A Fast Algorithm for Elastic Shape Distances between Closed Planar Curves

Günay Doğan[1,2] Javier Bernal[2] Charles R. Hagwood[2]
[1] Theiss Research [2] National Institute of Standards and Technology (NIST)
{gunay.dogan,javier.bernal,charles.hagwood}@nist.gov

Abstract

Effective computational tools for shape analysis are needed in many areas of science and engineering. We address this and propose a new fast iterative algorithm to compute the elastic geodesic distance between shapes of closed planar curves. The original algorithm for this has cubic time complexity with respect to the number of nodes per curve. Hence it is not suitable for large shape data sets. We aim for large-scale shape analysis and thus propose an iterative algorithm based on the original one but with quadratic time complexity. In practice, we observe subquadratic, almost linear running times, and that our algorithm scales very well with large numbers of nodes. The key to our algorithm is the decoupling of the optimization for the starting point and rotation from that of the reparametrization, and the development of fast dynamic programming and iterative nonlinear constrained optimization algorithms that work in tandem to compute optimal reparametrizations fast.

1. Introduction

A major challenge of modern data sets is not only their sheer sizes, but that their elements may represent complex geometric structures and objects, e.g., proteins, cells, mechanical parts, facial surfaces, and other morphologies. Typically, one wants to cluster, classify or compare elements of such data sets, but performing such analyses requires defining a proper topological space where these entities reside. However, for data sets of complex elements, the proper space may not be linear thus making it impossible to use algorithmic tools designed for Euclidean spaces. This can be the case for so-called shape spaces. In the particular case where elements of data sets represent shapes of objects, shape spaces have been proposed as an appropriate framework [4, 5, 8, 9, 13, 18, 14, 22]. Vaguely, shape is the aspect of an object where location, rotation and scale information have been removed. In this paper, we focus on work by Srivastava et al. [19] on the computation of the elastic shape geodesic distance between closed planar curves. Their work considers curves as points in an infinite-dimensional Riemannian manifold which allows for the computation of the geodesic distance between them (see Section 2). From this distance a flexible and intuitive geodesic distance measure between curve shapes is obtained, invariant to translation, scaling, rotation and reparametrization. However, it is computationally expensive. The original algorithm proposed in [19] using dynamic programming (*DP*) runs in $O(N^3)$ time, N the number of nodes per curve. This can be prohibitively expensive for many practical applications. In many computer vision applications, such as biometrics, shape retrieval, and in many science and engineering applications, practitioners work with hundreds, sometimes thousands of nodes per curve, and the shape distance algorithm may have to be run many times on pairs from large curve data sets. To enable such large-scale shape analyses, fast computation of the geodesic distance between two curve shapes is crucial. Accordingly, the goal of this paper is to develop a fast algorithm to compute shape distances based on that of Srivastava et al. [19] but considerably cheaper.

We propose a new fast iterative algorithm to compute the elastic geodesic distance between shapes of closed planar curves. The asymptotic time complexity of our algorithm is roughly $O(N^2)$. However, in our experiments, we have observed a linear trend with running times depending on the type of curve data. Mathematically, the distance computation is formulated as a global minimization over triplets of possible starting points (on the curve), rotations and reparametrizations [19]. It is very difficult to find a globally optimum triplet for this problem, and we are not aware of any solutions that guarantee it. However, we have found that our algorithm produces strong minima and frequently global minima. Moreover, when the starting point (or seed) and rotation are fixed and known, it computes a globally optimal reparametrization, just like the *DP* algorithm in [19], but much faster. For this, we developed fast *DP* and iterative nonlinear constrained optimization algorithms that we describe in Section 4. The resulting reparametrization algorithm is plugged into another procedure of outer iterations that computes the optimal starting point and rotation separately (see Algorithm 2 below). The overall computational cost of our algorithm is $O(\epsilon N^2 + K(N^2 + kN))$, K the

978-1-4673-6964-0/15/$31.00 ©2015 IEEE

number of outer iterations, k the number of iterations for reparametrization computation that depends on the type of curve data, and ϵ a small constant. Despite the quadratic terms in the cost, we observed subquadratic, almost linear growth in running times for curves with up to 1024 nodes.

2. Mathematical Formulation

Given S^1, the unit circle, let β_1 and β_2 be members of the family of closed curves $\beta : S^1 \to \mathbb{R}^2$ of class C^2 and unit length (in parametric form $\beta(t) = (\beta_x(t), \beta_y(t)), t \in S^1$).

Let q be the shape function of β: $q(t) = \dot{\beta}(t)/\|\dot{\beta}(t)\|^{1/2}$, $t \in S^1$ (see [19]). The shape of β is defined as the orbit

$$[q] = \{\sqrt{\dot{\gamma}} R q(\gamma) \mid \gamma \in \Gamma, R \in SO(2)\}.$$

Here *SO(2)* is the set of all rotations in $\mathbb{R}^2$ and Γ is the set of all orientation-preserving diffeomorphisms of S^1. Each element of $[q]$ is square-integrable and a shape function of $R\beta(\gamma)$, i.e., a rotation and a reparametrization of β. With q_1, q_2, shape functions of β_1, β_2, respectively, the geodesic distance between the shapes of β_1 and β_2 is then defined as

$$d([q_1],[q_2]) \equiv \inf_{\substack{\gamma\in\Gamma,\\ R\in SO(2)}} \int_{S^1} \|q_1(t) - \sqrt{\dot{\gamma}(t)} R q_2(\gamma(t))\|^2 dt, \tag{1}$$

which can be interpreted as finding $\tilde{q}_2$ in $[q_2]$ closest to q_1.

