ELPIGRAPH: SCALABLE METHOD FOR CONSTRUCTING ELASTIC PRINCIPAL GRAPHS

Algorithm description, version 1.1, 27/02/2018

Luca Albergante[1], Evgeny M. Mirkes[2], Emmanuel Barillot[1], Alexander N. Gorban[2], Andrei Zinovyev[1]

[1]*Institut Curie, Université PSL, U900 INSERM/Curie/Mines Paritech, rue d'Ulm 26*
Paris, 75005, France

[2]*Department of Mathematics, University of Leicester, University Road*
Leicester LE1 7RH, UK

Abstract

This document provides the most basic description of the ElPiGraph method for scalable construction of principal graphs. In the ElPiGraph method, the mapping of a principal graph into multidimensional data space is regularized by minimizing the stretching of the graph edges and the deviation from harmonicity for the graph stars. The structure of the graph is learnt simultaneously with learning the graph mapping function through application of the graph grammar approach. Basic notions of the method are introduced and the pseudo-code descriptions of the main algorithms are provided. Graph grammar operations needed to construct elastic principal curves, elastic closed principal curves and elastic principal trees are documented. Application of resampling to approximating complex branching and loopy data distributions is considered. Current implementations of ElPiGraph allows constructing complex data approximators for millions of data points in data spaces of hundreds of dimensions.

Elastic principal graphs: basic definitions

Let G be a simple undirected graph with set of vertices V and set of edges E. Let us consider a map ϕ:$V \rightarrow \mathbf{R}^m$ which describes an embedding of the graph into a multidimensional space by mapping a node of the graph to a point in the data space. *k-star* in a graph G is a subgraph with $k + 1$ vertices $v_{0,1,\ldots,k} \in V$ and k edges $\{(v_0, v_i)|i = 1, .., k\}$. We define an *elastic graph* as a graph with selected families of k-stars S_k and for which for all $E^{(i)} \in E$ and $S_k^{(j)} \in S_k$, the corresponding elasticity moduli $\lambda_i > 0$ and $\mu_{kj} > 0$ are defined. *Primitive elastic graph* is an elastic graph in which every non-leaf node (with the number of neighbours more than one) is associated with a k-star formed by *all* neighbours of the node. All k-stars in the primitive elastic graph are selected, i.e. the S_k sets are completely determined by the graph structure. Non-primitive elastic graphs are not considered here, but they are used, for example, for constructing 2D and 3D elastic principal manifolds, where a node in the graph can be a center of two 2-stars, in a rectangular grid [2].

Elastic principal tree is an acyclic primitive elastic principal graph.

Let $E^{(i)}(0)$, $E^{(i)}(1)$ denote two vertices of the graph edge $E^{(i)}$ and $S_k^{(j)}(0),\ldots, S_k^{(j)}(k)$ denote vertices of a k-star $S_k^{(j)}$ (where $S_k^{(j)}(0)$ is the central vertex, to which all other vertices are connected).

For computational implementations, it is convenient to describe the structure and the elastic properties of the graph by elastic matrix $E(G)$. The elastic matrix is a $|V|\times|V|$ symmetric matrix with non-negative elements containing the edge elasticity moduli λ_i at the intersection of rows and lines, corresponding to each pair $E^{(i)}(0)$, $E^{(i)}(1)$, and the star elasticity module μ_{kj} at the diagonal element corresponding to $S_k^{(j)}(0)$. Therefore, $E(G)$ can be represented as a sum of two matrices Λ and M:

$E(G) = \Lambda(G) + M(G)$,

where Λ is an analog of weighted adjacency matrix for the graph G, with elasticity moduli playing the role of weights, and $M(G)$ is a diagonal matrix having non-zero elements only in the centers of the stars with corresponding elasticity moduli for the stars. Λ can in principle contain negative elements, corresponding to repulsive, negative springs.

An example of elastic matrix is shown in Figure ElMatrix,A.

The *elastic energy of the graph embedment* is defined as a sum of squared edge lengths (weighted by the λ_i elasticity moduli) and the sum of squared *deviations from harmonicity* for each star (weighted by the μ_{kj}).

$$U^{\phi}(G) := U_E^{\phi}(G) + U_R^{\phi}(G)\,, \qquad (1)$$

$$U_E^{\phi}(G) := \sum_{E^{(i)}} \lambda_i \left(\phi(E^{(i)}(0)) - \phi(E^{(i)}(1))\right)^2\,, \qquad (2)$$

$$U_R^{\phi}(G) := \sum_{S_k^{(j)}} \mu_{kj} \left(\phi(S_k^{(j)}(0)) - \frac{1}{k}\sum_{i=1}^{k} \phi(S_k^{(j)}(i))\right)^2 \qquad (3)$$

