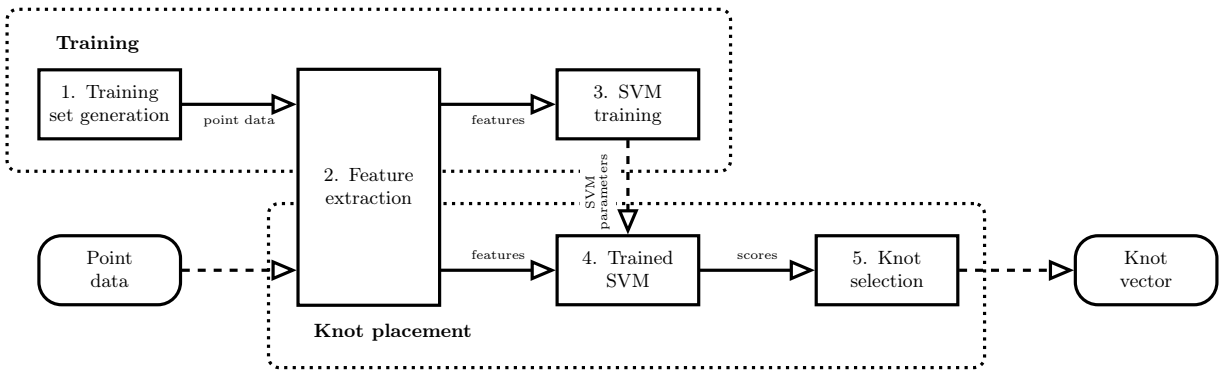

Figure 2: Overview of knot vector generation process.

The generation of a knot vector for B-spline curve approximation using SVMs has two major stages. In the first stage, the SVM is trained. In the second stage, the trained SVM is used to generate knot vectors for every new approximation task. An overview of this process is shown in Figure 2. Thus, for B-spline curve approximation, the complete process consists of six building blocks, where blocks 1, 2, 3 make up the training stage while blocks 2, 4, 5 make up the knot placement stage.

1. *Generation of training sets* consisting of point sequences (Section 4.2).
2. Local *feature extraction* for point sequences (Section 4.1).
3. *SVM training* based on a pre-computed training set and model selection (Section 4.3).
4. Using the *trained SVM* on feature vectors of new point data for knot classification and score computation (Equation (8)).
5. Score-based *knot selection* (Section 4.4).
6. *Approximation* of point set by B-spline curve using (1) (not shown in Figure 2).

The sixth block in the process (i.e. approximation) is used only for the evaluation of the proposed training and knot placement stages. Its results are discussed in Section 5.

4.1 Geometric features

Machine learning methods need a compact representation of the data. Good representations contain the most discriminatory features of the data. The data for curve approximation is a point set $\mathbf{p} = \{p_0, \dots, p_m\}$. For each point p_i we use a 14-dimensional feature vector $\mathbf{x}_i := (f_1^i, \dots, f_{14}^i)$ based on distances, curvatures, and angles of point pairs. These features encode the geometry of the local point neighborhood. Thus, they are symmetric in the sense that flipping the order of points in $\mathbf{p}$ has no effect on the resulting feature vectors.

In the literature, curvature is one of the main indicators for knot placements [36, 23]. This yields three different curvature based features.

F.1 Curvature $f_1^i = \kappa_i$ at p_i. We compute $\kappa_i = \frac{1}{r_i}$ with r_i being the radius of the osculating circle at p_i estimated by [30].

F.2 & 3 Minimum and maximum of the approximate curvature derivatives to the left and right

$$f_2^i = \min\left(\frac{\nabla\kappa_i}{\nabla t_i}, \frac{\Delta\kappa_i}{\Delta t_i}\right), \qquad f_3^i = \max\left(\frac{\nabla\kappa_i}{\nabla t_i}, \frac{\Delta\kappa_i}{\Delta t_i}\right)$$

using the forward Δ and backward ∇ difference operators.

Due to their importance in knot-placement, we also use local curvature maximum (LCM) points to define LCM-based features. The set $\mathbf{p}^L = \{p_0^L, \dots, p_s^L\} \subseteq \mathbf{p}$ of LCM points contains $p_0^L = p_0$ and $p_s^L = p_m$. The other LCM points $p_j^L = p_\ell$ are defined as

$$\kappa_\ell > \max(\kappa_{\ell-1}, \kappa_{\ell+1}) \quad \text{and} \quad \kappa_\ell \geq \kappa_{LCM}$$

with

$$\kappa_{LCM} = \frac{1}{4(m+1)} \sum_{i=0}^{m} \kappa_i.$$

We further define an LCM segment as $\mathbf{p}^j = \{p_1^j, \dots, p_{r_j}^j\}, j = 0, \dots, s-1$ consisting of all r_j points p_i strictly between p_j^L and p_{j+1}^L, i.e.

$$\mathbf{p} = \mathbf{p}^L \dot\cup \mathbf{p}^0 \dot\cup \dots \dot\cup \mathbf{p}^{s-1},$$

where $\dot\cup$ denotes the disjoint union. Thus, there is a one-to-one correspondence of points p_j^L and p_i^j to indices $\{0, \dots, m\}$. We denote their parameters with $t(p_j^L)$ and $t(p_i^j)$, respectively. Using the parameter values $t(p_j^L)$ as knots yields B-spline curve segments with smaller and more uniform distribution of the shape index along curve segments, see [19]. This leads to a better approximation. Combining LCM points with curvature features gives two further features

F.4 Ratio of the approximate curvature integral over all points $\mathbf{p}$ and the approximate curvature integral over points $\mathbf{p}^j$ with $p_i \in \mathbf{p}^j$.

$$f_4^i = \frac{\sum_{\ell=0}^{r_j} \kappa_\ell}{\sum_{\ell=0}^{m} \kappa_\ell}.$$

F.5 Mean curvature of the points $\mathbf{p}^j \cup \{p_j^L, p_{j+1}^L\}$.