We aim to devise an algorithm for the computation of (1). For this, we work with real finite intervals as domains of curves rather than S^1. Thus we have $\beta_i : [0,1] \to \mathbb{R}^2$, $i = 1, 2$, of class C^2 and of unit length; moreover $\beta_i(0) = \beta_i(1)$, $\dot{\beta}_i(0) = \dot{\beta}_i(1)$, $\ddot{\beta}_i(0) = \ddot{\beta}_i(1)$, $i = 1, 2$, which imposes that β_i, $\dot{\beta}_i$, $\ddot{\beta}_i$, $i = 1, 2$, be closed. We also have square-integrable q_i, the shape function of β_i, $i = 1, 2$, but with domain $[0, 1]$ instead of S^1. q_1, q_2 are closed as well, and this way, β_1, β_2, q_1, q_2 can all be treated as periodic functions on the real line. This new domain representation brings up the question of how to obtain the appropriate reformulation of the distance computation (1) with $[0, 1]$ in place of S^1. As the value of any such computation will depend on where we start on q_2, and since we have the flexibility to choose the starting point on q_2 (q_2 is periodic), we need to introduce a starting point or seed t_0 into the distance computation to account for this degree of freedom, i.e., we need to include t_0 as a parameter to be optimized in our distance formulation. Hence we introduce the following energy formulation that if optimized is the geodesic distance:

$$E(t_0, \theta, \gamma) \equiv \int_0^1 \|q_1(t) - \sqrt{\dot{\gamma}(t)} R(\theta) q_2(t_0 + \gamma(t))\|^2 dt. \tag{2}$$

Here γ is a diffeomorphism of $[0, 1]$ into $[0, 1]$ (with $\gamma(0) = 0$, $\gamma(1) = 1$, and $\dot{\gamma}(0) = \dot{\gamma}(1)$ to ensure periodicity of $\hat{q}_2$, $\hat{q}_2(t) \equiv q_2(t_0 + \gamma(t))$), $R(\theta)$ is a 2×2 rotation matrix defined by an angle θ, and t_0 is the starting point or seed for the reparametrization of q_2. Note $q_2(t_0 + \gamma(t))$ is well defined for any t_0 by the periodicity of q_2 and $\hat{q}_2$, and therefore (2) is well defined as $\dot{\gamma}$ is positive everywhere.

3. Discretization and Optimization

In practice, we work with curves shuch as β_1 and β_2, given as discrete sets of points, rather than infinite-dimensional geometric objects in the plane. Likewise, rather than optimizing energy (2), we introduce a discretization of the integral in (2) and this is what we actually optimize. We start by partitioning the interval $[0, 1]$ into subintervals with endpoints $t_i = (i-1)h$, where $h = 1/(N-1)$, $i = 1, \ldots, N$, and assume $\beta_{1,i} = \beta_1(t_i)$, $\beta_{2,i} = \beta_2(t_i)$, $i = 1, \ldots, N$, are defined. This enables us to define discretizations of $\dot{\beta}_1$, $\dot{\beta}_2$ by centered finite differences from which discretizations of shape functions q_1, q_2 can be defined:

$$\begin{aligned} \dot{\beta}_j(t_i) &\equiv \dot{\beta}_{j,i} = (\beta_{j,i+1} - \beta_{j,i-1})/2h, \\ q_j(t_i) &\equiv q_{j,i} = \dot{\beta}_{j,i}/\|\dot{\beta}_{j,i}\|^{1/2}, \end{aligned}$$

for $j = 1, 2$, $i = 1, \ldots, N$, with $\beta_{j,1} = \beta_{j,N}$, $\beta_{j,0} = \beta_{j,N-1}$ and $\beta_{j,N+1} = \beta_{j,2}$ by periodicity. Although this gives a discretization of both shape functions q_1 and q_2, we also need a continuous and differentiable function $q_2(t)$ in the discretization of (2). We obtain this by interpolating the set of points $\{q_{2,i}\}_{i=1}^N$ using cubic splines. We continue to denote this interpolant by $q_2(t)$.

We introduce a discrete diffeomorphism vector $\vec{\gamma}$ defined by the entries $\gamma_i = \gamma(t_i)$, $i = 1, \ldots, N$, ($\gamma_1 = 0$, $\gamma_N = 1$), and its discrete derivative $\vec{\rho}$ through approximation by forward finite differences. We opt to use the discrete derivative vector $\vec{\rho}$, rather than $\vec{\gamma}$, as the variable in the discretization of the energy (2), as this makes it easier to compute the discretized energy and its derivatives efficiently, as well as to impose discrete versions of the continuous conditions: $\gamma(0) = 0, \gamma(1) = 1, \dot{\gamma} > 0, \dot{\gamma}(0) = \dot{\gamma}(1)$. The specific relationships between γ_i and ρ_i are for $i = 2, \ldots, N-1$:

$$\rho_i = \frac{\gamma_{i+1} - \gamma_i}{h} \text{ and } \gamma_i = h \sum_{j=1}^{i-1} \rho_j \text{ with } \rho_1 = \gamma_2/h$$

$$\Rightarrow \frac{\partial \gamma_i}{\partial \rho_j} = \begin{cases} h, & \text{if } 1 \le j < i \le N, \\ 0, & \text{if } 1 \le i \le j \le N, \end{cases} \tag{3}$$

with $\gamma_N = h(\rho_1 + \ldots + \rho_{N-1}) = 1$, $\rho_N = \rho_1$, $\gamma_1 = 0$.

Noting $\rho_i > 0$ for all i we now discretize with $\gamma_i(\vec{\rho})$ the integral (2) using the trapezoidal rule for closed curves:

$$E^h(t_0, \theta, \vec{\rho}) = h \sum_{i=1}^{N-1} \|q_{1,i} - \sqrt{\rho_i} R(\theta) q_2(t_0 + \gamma_i(\vec{\rho}))\|^2. \tag{4}$$

Note that we can easily get $\vec{\gamma}$ back from $\vec{\rho}$ using (3), once we find $(t_0, \theta, \vec{\rho})$ that minimizes the energy (4). To devise a

minimization approach, we differentiate (4) with respect to variables to obtain optimality conditions:

$$\frac{\partial E^h}{\partial t_0} = 0, \quad \frac{\partial E^h}{\partial \theta} = 0, \quad \frac{\partial E^h}{\partial \rho_i} = 0.$$

For fixed t_0 and $\vec{\rho}$, we can solve $\frac{\partial E^h}{\partial \theta} = 0$ and obtain $\theta = \theta(t_0, \vec{\rho})$ depending on t_0, $\vec{\rho}$. This can be done using the Kabsch algorithm [11, 12, 19]. Then we substitute θ by $\theta(t_0, \vec{\rho})$ in (4) and obtain the reduced energy

$$\tilde{E}^h(t_0, \vec{\rho}) = E^h(t_0, \theta(t_0, \vec{\rho}), \vec{\rho}). \tag{5}$$

We propose to optimize (5) using an alternating approach: We fix $\vec{\rho}$ and optimize (5) with respect to t_0, computing $\theta(t_0, \vec{\rho})$ with the Kabsch algorithm [11, 12] where required. With optimal t_0, $\theta = \theta(t_0, \vec{\rho})$ fixed in (4), we optimize E^h with respect to $\vec{\rho}$. We alternate between optimizations with respect to t_0 and $\vec{\rho}$ until convergence. The optimization with respect to t_0 is done in $O(N^2)$ time by looping through $t_i = (i-1)h$, $i = 1, \ldots, N$, and evaluating (5) for each t_0. The optimization with respect to $\vec{\rho}$ for fixed t_0, θ is an $O(kN)$ iterative nonlinear constrained optimization problem initialized in $O(\epsilon N^2)$ time with a fast *DP* algorithm (k, ϵ as defined in Introduction). We explain this solution in the next section. The main optimization algorithm is summarized in Algorithm 2 (contrast with Algorithm 1 from original approach of Srivastava et al. [19] or with the LM-BFGS Riemannian optimization approach of Huang et al. [10]).