The deviation from harmonicity term in the case of 2-star is a simple surrogate for minimizing the local curvature. In the case of *k*-stars with *k*>2 it can be considered as a generalization of local curvature defined for a branching point [3]. Note that $U_R^{\phi}(G)$ can be re-written as

$$U_R^{\phi}(G) := \sum_{S_k^{(j)}} \frac{\mu_{kj}}{k} \sum_{i=1}^{k} (\phi(S_k^{(j)}(0)) - \phi(S_k^{(j)}(i)))^2 - \sum_{S_k^{(j)}} \frac{\mu_{kj}}{k^2} \sum_{i=1,\, p=1,\, i<>p}^{k} (\phi(S_k^{(j)}(i)) - \phi(S_k^{(j)}(p)))^2 , \qquad (4)$$

i.e., the term $U_R^{\phi}(G)$ can be considered as a sum of the energy of elastic springs connecting the star centers with its neighbors (with elasticity moduli μ_{kj}/k) and the energy of negative (repulsive) springs connecting all non-central nodes in a star pair-wise (with negative elasticity moduli $-\mu_{kj}/k^2$). The resulting system of springs which energy is minimized is shown in Figure FigSprings. In simple terms, the elasticity of the principal graph contains three parts: positive springs corresponding to elasticity of graph edges (Figure ElMatrix,B), negative repulsive springs describing the node repulsion to make the graph embedding function ϕ as smooth as possible (Figure ElMatrix,D), positive springs representing the correction term such that the smoothing would correspond to the deviation of harmonicity (Figure ElMatrix,C).

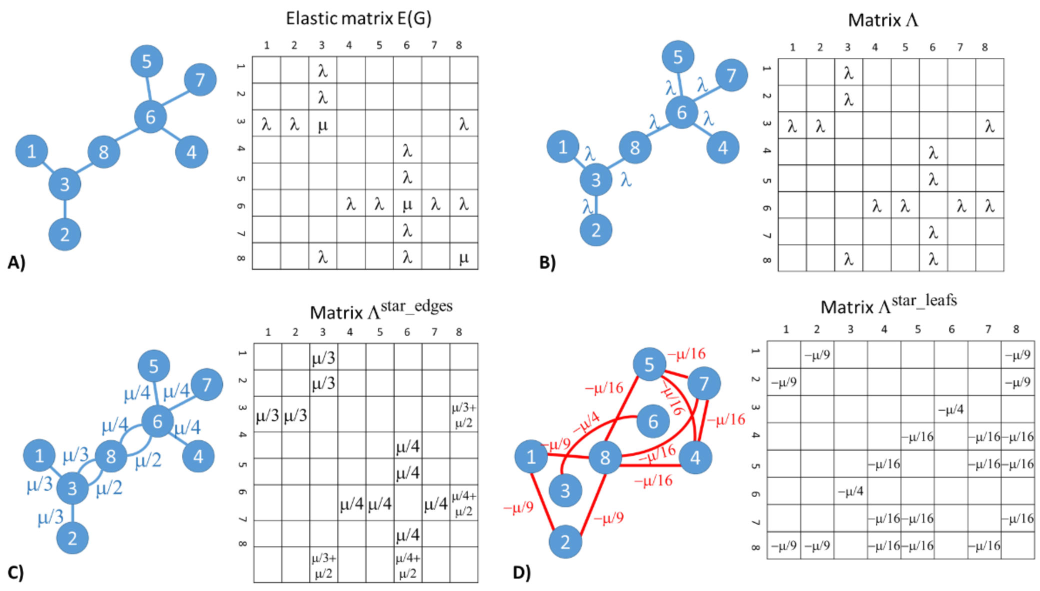
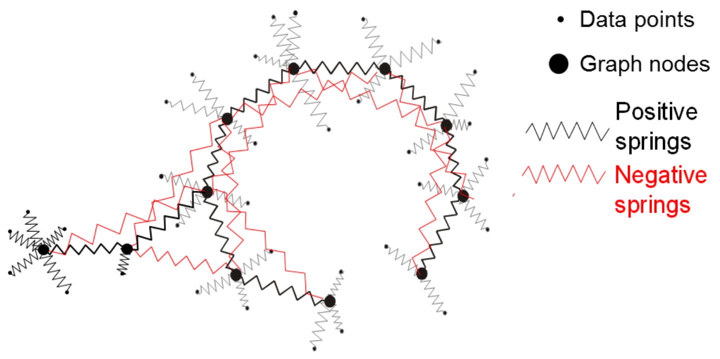


Figure ElMatrix. Simple example of an elastic matrix for a primitive elastic graph constructed from eight nodes and having two branching points (at nodes 3 and 6). The elastic graph is characterized by elasticity moduli λ (stretching, assigned to edges) and μ (bending, assigned to graph stars), common to all edges and stars respectively. A) Elastic matrix *E*(*G*), which is transformed for further computations into three matrices Λ, Λ^{star_edges}, Λ^{star_leafs}. B) Part of elastic matrix Λ with edge elasticity moduli. C) Matrix Λ^{star_edges} containing the contributions from star edges. D) Matrix Λ^{star_leafs} containing the contributions from springs with negative elasticity connecting leafs of each star into a clique. Empty table cells contain zeros.