Li and Xu [15] define a heuristic based on the angle $\sphericalangle(p_{i-1}, p_i, p_{i+1})$ of three consecutive points. If this angle is smaller than $\pi/6$, the spline curve is locally of small deflection and can be interpolated by a cubic spline curve. Knot vectors generated with this heuristic yield good approximations. Based on this observation, we define the angular feature:

F.6 The angle $f_6^i = \sphericalangle(p_j^L, p_i^j, p_{j+1}^L)$.

Placing knots based on Euclidean or parametric distance proved also effective for curve approximation quality ([19, 23, 21]). Hence, a third group of features contains distance based features.

F.7 & 8 Minimum and maximum of the Euclidean distances of point p_i^j to p_j^L and p_{j+1}^L.

F.9 & 10 Minimum and maximum of the approximate arc-lengths between p_i^j and p_j^L as well as p_i^j and p_{j+1}^L as the sum of Euclidean distances between all intermediate points.

F.11 & 12 Minimum and maximum of the parametric distances $\Delta_{ij} = t(p_j^L) - t(p_i^j)$ and $\nabla_{ij} = t(p_i^j) - t(p_{j+1}^L)$ relative to the parametric distance of p_j^L to p_{j+1}^L as

$$f_{11}^i = \min\left(\frac{\nabla_{ij}}{\Delta t(p_j^L)}, \frac{\Delta_{ij}}{\Delta t(p_j^L)}, \right), \qquad f_{12}^i = \max\left(\frac{\nabla_{ij}}{\Delta t(p_j^L)}, \frac{\Delta_{ij}}{\Delta t(p_j^L)}, \right)$$

F.13 & 14 Minimum and maximum of the parametric distances from p_{i-1}^j to p_i^j and p_i^j to p_{i+1}^j relative to the parametric distance of p_j^L to p_{j+1}^L

$$f_{13}^i = \min\left(\frac{\nabla t(p_i^j)}{\Delta t(p_j^L)}, \frac{\Delta t(p_i^j)}{\Delta t(p_j^L)}\right), \qquad f_{14}^i = \max\left(\frac{\nabla t(p_i^j)}{\Delta t(p_j^L)}, \frac{\Delta t(p_i^j)}{\Delta t(p_j^L)}\right).$$

These features are computed for all non-LCM points. We normalize features to have zero-mean and unit-variance. This results in $m - s$ feature vectors $\mathbf{x}_i$ of 14 normalized features per point set $\mathbf{p}$.

4.2 Training set generation

For the approximation of sequentially arranged 2d points, a suitable knot vector is required. Since different knot vectors yield different deviations of the curve from the points, a good knot vector generates little deviation with a small number of knots.

Finding the optimal knot vector is infeasible because exhaustive search for the optimal knot vector is computationally too expensive due to its combinatorial complexity. However, using a machine learning approach to find an almost optimal knot vector, the training point sets have to be generated only once for the pre-processing stage.

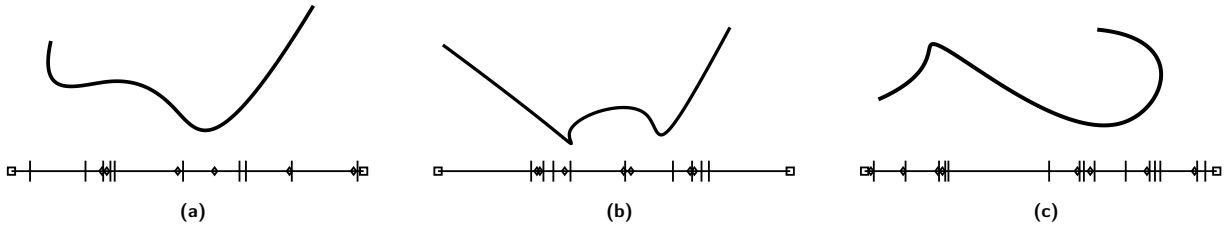

Figure 3: Three example curves from the training dataset with corresponding knot vectors. Diamonds are LCM knots and vertical lines are knots found by exhaustive search.

The training set generation process consists of eight steps:

T.1 Generate random control points c_i using the standard normal distribution with mean μ and variance σ to define uniform B-spline curves of degree k. We shift the mean by $\Delta\mu > 0$ for x-coordinates of consecutive control points. Curves with self-intersections are discarded.

T.2 Sample $m+1$ points $\mathbf{p} = (p_0, \ldots, p_m)$ uniformly along the curve and compute their centripetal parameters $\mathbf{t} = (t_0, \ldots, t_m)$ [14].

T.3 Add parameter values t_0 and t_m of p_0 and p_m to the initially empty knot vector $\mathbf{u}$.

T.4 Compute the LCM points of $\mathbf{p}$ and add the corresponding parameter values to the *initial knot vector* $\mathbf{u} = (u_0, ..., u_w)$.

T.5 Approximate $\mathbf{p}$ with a B-spline curve $\widetilde{C}$ of degree k and knot vector $\mathbf{u}$.

T.6 Insert further knots from $\mathbf{t}$ to $\mathbf{u}$ to minimize the Hausdorff distance between $\mathbf{p}$ and $\widetilde{C}$.

T.7 Repeat Steps T.5 to T.7 until a predefined improvement rate ε is achieved.

T.8 Raise the multiplicity of the end-knots to $k+1$.

Remark 1. *Self-intersecting curves are discarded because the sequential order of their sampled point sets is not unique. In reverse engineering, such point sets are usually split into subsets at the self-intersection.*

For our experiments we used the following parameters.

Ad T.1: We use $k = 3$. For the control point generation we use $\mu = 1$ and $\sigma = 0.5$ for y-coordinates. For x-coordinates we start with $\mu = 1$ and increase it by $\Delta\mu = 0.1$ for every consecutive control point using $\sigma = 0.1$.

Ad T.2: We use $m = 149$.

Ad T.7: We use $\varepsilon = 0.15$.