Algorithm 1 The optimization algorithm in [19]

Initialize $t_i = (i-1)h, \gamma_i^0 = t_i$, $i = 1, \ldots, N$.
for $t_0 =$ all or some of t_i **do**
  Compute optimal $\theta = \theta(t_0, \vec{\gamma}^0)$ with Kabsch algo.
  Fix t_0, θ in (4) and optimize E^h w.r.t. $\vec{\gamma}$ with *DP*.
  Compute $\theta = \theta(t_0, \vec{\gamma})$ for current $\vec{\gamma}$ with Kabsch algo.
end for
Return $(t_0, \theta, \vec{\gamma})$ that gives smallest value for E^h in (4).

Algorithm 2 The main optimization algorithm

Initialize $\rho_i = 1$, $t_i = (i-1)h$, $i = 1, \ldots, N$.
repeat
  Fix current $\vec{\rho}$ in (5) and loop over all t_i to find optimal t_0 and $\theta = \theta(t_0, \vec{\rho})$ with Kabsch algorithm.
  Fix current t_0, $\theta = \theta(t_0, \vec{\rho})$ in (4).
  If 1^{st} iteration, then compute new $\vec{\rho}$ with fast *DP*.
  Optimize E^h in (4) w.r.t. $\vec{\rho}$ with iterative nonlinear constrained optimiz. algo. initialized with current $\vec{\rho}$.
until Energy change $< tol = 10^{-6}$ or iteration # > 50.

4. Fast Optimization for the Diffeomorphism $\vec{\gamma}$

A crucial step in Algorithm 2 for the optimization of the energy (5) is the computation of the optimal diffeomorphism $\vec{\gamma}$ or its derivative $\vec{\rho}$ (related by (3)) when we fix t_0 in (5), actually t_0, θ in (4), as part of our alternating optimization approach. With $\vec{\rho}_0$ as the current $\vec{\rho}$ at start of the step, for fixed t_0, $\theta = \theta(t_0, \vec{\rho}_0)$, we deal with an energy depending on a single vector variable:

$$E^h_{t_0}(\vec{\rho}) = E^h(t_0, \theta, \vec{\rho}) \text{ or } E^h_{t_0}(\vec{\gamma}) = E^h(t_0, \theta, \vec{\rho}(\vec{\gamma})). \tag{6}$$

The global minimum for $E^h_{t_0}(\vec{\gamma})$ can be obtained using a *DP* algorithm [3, 19]. The drawback of the original *DP* algorithm used for this problem is that it has $O(N^2)$ time complexity [19]. This is expensive for curves with many nodes, especially if we need to repeat this computation for many t_0 candidates. An efficient alternative is to use an iterative algorithm, which can have $O(N)$ cost per iteration, but usually converges to a local minimum of the energy. We combine the strengths of the two approaches: we first use a fast approximate *DP* algorithm. Our *DP* algorithm works on a reduced search space and produces a rough approximate global minimum very fast. It still has quadratic time complexity, albeit with a very small constant. Then we use this approximation as the initial iterate for an efficient iterative nonlinear constrained optimization algorithm, which takes it to the precise global minimum very fast in a small number of iterations, each of which has $O(N)$ cost. We describe each of these algorithms in the following subsections.

4.1. A Faster Dynamic Programming Algorithm

The goal is to modify the *DP* algorithm in [3, 17] in a way that makes it significantly faster while producing a rough approximation of a global solution. As in [3, 17], an $N \times N$ square grid of $[0,1] \times [0,1]$ is formed such that for all integers i, j, $0 \le i$, $j \le N-1$, grid point labeled (i,j) is identified with planar point (t_{i+1}, t_{j+1}). Given such (i,j), $i > 0$, $j > 0$, a set $\Gamma(i,j)$ of piecewise linear functions from $[0, t_{i+1}]$ onto $[0, t_{j+1}]$ is defined by

$$\begin{aligned}\Gamma(i,j) \equiv \{\gamma \mid \gamma : [0, t_{i+1}] \to [0, t_{j+1}], \gamma(0) = 0, \gamma(t_{i+1}) = \\ t_{j+1},\ \gamma \text{ piecewise linear, strictly increasing,} \\ \text{all vertices of graph of } \gamma \text{ at grid points}\}.\end{aligned}$$

The energy $E^h_{i,j}(t_0, \theta, \vec{\gamma})$ needed to reach (i,j) from $(0,0)$ with γ in $\Gamma(i,j)$ is then

$$E^h_{i,j}(t_0, \theta, \vec{\gamma}) = \sum_{m=1}^{i+1} h_m \| q_{1,m} - \sqrt{\dot{\gamma}_m} R(\theta) q_2(t_0 + \gamma_m) \|^2,$$

where $\gamma_m \equiv \gamma(t_m)$, (note $\gamma_1 = 0$, $\gamma_{i+1} = t_{j+1}$), $\vec{\gamma}$ is the discretization of γ in terms of these γ_m's, and $h_1 = h_{i+1} \equiv h/2$, $h_m \equiv h$, $1 < m < i+1$.

$H_{i,j}(t_0, \theta) \equiv \min_{\gamma \in \Gamma(i,j)} E^h_{i,j}(t_0, \theta, \vec{\gamma})$ is then the minimum energy needed to reach (i,j) from $(0,0)$, and the aim is then to find γ in $\Gamma(N-1, N-1)$ that can be used

to compute $H_{N-1,N-1}(t_0,\theta)$. Since $E^h_{i,j}(t_0,\theta,\vec{\gamma})$ is clearly additive over the segments of the graph of any γ, this can be done with *DP* as it is then apparent that the desired solution can be obtained as the result of incrementally combining solutions of previously solved subproblems.