With this setup, we computed a training set of $1,000$ curves and a test set of 250 curves. This yields point sets consisting of $173,889$ points in total. For the training, only non-LCM points are used. Figure 3 shows three example curves from the training dataset.

Remark 2. *Exhaustive search for the optimal knot vector without the segmentation at LCM points would yield better training data, but is computationally infeasible. With the above parameters, we get an average of* 6 *LCM segments per curve, consisting of* 24 *data points. Exhaustive search yields an average of* 2 *knots per curve segment. This leads to an average of* 276 *possible knot placements within each segment. The average number of knots per curve (excluding* u_0 *and* u_n*) is* 17*. Overall this approach led to* $1,656,000$ *tested knot vectors for our training dataset. Searching for a set of* 17 *knots out of* 148 *possible locations without segmenting at the LCM points would yield approximately* 8.48e21 *possible knot placements, which renders the generation of a sufficiently large data set infeasible.*

In our experiments, the parameters T.1, T.2 and T.7 led to a very diverse set of curves containing sections with very little to no curvature as well as sections with high curvature and even sharp features. While we use cubic B-spline curves for dataset generation, our approach is not limited to point clouds computed this way. An ideal data set would consist of diverse real-world point clouds, with known sequential order of points. Since no sufficiently large datasets are publicly available, we chose to synthesize the data using B-spline curves.

4.3 SVM training

The training of an SVM requires the computation of (ω, b) and (C, γ). The parameters (ω, b) are determined from the training set solving the optimization problem (5) with respect to (6).

Computing (C, γ) is called model selection. For model selection it is best practice to sample the (C, γ)-space at exponentially growing values for C and γ [11]. Based on previous experiments with SVMs for geometric problems [13] we used the (C, γ)-set

$$T = \{0.01, 0.1, 1, 100, 1000, 10000, 50000\} \times \{0.01, 0.1, 0.25, 0.5, 1.0, 2, 5, 10\}.$$

For each grid point of T the (ω, b)-parameters of an SVM are computed via (5) and (6). For each of these SVMs, the point sets of the test set are approximated using the proposed knot placement method of Section 4.4 with varying numbers of knots $5, \ldots, 25$. The SVM which yields the minimal mean Hausdorff distance over all test curves and number of knots determines the (C, γ)-parameters; here $C = 0.01$ and $\gamma = 10$.

Algorithm 1 Score based knot placement (SKP) for curves.

Require:
$\mathbf{t} = (t_0, ..., t_m)$, parameter vector
$\mathbf{s} = (s_0, ..., s_m)$, score vector
$\mathbf{r} = (r_0, ..., r_m)$, prominence vector
k, b-spline degree
c, target number of knots
$\#(\,\cdot\,)$, number of entries in a vector

function SKP($\mathbf{t}$, $\mathbf{s}$, $\mathbf{r}$, k, c)
  initialize $\mathbf{u}$ with 0 and 1
  $\mathbf{v}$:=vector of indices of relative maxima in $\mathbf{s}$
  while $c > 0$ **do**
    if $\#(\mathbf{v}) > 0$ **then**
      $\mathbf{v}_{\max}$:=$\operatorname{argmax}_{i \in \mathbf{v}}(\mathbf{r})$
      if $\#(\mathbf{v}_{\max}) > 1$ **then**
        $\ell := \operatorname{argmax}_{i=0,\dots,\#(u)-2} (u_{i+1} - u_i)$
        $\bar{u}$:=$(u_\ell + u_{\ell+1})/2$
        j :=$\operatorname{argmin}_{i \in \mathbf{v}_{\max}}(|t_i - \bar{u}|)$
      else
        j :=$(\mathbf{v}_{\max})_0$
      $\bar{t}$:=t_j
      $\mathbf{v}$:=$\mathbf{v} \setminus (j)$
    else
      $\ell := \operatorname{argmax}_{i=0,\dots,\#(u)-2} \Big(\sum_{t_j \in [u_i, u_{i+1}]} s_j \Big)$
      $\bar{u}$:=$(u_\ell + u_{\ell+1})/2$
      $\bar{t}$:=$\operatorname{argmin}_{t \in \mathbf{t}}(|t - \bar{u}|)$
    $\mathbf{u}$:=$\mathbf{u} \cup (\bar{t})$
    $\mathbf{t}$:=$\mathbf{t} \setminus (\bar{t})$
    c :=$c - 1$
  raise multiplicity of end-knots of $\mathbf{u}$ to $k + 1$
  return u

4.4 Score based knot placement (SKP) for curves

To place the knots for the approximation of a new point set $\mathbf{p}$ based on score based knot placement (SKP), first the feature vectors $\mathbf{x}_i$ for all points $p_i \in \mathbf{p}$ are computed. If $p_i \in \mathbf{p} \setminus \mathbf{p}^L$, the features $\mathbf{x}_i$ are fed into the trained SVM to compute the score s_i by (8). This score is normalized to $[0, 1]$. If $p_i \in \mathbf{p}^L$, the score s_i is set to one, $s_i = 1$. Note that this implies $s_0 = s_m = 1$ and yields a score vector $\mathbf{s} = (s_0, \dots, s_m)$.

For a point p_i whose feature $\mathbf{x}_i$ yields a classification as knot, the score measures the certainty of the SVM for t_i to be a knot. For a point p_i whose feature $\mathbf{x}_i$ does not yield a classification as knot, the score measures how close t_i is to being a knot. Thus, using the score, positions along $\mathbf{p}$ can be identified, where the respective parameters have the highest SVM scores. Presumably, these are the positions where insertion of a knot is most beneficial for the approximation. Figure 4 shows example curves colored by score.

The identification of score maxima s_i for $i = 1, \dots, m-1$ is based on two topographic measures. The first is the *relative maximality* of a score s_i, i.e. $s_i > s_{i-1}, s_{i+1}$. The second is the so-called *prominence*, which measures the height of a local score maximum s_i relative to the surrounding local maxima. Denote by s_j and s_k the closest scores to the left and right of s_i larger than s_i. This means, $s_j, s_k > s_i$ for $j < i < k$ and all scores in the sub-vector $(s_{j+1}, \dots, s_{k-1})$ are at most s_i. Then, the prominence r_i of s_i is defined as

$$r_i = s_i - \max\big(\min(s_j, \dots, s_i), \min(s_i, \dots, s_k)\big).$$

For the score based knot placement, first the relative maxima in $\mathbf{s}$ are identified. Out of these, the scores with maximal prominence are selected. If there exists more than one score with maximal prominence, the one closest to the midpoint of the largest knot interval is selected. Then, the parameter t_j corresponding to the selected score is inserted to $\mathbf{u}$. If the list of relative score maxima is empty, further knots are inserted in the middle of the knot interval with the largest sum of scores. This sum of scores for a knot interval $[u_i, u_{i+1}]$ is computed as the sum of scores corresponding to parameters $t_j \in [u_i, u_{i+1}]$. Finally, the parameter t_j closest to the midpoint of the parameter interval with the largest sum of scores is inserted to $\mathbf{u}$. This process is repeated until the target number of knots is reached. The idea for this strategy is adapted from orography: if the score is interpreted as height in a mountainous region, then firstly pick those peaks with the largest prominence and secondly subdivide regions according to their height.

The steps for the score based knot placement are detailed in Algorithm 1. Since classification is computationally inexpensive and has to be done only once at the start of this procedure, it has no major impact on the overall execution time.

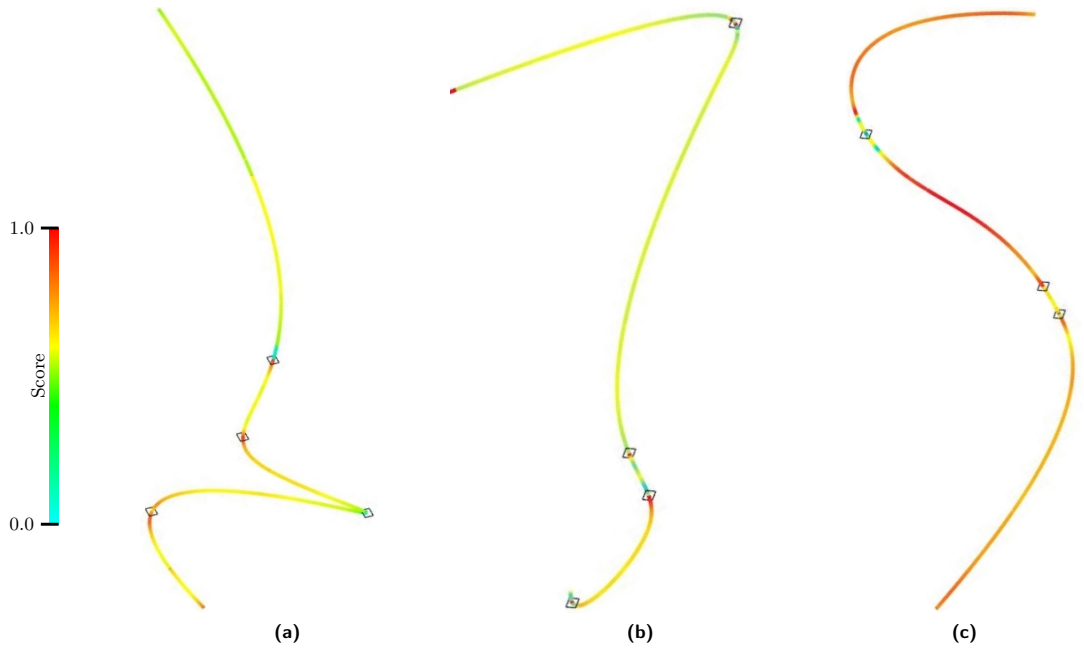

Figure 4: Example curves colored by score, where high scores are red and low scores are blue, diamonds represent LCM points.

This knot placement approach ensures that every knot interval $[u_i, u_{i+1}]$ contains at least one parameter t_j. Thus, $\mathbf{N}^T\mathbf{N}$ in (1) is regular, because (2) is satisfied. Instead of picking only parameters from $\mathbf{t}$ to become new knots, also averaged parameters could be used. These do not guarantee at least one parameter in each knot interval, such that (2) is not necessarily satisfied. In this case, numerical methods like SVD could be used to solve (1).

Most applications require approximation to a target tolerance rather than a target number of knots. This can be achieved by starting with the minimum number of knots and adding knots using Algorithm 1 until the target tolerance is satisfied.

5 Results and discussion

In this section, we present results of curve parametrization and approximation.

Training and evaluation were parallelized on two systems with Intel-Core-i7@3, 4GHz 4-core processors, each system with 16Gb RAM. With this setup, the training data generation took 51 hours while grid-search and evaluation of trained SVMs took 42 hours. We used our complete set of 14 features for training and evaluations and did not test subsets of these features due to the SVMs ability to discover meaningful and separable representations in high dimensional space. All methods were implemented using MATLAB.

5.1 Parametrization results

Since the scores for the individual point samples are the basis for the knot placement strategy, it is instructive to analyze the score distribution along the sequential point sets. Figure 4 shows three example curves colored by score. The LCM points are marked by diamonds. They split the curve into curve segments.

There are several observations to be made. First, symmetric segments yield a symmetric score distribution. This is due to the symmetry of the features.

Second, the score changes smoothly between two LCM-points, which is due to the smoothness of the feature between two LCM points. At LCM points the score might be discontinuous, since the closest LCM point to the left or right changes discontinuously in some of the geometric features, e.g. F8. Even if the curve does not vary much at an LCM-point, the scores to the left and right can vary significantly. Intuitively, inserting a knot close to the left or right of an LCM would yield similar approximation errors. However, the score does not only measure the local fitness of a knot but also the fitness of its enclosing knot interval. Thus, given the refinement strategy of Algorithm 1 it matters much if the knot is inserted to the left or right of an LCM. This means, moving a knot position slightly to the left or right will not change the overall approximation error much, knowing which segment of the curve needs further refinement has a larger impact on the approximation error. This is consistent with the observation of Piegl and Tiller [21] that there are flexibility intervals within which knot positions can be moved freely without changing the approximation error much.