For this purpose, given grid points labeled (k,l) and (i,j), $k=l=0$ or $0<k<i$, $0<l<j$, then a localized form of energy over the line segment joining (k,l) and (i,j) is defined as

$$F_{k,l,i,j}(t_0,\theta) \equiv \sum_{m=k+1}^{i+1} h_m \|q_{1,m} - \sqrt{L}R(\theta)q_2(t_0+f_m)\|^2,$$

where L is the slope of the segment, $f_m \equiv f(t_m)$, f the linear function from $[t_{k+1},t_{i+1}]$ onto $[t_{l+1},t_{j+1}]$, the graph of which is the segment, (note $f_{k+1}=t_{l+1}$, $f_{i+1}=t_{j+1}$), and $h_{k+1}=h_{i+1}\equiv h/2$, $h_m\equiv h$, $k+1<m<i+1$.

$$\text{Thus } H_{i,j}(t_0,\theta) = \min_{\substack{k=l=0 \text{ or}\\ 0<k<i, 0<l<j}} H_{k,l}(t_0,\theta)+F_{k,l,i,j}(t_0,\theta),$$

and this is then the recurrence relation of the *DP* approach.

For the purpose of making the above *DP* approach faster, we reduce the sets $\Gamma(i,j)$ by requiring that (i,j) and the graph of each γ in $\Gamma(i,j)$ be inside a strip M along the diagonal of the unit square, a strip of width d ($\approx 0.3\sqrt{2}$ in our experiments), as illustrated in Figure 1. Then only grid points in M are examined.

Additionally, we restrict grid points (k,l), $k<i$, $l<j$, for which $F_{k,l,i,j}(t_0,\theta)$ is defined, to a square neighborhood $N(i,j)$ (covering a 3×3 subgrid in our experiments), the right-upper corner of which is $(i-1,j-1)$, as illustrated in Figure 1. Note the computation of $H_{i,j}(t_0,\theta)$ above must then be adjusted accordingly.

Since intuitively the graph of an optimal γ should not deviate much from the diagonal and the distance between consecutive vertices of the graph should not be large, using M and $N(i,j)$ this way should speed up the *DP* algorithm without compromising its accuracy very much.

4.2. Iterative Optimization for the Diffeomorphism

We propose an iterative nonlinear constrained optimization algorithm to minimize the energy $E^h_{t_0}(\vec{\rho})$ in (6) with respect to $\vec{\rho}$ (t_0,θ are fixed) implemented using the interior point method [7, 6, 20]. Specifically, we use Matlab `fmincon` with the interior point option. Note however that `fmincon` by itself does not guarantee superior performance; it requires efficient implementations of the energy gradient and Hessian. We now derive formulas for this purpose and explain how to apply them in linear time. This is one of our key contributions and is central to achieving a fast minimization algorithm for $E^h_{t_0}(\vec{\rho})$. We start with

$$E^h_{t_0}(\vec{\rho}) = h\textstyle\sum_{k=1}^{N-1}\|f_k\|^2,$$
$$f_k = q_{1,k} - \sqrt{\rho_k}R(\theta)q_2, \quad q_2 = q_2(t_0+\gamma_k(\vec{\rho})). \tag{7}$$

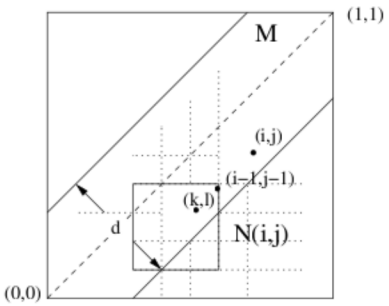

Figure 1. Only grid points in the strip M of width d can be vertices of the graph of any γ. Given that grid point (i,j) is a vertex or endpoint of the graph, only grid points in the intersection of the strip M and the square neighborhood $N(i,j)$, such as (k,l), can be candidates for the lower left endpoint of the line segment of the graph of γ ending at (i,j). $N(i,j)$ covers a 4×4 subgrid here as opposed to a 3×3 subgrid in our experiments.

Derivatives of f_k with respect to components of $\vec{\rho}$ are

$$\frac{\partial f_k}{\partial \rho_i} = -\rho_k^{-\frac{1}{2}} R\left(\frac{q_2}{2}\delta_{ki} - \rho_k \dot{q}_2 \frac{\partial \gamma_k}{\partial \rho_i}\right),$$
$$\frac{\partial^2 f_k}{\partial \rho_i \partial \rho_j} = \frac{\partial}{\partial \rho_j}\frac{\partial f_k}{\partial \rho_i}.$$

Thus the component $G_i = \frac{\partial E^h_{t_0}}{\partial \rho_i}$ of the gradient of $E^h_{t_0}$ is

$$G_i = 2h\textstyle\sum_{k=1}^{N-1}\langle f_k, \frac{\partial f_k}{\partial \rho_i}\rangle =$$
$$-h\rho_i^{-\frac{1}{2}}\langle f_i, Rq_2\rangle - 2h\sum_{k=i+1}^{N-1}\rho_k^{\frac{1}{2}}\langle f_i, R\dot{q}_2\rangle. \tag{8}$$

The last term above is a cumulative sum that we compute in linear time with Matlab `cumsum`. It is obtained from equations (3) for $\frac{\partial \gamma_k}{\partial \rho_i}$ which allow the following matrix-vector products to be computed in linear time using `cumsum`:

$$\sum_{k=1}^{N-1}\frac{\partial \gamma_k}{\partial \rho_i}x_k = h\sum_{i+1}^{N-1}x_k,$$
$$\sum_{j=1}^{N-1}\sum_{k=1}^{N-1} x_k \frac{\partial \gamma_k}{\partial \rho_i}\frac{\partial \gamma_k}{\partial \rho_j}y_j = h^2\sum_{k=i+1}^{N-1}x_k\sum_{l=1}^{k-1}y_l. \tag{9}$$

Using identities (9) and Hessian $\mathbf{H} = [H_{ij}] = [\frac{\partial G_i}{\partial \rho_j}]$, one can show that the Hessian-vector product $\mathbf{H}\vec{y}$ can also be computed in linear time, even though the Hessian $\mathbf{H}$ is a dense matrix if assembled explicitly. Hence, by carefully recasting the formulations of the energy gradient and the Hessian-vector product, we achieve $O(N)$ complexity per iteration of the optimization scheme. This is in contrast with the gradient descent algorithm of [19], which has $O(mN)$ cost per iteration, due to the use of m Fourier basis functions. We also find that the number of iterations with our

method is much smaller than with a gradient descent algorithm (which is known to converge slowly [21]). Thus the total cost of our optimization algorithm is $O(kN)$, where k is the number of iterations, a number that changes for each experiment depending on the type of curve data. Nonetheless, in our tests, on average, for different values of N, k has been small and has resulted in small computation times.