Third, curvature is a strong indicator for a good knot position. Curve regions with higher curvature usually have a higher score while flat regions have lower score. However, there are exceptions. If two LCM points are close to each other, the score in-between is low even if curvature is high. Again, the score combines the fitness of a knot with the fitness of its enclosing the knot interval. Another factor is curvature variation. Segments with a homogeneous curvature distribution and only little variation of the curvature direction will yield a lower average score. Especially a change in the orientation of the curvature direction as in Figure 4c results in higher average scores for the segment with a local score maximum close to the inflection point.

Fourth, the score usually increases towards LCM-points, except for very short intervals. The increasing score can be explained by the correlation of LCM-locations and curvature maxima, which has been recovered by the SVM. However, in close proximity to LCM-points the score drops (see high curvature regions in Figure 4a and 4b) to maintain a minimum spacing between knots.

5.2 Approximation results

To evaluate the effectiveness our Score-based Knot Placement (SKP) method we compared cubic curve approximations computed by SKP, a knot placement method by Piegl and Tiller (NKTP[1]) [20], and the Adaptive Knot Placement (AKP) method [15]. Like our approach, these methods support instant knot vector computation. We also compare our results to the Dominant-Points-based Knot Placement (DPKP) method by Park et al. [19]. This method is not able to instantly compute the knot vector but uses adaptive refinement which includes curve approximation for every new knot and is strongly related to optimization methods.

In the experiments, the Hausdorff distance is computed for an increasing number of knots in the approximation. For the evaluation of curve approximation quality, the Hausdorff distance is the de facto standard [27, 3, 19]. Figure 5a shows results for average Hausdorff distance over our test set of 250 curves approximated by NKTP, AKP, DPKP, and SKP. In the range from 5 to 25 knots, the approximation by SKP is superior to both NKTP and AKP. The SKP results are also very close to those of the DPKP method, with DPKP performing slightly better.

In Figure 5b we compare run-times of the different algorithms for knot placement on our test set. Run-times of NKTP, AKP and SKP increase only slightly for additional knots with all methods staying below 0.002 seconds for 25 knots. Run-time of the DPKP method increases almost linear with the number of knots resulting in 0.168 seconds for a knot placement of 25 knots.

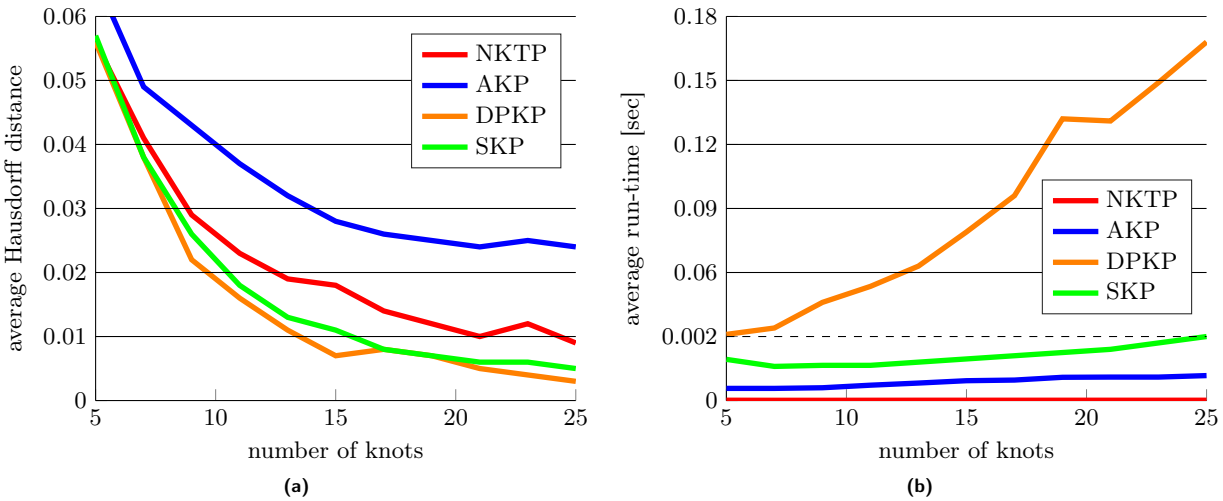

Figure 5: (a) Average Hausdorff distance and (b) average run-times over the test set at different numbers of knots for NKTP, AKP, DPKP, and SKP. Please note the non-linear scale on the ordinate in (b).

Figure 6a and Figure 7a show example point clouds which were used in the experiments. The point cloud in Figure 6a consists of 375, and the one in Figure 7a of 235 points. Figures 6b and 7b show the point clouds together with the approximated B-spline curves for 34 knots in 6b and 7b. Figures 6d and 7d show the difference of curve approximation methods at a detailed region. While SKP and DPKP manage to follow the data points closely, AKP and NKTP miss highly curved and more complex sections. Figures 6c and 7c show the Hausdorff distances for increasing numbers of knots for all methods.

Since knot placement relies on the parametrization $\mathbf{t}$ of the initial point cloud $\mathbf{p}$, we additionally tested the chord length parametrization for the generation of the test data and knot placement. This led to an overall increase of the approximation error for all knot placement methods. In relation to each other, the methods performed comparably to the results presented.

5.3 Discussion

We tested our approach on the 250 point clouds of our test data set for evaluating the performance of SKP. Since SKP uses the geometric properties of point clouds, its strength lies in the approximation of complex curves (see Figures 6d, 7d. Although the AKP method also relies on geometric properties, it does not prioritize the order of the refinement. This leads to the method resulting in high-quality approximations only for large numbers of knots. For very simple curves, results of SKP and NKTP are very close, sometimes with a small advantage for NKTP. The re-computation of the whole knot vector for each additional knot in the NKTP method leads to an oscillation in the Hausdorff distance, which can be seen e.g. in Figure 6c. The SKP method leads to a more monotone decrease in Hausdorff distance.

The approximation quality of SKP is very close to results computed using DPKP. Like SKP, NKTP, and AKP the DPKP method iterates until a certain error threshold is reached. At each iteration step, DPKP computes a curve approximation to decide on segments that need further refinement. The segment with the largest data point to curve distance is selected and refined by knot insertion. Our method is able to produce results of comparable quality, often outperforming DPKP, without any intermediate approximation based solely on score information. Compared to the methods for instant knot vector computation, the DPKP method is computationally expensive. For our test set the computation of 25 knots took an average 84 times longer using DPKP than SKP. Since training data generation and training the SVM have to be done only once prior to all subsequent knot placement tasks, they can be excluded from the performance analysis. Feature extraction has to be done only once at the beginning of the SKP method for each new point cloud. After that, the computational performance of SKP, NKTP, and AKP is equal since all methods instantly yield a knot vector. If the number of knots is given in advance no iterations are required.

Noisy point clouds may lead to an incorrect identification and a larger number of LCM-points. This leads to an increasing number of maxima in the score $\mathbf{s}$ and a decreasing influence of the SVM on the knot placement process. For noisy point clouds or if there is no information about the level of noise in the data, point cloud filters like [26] should be applied before LCM and feature computation.

Recent achievements of machine learning methods, especially deep neural networks, raise the question which method to use for the tasks at hand. For our approach, SVMs had several advantages over other methods. While methods like deep neural

[1] While the abbreviation NKTP is used throughout literature it is never explained.

networks are very flexible, they are hard to parametrize. Moderate training times, due to the small size of the parametric grid of SVMs, enabled us to improve our methods for training set generation and knot selection. Training set generation by exhaustive search is computationally expensive. Due to the SVMs ability to generalize well, even on smaller data sets [6], we were able to outperform state of the art knot placement methods, using only a moderately sized training set. For neural networks there exists an almost linear correlation between training set size and performance [29]. Using SVMs enables our method to be used for smaller real-life data sets. While the SVM does not provide posterior probabilities of class membership, it has been shown that the score may be used to derive such probabilities [22, 34], which makes it a suitable measure for knot placement.

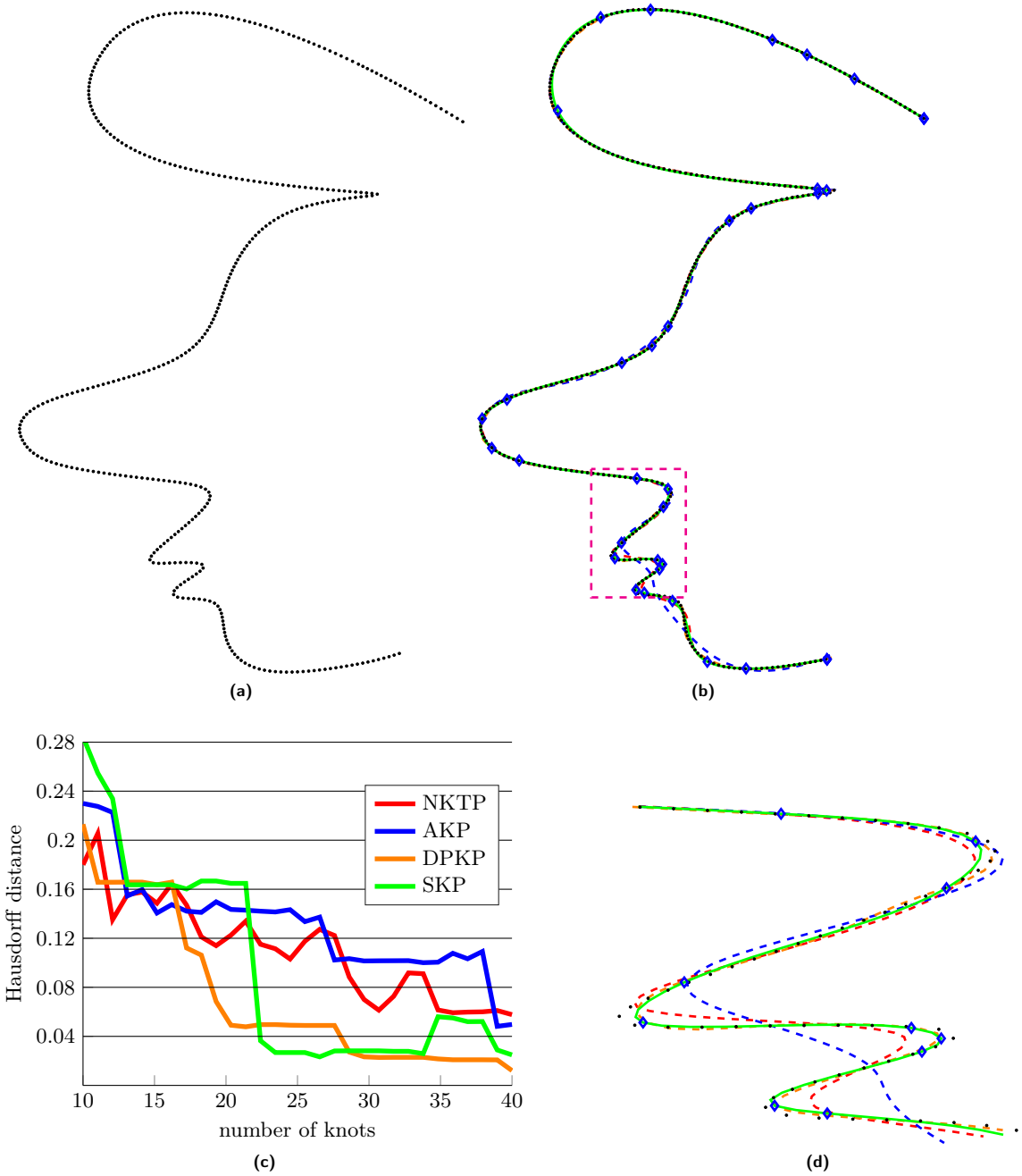

Figure 6: (a) Ordered point set. (b)Point set (black) and curve approximation by NKTP, AKP, DPKP, and ,SKP. Blue diamonds represent the knot positions. (c) Hausdorff distance for different numbers of knots for NKTP, AKP, DPKP, and SKP. (d) Close-up of the dashed magenta region from (b).