5. Numerical Experiments

In this section we empirically demonstrate the speed and accuracy gains achieved by Algorithm 2. We compare it with Srivastava et al.'s Algorithm 1 [19]. To this end we downloaded an implementation of Algorithm 1 from its authors' web site [3]. The core of that code is the *DP* algorithm (*original-DP*) written in C; thus it is fast, albeit with asymptotic time complexity $O(N^2)$, N the number of nodes per curve. Algorithm 1 calls *original-DP* for all or a specified fraction (say every fifth) of the curve nodes. Thus the time complexity of Algorithm 1 is $O(N^3)$ [19]. On the other hand, the core of our implementation of Algorithm 2 is the fast *DP* algorithm (*fast-DP*) and the iterative nonlinear constrained optimization algorithm (*iterative*), the former initializing the latter (*fast-DP+iter*), as described in Section 4. *fast-DP* is coded in Fortran, so its speed is similar to that of *original-DP*. Actually, it has quadratic time complexity as well, but because of its limited search space, has a much lower constant, running much faster than *original-DP* in practice. *iterative* is implemented completely in Matlab; therefore, its code is not as fast as a C or Fortran implementation but this is compensated by the fact that the computations of the energy value (7), the gradient (8) and the Hessian-vector product (with respect to $\vec{\rho}$) all have $O(N)$ time complexity. Thus the time complexity of *iterative* is the number of iterations k times $O(N)$. We ran most of our experiments on a laptop computer with Intel Core™2 Duo CPU T9300 @ 2.50GHz and 4GB RAM. The operating system was 32-bit Centos 5 Linux. The version of the Matlab™installation was 7.14.0.739 (R2012a).

For our numerical experiments we worked with a synthetic shape data set and three real shape data sets: Cells A & B ((biological) cell boundary curves), MPEG7 [1, 15], and Leaf [2, 16] (see Figure 2). Synthetic shape data (synthetic curves) allowed us to evaluate the performance of our algorithm on a variety of well-known but simple geometric curves. On the other hand, real shape data allowed us to do this on examples with a wider range of shape variability.

We started by evaluating the ability of the different algorithms mentioned above to compute optimal diffeomorphisms. Then we evaluated the ability of Algorithm 2 to compute geodesic distances between curve shapes. For both evaluations we used synthetic and cell boundary curves. Finally, we performed shape retrieval experiments on MPEG and Leaf curve shapes in order to evaluate indirectly the ability of Algorithm 2 to compute geodesic distances between shapes with a large range of variability.

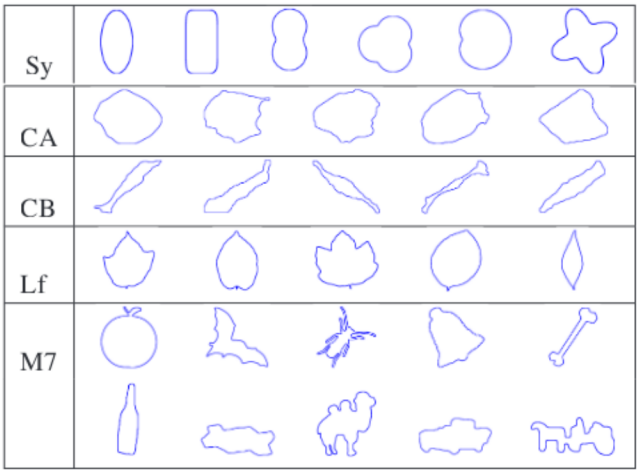

Figure 2. **Curve examples for testing.** Synthetic (Sy), Cells A (CA), Cells B (CB), Leaf (Lf), and MPEG7 (M7).

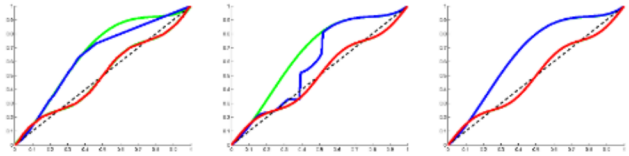

Figure 3. **Diffeomorphism examples for testing.** Synthetic diffeomorphisms γ^{easy} (in red), γ^{hard} (in green, (partly) overlapped with blue) used in experiments. Computed solutions for γ^{easy} are still in red as they are correct, and for γ^{hard} they are in blue. Only *original-DP* and *fast-DP+iter* computed γ^{hard} correctly.

Computing optimal diffeomorphisms:
In the experiments described below we used seed $t_0 = 0$ and did not apply rotations to any curve, i.e., used $R = Id$ (the identity matrix). With $N = 256$ we performed the experiments on all synthetic and cell boundary curves using synthetic diffeomorphisms. Given one such diffeomorphism, $\gamma(t)$, we deformed a given synthetic or cell boundary curve β_2 with it to obtain $\beta_1(t) = \beta_2(\gamma(t))$, and then using each algorithm mentioned above tried to recover $\vec{\gamma}$ from the shape functions $q_1(t), q_2(t)$ of $\beta_1(t), \beta_2(t)$ by optimizing the energy (7) for $\vec{\rho}$ and setting $\vec{\gamma} = \vec{\gamma}(\vec{\rho})$ (see Section 4.2). Accordingly, we used two synthetic diffeomorphisms: γ^{easy}, which was a sine wave added to the identity $\alpha(t) = t$ that stayed close to the identity, hence was easier to recover, and γ^{hard}, which was far from the identity and whose derivative got close to zero thus causing at times large errors and high computation times (see Figure 3). In particular, for the epitrochoid, we found that each algorithm successfully recovered γ^{easy} in these exper-

iments, but γ^{hard} proved difficult to get. The search space of *fast-DP* was not large enough, so it returned a suboptimal solution. *iterative* got stuck at a local minimum of (7), and the quality of the solution was not good. When we used *fast-DP* to initialize *iterative* (*fast-DP+iter*), we recovered both γ^{easy} and γ^{hard} successfully, just as we did with *original-DP*, but at a fraction of the computational cost. These solutions are shown in Figure 3 (for the epitrochoid).

With increasing resolution, $N = 64, 128, \ldots, 4096$, we performed the same experiments described above on all synthetic and cell boundary curves. For each N and curve, an L^2 error was calculated by

$$\|\vec{\gamma}^* - \vec{\gamma}^h\|_{L^2} \approx h(\Sigma_i |\gamma_i^* - \gamma_i^h|^2)^{1/2},$$

where $\vec{\gamma}^*$ was the true solution and $\vec{\gamma}^h$ was the computed solution. Errors and computation times for the epitrochoid and the first Cells A curve are listed in Table 1. We observed that the L^2 error of *original-DP* converged with a rate of $O(N^{-0.5})$ and that of *fast-DP+iter* converged much faster with a rate of $O(N^{-1.5})$, but began to plateau when it reached the internal tolerance of the iterative solver.

Although *original-DP* and *fast-DP* both have quadratic computational cost, *fast-DP* was about 20X faster (with an accuracy trade-off). *iterative*, when used on its own, was very fast for γ^{easy} (up to 60X faster than *original-DP* for $N = 4096$) but slowed down for γ^{hard} as it ended at a local minimum. The running times of *iterative* were subquadratic and grew slowly with increasing N. This made *iterative* advantageous for curves with large numbers of nodes.

We obtained the best trade-off for computation times and accuracy when we used the solution from *fast-DP* to initialize *iterative* (*fast-DP+iter*). The running times were fast, about 10X-15X faster than *original-DP* for large curves and the L^2 errors were up to 20X lower. For curves with $N = 4096$ nodes, *original-DP* took about 1.5 minutes, whereas *fast-DP+iter* took 6-7 seconds. *original-DP* was still competitive for curves with N=128-256 nodes.

Computing the geodesic distances:
Here we evaluate how well Algorithm 2 computes the geodesic distance between two curves. We examine both distance values and computation times. Ideally, for testing, we would work with two curves that we know exactly. Accordingly we would know the optimal triple $(t_0, \theta, \vec{\gamma})$ to compute the exact geodesic distance between the curves. But this is difficult in practice, so instead, we tried a variety of numerical tests with synthetic and cell boundary curves where we partly knew the exact parameters and outputs.

Scalability and accuracy for synthetic curves with increasing N: We first constructed a setup in which we knew the theoretical distance exactly. We took a synthetic curve β_2 (see Figure 2), shifted its nodes by $0.25N$, i.e., set t_0 to 0.25, rotated it by an angle of $\theta = \pi/3$, and obtained a second curve β_1, one version by reparametrizing the shifted and rotated curve with γ^{easy}, another with γ^{hard} (see Figure 3). This way, we knew the geodesic distance between β_1 and β_2 had to be exactly zero. We tested this using the limaçon as an example (tests with different synthetic curves produced consistent behavior). We sampled the curve at increasing number of nodes $N = 64, 128, 256, 512, 1024$, then computed the geodesic distances between the two versions of the limaçon. We report distances and running times for both Algorithm 1 and Algorithm 2 in Table 2.

We found that Algorithm 2 gave better distance values than Algorithm 1 in these tests. In the example constructed with γ^{easy}, our algorithm computed exact 0 for the distance, whereas Algorithm 1 computed small values around 0.02-0.03. In the case of γ^{hard}, both algorithms got stuck at local minima and computed significant nonzero distances. Algorithm 2 also did very well in terms of computation time. The computational cost of Algorithm 1 grows cubically with respect to N, independent of the data. The performance of our algorithm, on the other hand, depends on the data, and is larger for difficult examples, e.g., at $N = 1024$, the computational time for γ^{easy} was 2.6 s, but it was 89 s for γ^{hard} (as it took more iterations). These running times were still much less than the 1.5 hours it took with Algorithm 1. In contrast with the cubic time complexity of Algorithm 1, the empirical timings of our algorithm clearly exhibited a subquadratic, almost linear trend, which implied superior scalability for curves with hundreds, even thousands of nodes. Note that the computational cost of Algorithm1 can be reduced by skipping t_0 seed candidates, e.g., by a factor of 10 by using only every 10th node as a seed. But this has an accuracy trade-off; one can miss important features and might even get stuck at unfavorable local minima by skipping nodes. In these experiments, we made numerical choices that favor *accurate* distance computations. Thus, for Algorithm 1, we did not skip nodes, and for Algorithm 2, we used small tolerances and stringent stopping criteria, which translated into a larger number of iterations in experiments.

Cost and correctness of distances between various curves: We examine now how Algorithm 2 performed on a more diverse collection of curves and compare it with Algorithm 1. We evaluate both correctness and speed. In general, for two arbitrary curves, it is not feasible to know the theoretical geodesic distance between them. But, as we did above for scalability tests, we can construct examples, where we know the exact distance is zero. We took a known curve β_2 with $N = 256$, shifted it by $0.25N$, rotated it by $\theta = \pi/3$, and obtained a second curve β_1, one version by reparametrizing the shifted and rotated curve with γ^{easy}, another with γ^{hard}. Distances computed by Algorithms 1 and 2 for four synthetic curves and first four Cells A curves are given in Table 3 (there γ^e is γ^{easy} and γ^h is γ^{hard}). Of the 16 result pairs listed, Algorithm 2 got 5 exact zeros, 6 small nonzero values (< 0.1), and 5 large values (> 0.1).

	timings for epitrochoid, γ^{easy}							timings for epitrochoid, γ^{hard}						
	N=64	*128*	*256*	*512*	*1024*	*2048*	*4096*	*N=64*	*128*	*256*	*512*	*1024*	*2048*	*4096*
original-DP	.023	.087	.36	1.4	5.8	23	93	.024	.087	.35	1.4	5.8	23	93
fast-DP	.001	.006	.022	.086	.33	1.3	5.6	.001	.006	.027	.084	.35	1.4	5.7
iterative	.15	.094	.079	.11	.23	.38	1.4	.17	.51	.59	.29	.2	6.9	8.6
fast-DP+iter	.081	.071	.093	.17	.47	1.6	6.0	.098	.10	.13	.23	.56	1.7	6.6
	timings for cell boundary curve, γ^{easy}							**timings for cell boundary curve,** γ^{hard}						
original-DP	.022	.087	.36	1.4	5.9	23	94	.021	.086	.37	1.4	5.8	23	94
fast-DP+iter	.10	.09	.15	.20	.50	1.6	6.1	.11	.16	.22	.45	.90	2.3	7.4
	$\times 10^{-4}$ **errors for epitrochoid,** γ^{easy}							$\times 10^{-4}$ **errors for epitrochoid,** γ^{hard}						
original-DP	2.1	1.6	.39	.25	.12	.077	.057	4.2	2.0	1.7	1.0	.71	.49	.35
fast-DP	4.0	1.6	1.1	.50	.33	.23	.17	14.3	9.8	6.9	4.9	3.4	2.4	1.7
iterative	1.5	.51	.18	.063	.022	.008	.004	127	101	71	.089	34	31	20
fast-DP+iter	1.5	.51	.18	.063	.023	.019	.038	1.9	.67	.23	.083	.030	.017	.017
	$\times 10^{-4}$ **errors for cell boundary curve,** γ^{easy}							$\times 10^{-4}$ **errors for cell boundary curve,** γ^{hard}						
original-DP	2.2	1.4	.36	.17	.088	.058	.040	4.3	2.1	1.3	.87	.61	.43	.30
fast-DP+iter	1.6	.53	.11	.26	.007	.005	.010	2.2	.86	.18	.038	.010	.005	.005