6 Conclusion

We propose an approach for parametrization in B-spline curve approximation. We train SVMs to estimate the fitness (score) of pre-computed parameter values of points as knot values. Based on this score we perform knot selection using local score maxima and subdivision of high score regions. The proposed approach is compared to two methods for instant knot vector computation (AKP, NKTP) as well as an adaptive refinement based method (DPKP). Our approach surpasses approximation quality of the AKP and NKTP method. The resulting approximation quality is very close to that of the DPKP method while having significantly lower computation times. We could show that SVMs are able to learn characteristics of positions along a curve where knot placement has a positive impact on approximation quality.

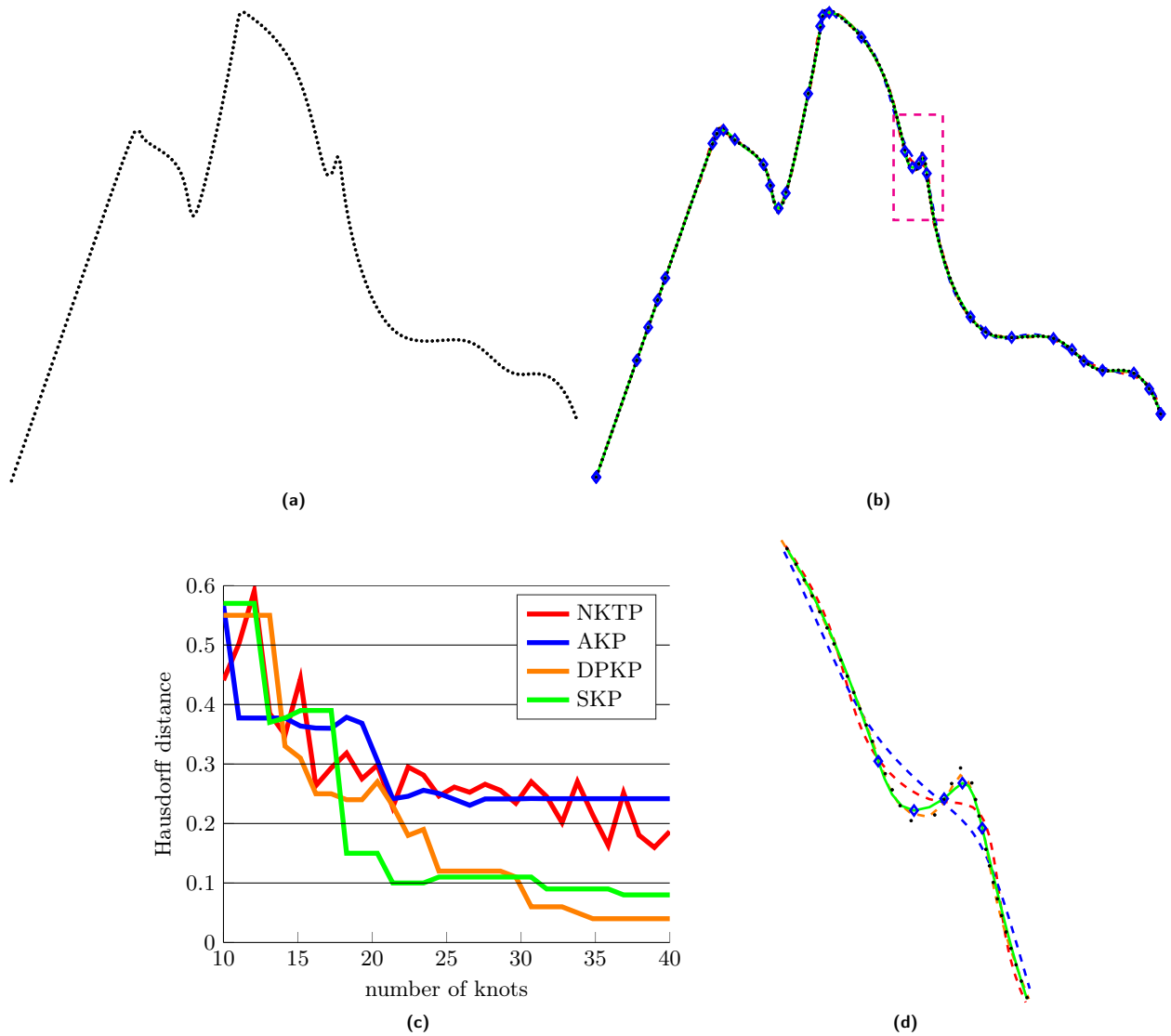

Figure 7: (a) Ordered point set. (b) Point set (black) and curve approximation by NKTP, AKP, DPKP, and our method SKP. Blue diamonds represent the knot positions. (c) Hausdorff distance for different numbers of knots for NKTP, AKP, DPKP, and SKP. (d) Close-up of the dashed magenta region from (b).

Our approach is strongly depended on the training data set which limits the performance of our algorithm to that of the training data. For future work, we will investigate the applicability of neural networks to find suitable parametrizations without using a training data set generated by exhaustive search. Also, we would like to expand our approach to knot placement for surface approximation. We further plan to apply deep learning techniques that render manual feature selection obsolete such as auto-encoders.