Table 1. L^2 **errors and timings of diffemorphism computations for increasing** N**.** We reparametrized $\beta_2(t)$ (epitrochoid or cell boundary curve), with γ (γ^{easy} *or* γ^{hard}). We then used the shape $q_2(t)$ of the original curve $\beta_2(t)$ and the shape $q_1(t)$ of the reparametrized curve $\beta_1(t)$ to recover γ. We report computation times (in seconds) and L^2 errors.

	Timings for γ^{easy}				
	$N = 64$	128	256	512	1024
Algo.1	1.5	11	91	740	5922
Algo.2	0.5	0.3	0.5	1.1	2.6
	Timings for γ^{hard}				
Algo.1	1.4	11	92	735	5917
Algo.2	4	11	19	50	89
	Distances for γ^{easy}				
Algo.1	.0283	.0339	.0234	.0213	.0205
Algo.2	0	0	0	0	0
	Distances for γ^{hard}				
Algo.1	.2400	.2354	.2279	.2264	.2258
Algo.2	.2092	.1902	.2138	.2137	.2145

Table 2. **Scalability and accuracy of distance computations for increasing** N**.** The theoretically zero geodesic distance between the shapes of two versions of a limaçon was computed.

	ellipse	limaçon	hippopede	epitrochoid
γ^e	.025/ 0	.023/ 0	.025/ 0	.022/ 0
γ^h	.399/ 0	.228/.214	.358/.020	.270/ .037
	cell 1	cell 2	cell 3	cell 4
γ^e	.089/.073	.050/.017	**.129/.248**	.046/.012
γ^h	.506/.165	**.394/.536**	.277/.083	.394/.257

Table 3. **Zero distance between a curve and its reparametrization.** With $N = 256$ the theoretically zero geodesic distance between the shapes of two versions of each of eight curves is computed. Computed distances from Algorithm 1 and Algorithm 2 are given as pairs of numbers "*dist 1 / dist 2*" for each curve.

Algorithm 1 got no exact zeros, 7 small nonzero values, and 9 large values. Algorithm 1 returned smaller distances than Algorithm 2 in only 2 of the 16 result pairs (shown in bold). Algorithm 2 clearly did a better job in this set of experiments where zero distance was expected.

Another important benchmark for each algorithm is the computation of a distance matrix consisting of geodesic distances between curves in a given set. The distance matrix can be used for various statistical analyses, including clustering and classification of curves based on their shapes. We computed two distance matrices, one for synthetic curves (see Table 4), and one for cell boundary curves (not shown), all sampled at N=256. In the case of synthetic curves, Algorithm 2 found lower distance values for all pairs. Thus, in this case, Algorithm 2 found better minima. The total computation time of Algorithm 2 was 733 s, or about 22% of that of Algorithm 1 (3364 s). In the case of cell boundary curves, the results were not as clear-cut. Algorithm 2 returned lower distances in 72 of 100 distance pairs and higher in the remaining 28 (see Table 5). However, in only 11 (of the 28), Algorithm 1 produced significantly lower distances (with difference $\geqslant 0.02$ shown in bold). The difference in computation times was dramatic. Algorithm 1 took 9134 s or 2.5 hours to compute a 10×10 distance matrix, whereas Algorithm 2 took only 270 s or 4.5 minutes.

Experiments with MPEG7 and Leaf data sets: In order to test our algorithm on shapes with a large range of variability, we used subsets of the MPEG7 [1, 15], and Leaf [2, 16] data sets (see Figure 2). We used ten shape

	ellipse	super-ellipse	limaçon
ellipse	0 / 0	.259 / .253	.254 / .249
supr-ellps	.258 / .253	0 / 0	.312 / .306
limaçon	.253 / .250	.312 / .306	0 / 0
hippopede	1.102/ .980	1.128/1.018	1.256/1.069
epitrochd	.367 / .361	.386 / .381	.224 / .220
clover	.763 / .733	.691 / .682	.734 / .723

	hippopede	epitrochoid	clover
ellipse	1.122/ .972	.371 / .346	.802 / .678
supr-ellps	1.141/ .992	.391 / .364	.714 / .578
limaçon	1.314/1.110	.231 / .210	.746 / .619
hippopede	0 / 0	1.250/1.063	1.167/ .958
epitrochd	1.292/1.055	0 / 0	.634 / .584
clover	1.161/1.023	.633 / .629	0 / 0

Table 4. **Distance matrix of synthetic curves.** With $N = 256$ geodesic distances between shapes of synthetic curves are computed. Computed distances from Algorithm 1 and Algorithm 2 are given as pairs of numbers "*dist 1 / dist 2*" for each curve pair.

A1	.432	.327	.332	**.596**	.427	**.561**	.363
A2	.443	.329	.337	**.635**	.428	**.593**	.375
A1	**.541**	.537	**.582**	.540	.578	**.560**	**.561**
A2	**.638**	.545	**.618**	.553	.580	**.593**	**.639**
A1	.565	**.546**	**.555**	.506	.516	.323	.375
A2	.569	**.606**	**.585**	.520	.532	.329	.379
A1	.515	**.318**	**.550**	.386	.512	.488	**.527**
A2	.525	**.338**	**.614**	.392	.515	.497	**.554**

Table 5. **Distances of cell curves when Algorithm 1 produced lower values.** Here A1 is for Algorithm 1, A2 is for Algorithm 2.

classes from the MPEG7 data set, with ten example curves per class. The classes were apple, bat, beetle, bell, bone, bottle, brick, camel, car, carriage. From the Leaf data set, we used five shape classes with fifteen example curves per class. The classes were Acer Opalus, Acer Rufinerve, Alnus Cordata, Alnus Viridis, Callicarpa Bodinieri. Since we did not know the exact values of the shape distances between any two curves of these data sets, we used an indirect way of evaluating the performance of our algorithm: we did shape retrieval experiments and scored the results using the bull's eye test [15]. For this, we computed all pairwise distances within each data set (from curves resampled to $N = 256$ nodes), a total of $(10 \times 10)^2 = 10000$ distances for MPEG7, $(5 \times 15)^2 = 5625$ distances for Leaf. These distance computations were the most time-consuming part of these experiments. We used both our laptop computer and additional desktop computers for these computations. To be able to compare the computation times, we normalized the desktop computer times to equivalent times for the laptop computer. The computation times for the distance matrices with Algorithms 1 and 2, respectively, were 240 hours and 38.5 hours (6X speed-up) for MPEG7 shapes, 129 hours and 12.5 hours (10X speed-up) for Leaf shapes. Note that, for curves with larger numbers of nodes, the contrast in computation time would be even more dramatic.

	MPEG4 # retrieved			Leaf # retrieved			
	20	15	10	30	25	20	15
A1	.978	.965	.950	1.0	.993	.982	.956
A2	.976	.966	.952	1.0	.994	.985	.961

Table 6. **Shape retrieval hit rates.** Percentages of successful retrievals for decreasing retrieval set size for Algorithms 1 and 2.

We then executed the shape retrieval tests as follows. For each curve, say one of the bottle curves from the MPEG7 data, we chose the corresponding twenty curves with the smallest distance to that bottle curve. This selection (or query) would have 100% success if all ten bottle curves were included in the retrieved/returned set of twenty curves, 50% success if only five were included. We repeated this for all curves in the data set, summed the retrieval scores and normalized the sum to a percentage to get an aggregate hit rate. We found that we had near-perfect retrieval scores for both Algorithms 1 and 2. So we made the retrieval set size more stringent and gradually reduced it to ten curves from twenty curves per retrieval. The two algorithms were still very successful. As results show in Table 6, Algorithm 2 (A2) performed a little better than Algorithm 1 (A1).

Finally, to evaluate the effectiveness of the minimization by both Algorithms 1 and 2, we compared the distance values for each curve pair from the two algorithms and checked which one produced smaller distances implying smaller energy values and better minimization. We found that Algorithm 2 produced smaller distances 82.0% of the time for MPEG7 curves and 85.4% of the time for Leaf curves.

6. Conclusions

We propose a new fast iterative algorithm to compute the elastic shape geodesic distance between closed planar curves based on that of Srivastava et al. [19]. Their algorithm uses dynamic programming (*DP*) for reparametrizations and has cubic time complexity which is costly for curves with many nodes. Our algorithm, on the other hand, uses an iterative nonlinear constrained optimization algorithm for reparametrizations initialized by a fast *DP* algorithm. It has quadratic time complexity but actually runs in subquadratic, almost linear time, scaling very well with increasing number of nodes. We also see that, for most of our examples, it produces smaller distance values than Srivastava et al.'s which indicates it reaches better minima.

Acknowledgment

First author acknowledges support by NIST grant 70NANB13H018.

References

[1] MPEG-7 shape data set source web link. http://www.dabi.temple.edu/ shape/MPEG7/ MPEG7dataset.zip. Accessed: 2014-11-10. 5, 7

[2] One-hundred species plant leaves data set source web link. https://archive.ics.uci.edu/ml/datasets/One-hundred+plant+species+leaves+data+set. Accessed: 2014-11-10. 5, 7

[3] Source code from Statistical Shape Analysis and Modeling Group, Florida State University. http://ssamg.stat.fsu.edu/downloads/ClosedCurves2D3D.zip. Accessed: 2014-06-20. 3, 5

[4] F. L. Bookstein. Size and shape spaces for landmark data in two dimensions. *Statistical Science*, 1:181–242, 1986. 1

[5] F. L. Bookstein. *Morphometric tools for landmark data: Geometry and Biology*. Cambridge University Press, Cambridge, 1991. 1

[6] R. H. Byrd, J. C. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000. 4

[7] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999. 4

[8] J. L. Dryden and K. V. Mardia. *Statistical Shape Analysis*. John Wiley and Sons, Chichester, 1998. 1

[9] U. Grenander. *General Pattern Theory*. Clarendon Press, Oxford, 1994. 1

[10] W. Huang, K. A. Gallivan, A. Srivastava, and P.-A. Absil. Riemannian optimization for elastic shape analysis. In *The 21st International Symposium on Mathematical Theory of Networks and Systems*, 2014. 3

[11] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics*, 32(5):922–923, 1976. 3

[12] W. Kabsch. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics*, 34(5):827–828, 1978. 3

[13] D. G. Kendall. Shape manifolds, procrustean metrics and complex projective spaces. *Bulletin of the London Mathematical Society*, 16:81–121, 1984. 1

[14] E. Klassen, A. Srivastava, W. Mio, and S. Joshi. Analysis of planar shapes using geodesic paths on shape spaces. *IEEE Pattern Analysis and Machiner Intelligence*, 10:372383, 2004. 1

[15] L. J. Latecki. Retrieval results for shape similarity on the MPEG-7 data set. 5, 7, 8

[16] C. Mallah, J. Cope, and J. Orwell. Plant leaf classification using probabilistic integration of shape, texture and margin features. *Computer Graphics and Imaging: Signal Processing, Pattern Recognition and Applications (CGIM2013)*, page 798, 2013. 5, 7

[17] W. Mio, A. Srivastava, and S. Joshi. On shape of plane elastic curves. *International Journal of Computer Vision*, 73(3):307–324, 2007. 3

[18] E. Sharon and D. Mumford. 2d-shape analysis using conformal mappings. *In Proceedings of IEEE Conference on Computer Vision*, page 350357, 2004. 1

[19] A. Srivastava, E. Klassen, S. Joshi, and I. Jermyn. Shape analysis of elastic curves in Euclidean spaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(7):1415–1428, 2011. 1, 2, 3, 4, 5, 8

[20] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, 107(3):391–408, 2006. 4

[21] S. Wright and J. Nocedal. *Numerical optimization*, volume 2. Springer New York, 1999. 5

[22] L. Younes. Computable elastic distance between shapes. *SIAM Journal of Applied Mathematics*, 58:565586, 1998. 1