Acknowledgments

This research is funded by the Federal Ministry of Education and Research (BMBF) of Germany (project number 02P14A035).

References

[1] G. Arbeiter, S. Fuchs, R. Bormann, J. Fischer, and A. Verl. Evaluation of 3d feature descriptors for classification of surface geometries in point clouds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1644–1650, 2012.

[2] M. Caputo, K. Denker, M. O. Franz, P. Laube, and G. Umlauf. Support vector machines for classification of geometric primitives in point clouds. In *Curves and Surfaces*, pages 80–95. Springer, 2014.

[3] X.-D. Chen, W. Ma, and J.-C. Paul. Cubic b-spline curve approximation by curve unclamping. *Computer-Aided Design*, 42(6):523–534, 2010.

[4] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods.* Cambridge University Press, 2000.

[5] G. E. Farin. *Curves and surfaces for CAGD: a practical guide.* Morgan Kaufmann, 2002.

[6] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems. *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[7] A. Gálvez and A. Iglesias. Firefly algorithm for explicit b-spline curve fitting to data points. *Mathematical Problems in Engineering*, 2013, 2013.

[8] A. Gálvez, A. Iglesias, A. Avila, C. Otero, R. Arias, and C. Manchado. Elitist clonal selection algorithm for optimal choice of free knots in b-spline data fitting. *Applied Soft Computing*, 26:90–106, 2015.

[9] R. Goldenthal and M. Bercovier. Spline curve approximation and design by optimal control over the knots. In *Geometric Modelling*, pages 53–64. Springer, 2004.

[10] S. R. Gunn et al. Support vector machines for classification and regression. *ISIS technical report*, 14:85–86, 1998.

[11] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, Taiwan, 2003.

[12] H. Kang, F. Chen, Y. Li, J. Deng, and Z. Yang. Knot calculation for spline fitting via sparse optimization. *Computer-Aided Design*, 58:179–188, 2015.

[13] P. Laube, G. Umlauf, and M. O. Franz. Evaluation of features for svm-based classification of geometric primitives in point clouds. In *IAPR International Conference on Machine Vision Applications*. IEEE, 2017.

[14] E. T. Lee. Choosing nodes in parametric curve interpolation. *Computer-Aided Design*, 21(6):363–370, 1989.

[15] W. Li, S. Xu, G. Zhao, and L. P. Goh. Adaptive knot placement in b-spline curve approximation. *Computer-Aided Design*, 37(8):791–797, 2005.

[16] C.-T. Lin, W.-C. Cheng, and S.-F. Liang. Neural-network-based adaptive hybrid-reflectance model for 3-d surface reconstruction. *IEEE Transactions on Neural Networks*, 16(6):1601–1615, 2005.

[17] T. Lyche and K. Mørken. Knot removal for parametric b-spline curves and surfaces. *Computer Aided Geometric Design*, 4(3):217–230, 1987.

[18] T. Lyche and K. Mørken. A data-reduction strategy for splines with applications to the approximation of functions and data. *IMA Journal of Numerical analysis*, 8(2):185–208, 1988.

[19] H. Park and J.-H. Lee. B-spline curve fitting based on adaptive curve refinement using dominant points. *Computer-Aided Design*, 39(6):439–451, 2007.

[20] L. Piegl and W. Tiller. *The NURBS book*. Springer Science & Business Media, 2012.

[21] L. A. Piegl and W. Tiller. Surface approximation to scanned data. *The Visual Computer*, 16(7):386–395, 2000.

[22] J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[23] A. Razdan. Knot placement for b-spline curve approximation. *Tempe, AZ: Arizona State University*, 1999.

[24] M. Sarfraz and S. A. Raza. Capturing outline of fonts using genetic algorithm and splines. In *Information Visualisation, 2001. Proceedings. Fifth International Conference on*, pages 738–743. IEEE, 2001.

[25] B. Sarkar and C.-H. Menq. Parameter optimization in approximating curves and surfaces to measurement data. *Computer Aided Geometric Design*, 8(4):267–290, 1991.

[26] O. Schall, A. Belyaev, and H.-P. Seidel. Robust filtering of noisy scattered point data. In *Eurographics/IEEE VGTC Symposium Proceedings Point-Based Graphics*, pages 71–144. IEEE, 2005.

[27] J. Sklansky and V. Gonzalez. Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12(5):327–331, 1980.

[28] F. Steinke, B. Schölkopf, and V. Blanz. Support vector machines for 3d shape processing. *Computer Graphics Forum*, 24(3):285–294, 2005.

[29] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 843–852. IEEE, 2017.

[30] G. Taubin. Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(11):1115–1138, 1991.

[31] V. Tongur and E. Ülker. B-spline curve knot estimation by using niched pareto genetic algorithm (npga). In *Intelligent and Evolutionary Systems*, pages 305–316. Springer, 2016.

[32] E. Ülker. B-spline curve approximation using pareto envelope-based selection algorithm-pesa. *International Journal of Computer and Communication Engineering*, 2(1):60, 2013.

[33] O. Valenzuela, B. Delgado-Marquez, and M. Pasadas. Evolutionary computation for optimal knots allocation in smoothing splines. *Applied Mathematical Modelling*, 37(8):5851–5863, 2013.

[34] V. N. Vapnik and V. Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[35] F. Yoshimoto, M. Moriyama, and T. Harada. Automatic knot placement by a genetic algorithm for data fitting with a spline. In *Proceedings of International Conference on Shape Modeling and Applications*, pages 162–169. IEEE, 1999.

[36] Y. Yuan, N. Chen, and S. Zhou. Adaptive b-spline knot selection using multi-resolution basis set. *IIE Transactions*, 45(12):1263–1277, 2013.

[37] X. Zhao, C. Zhang, B. Yang, and P. Li. Adaptive knot placement using a gmm-based continuous optimization algorithm in b-spline curve approximation. *Computer-Aided Design*, 43(6):598–604, 2011.

View publication stats