FigSprings. System of springs representing the elastic energy $U^{\varphi}(X,G)$ which is to be minimized by ElPiGraph. Data points are connected by positive springs with elasticity moduli $1/N$ to the closest graph nodes.

It is convenient to represent (4) in the matrix form, by transforming $E(G)$ into three auxiliary matrices Λ, Λ^{star_edges} and Λ^{star_leafs}. Λ^{star_edges} is a weighted adjacency matrix for the edges connected to star centers, with elasticity moduli μ_{kj}/k, where k is the number of edges connected to the star center. Λ^{star_leafs} is a weighted adjacency matrix for the negative springs (shown in red in FigSprings and FigElMatrix), with elasticity moduli $-\mu_{kj}/k^2$. Example of transforming the elastic matrix $E(G)$ into three weighted adjacency matrices Λ, Λ^{star_edges} , Λ^{star_leafs} is shown in Figure ElMatrix.

For the system of springs shown in FigSprings, if one applies a distributed force to the nodes then propagation of the node perturbation will be described by a matrix which is a sum of three graph Laplacians.

$$L(G,E(G)) = L(\Lambda) + L(\Lambda^{star\_edges}) + L(\Lambda^{star\_leafs}) \,.$$

We remind that a Laplacian matrix for a weighted adjacency matrix A is computed as

$$L(\mathrm{A})_{ij} \;=\; \delta_{ij}\sum_k \mathrm{A}_{kj} - \mathrm{A}_{ij},$$

where δ_{ij} is the Kroeneker symbol.

Basic optimization algorithm

We define that the optimal embedding of a graph must minimize the squared distance between the position of graph nodes in and the data points and, at the same time, minimize the elastic energy of the graph embedment serving a penalty term for “irregularity” of the graph embedment. The irregularity can be manifested by stretching and non-equal distance between graph node positions (penalized by $U_E^{\phi}(G)$ and partly by $U_R^{\phi}(G)$) or by deviation from harmonicity (penalized by $U_R^{\phi}(G)$).

Assume that we have defined a partitioning K of all data points such that $K(i) = \arg\min_{j=1\ldots|V|}(X_i - \phi(V_j))^2$ returns an index of a node in the graph which is the closest to the ith data point among all graph nodes. Then our objective function for minimization is

$$U^{\phi}(X,G) = \frac{1}{N}\sum_{j=1}^{|V|}\sum_{K(i)=j}(X_i - \phi(V_j))^2 + U^{\phi}(G),$$

where N is the number of data points.

The objective of the core optimization algorithm is to find such a map $\phi{:}V\longrightarrow \mathbf{R}^m$ that $U^{\phi}(X,G) \rightarrow$ min over all possible elastic graph G embeddings in $\mathbf{R}^m$. In practice, we are looking for a local minimum of $U^{\phi}(X,G)$ by applying the expectation-minimization type algorithm, which pseudo-code is provided below:

ALGORITHM 1: BASE GRAPH OPTIMIZATION FOR A FIXED STRUCTURE OF THE ELASTIC GRAPH

```
1) Initialize the graph G, its elastic matrix E(G) and the map ϕ.
2) Compute matrix $L(G, E(G))$
3) Partition the data by proximity to the embedded nodes of the graph (i.e., compute K:{X}→{V} mapping of a data
   point i to a graph node j)
4) Solve linear equation for determining the new map ϕ:

   $\sum_{j=1}^{|V|}\left(\frac{|\{K(i)=j\}|}{N}\delta_{ij} + L(G,E(G))_{ij}\right)\phi(V_j) = \frac{1}{N}\sum_{K(i)=j} X_i$ , j = 1…|V|,

   where $\delta_{ij}$ is the Kroeneker symbol.

5) Iterate 3-4 till the map ϕ does not change more than ε in some appropriate measure.
```

The base algorithm optimizes the graph embedment ϕ, but contains neither a recipe for initializing the map ϕ, nor recipe for choosing the structure of the graph *G*. This initialization can be produced by a number of ways, starting from the structural analysis of the *kNN* graph after some dimension reduction and/or pre-clustering (e.g., computing a spanning tree) or by other heuristic approaches.

By contrast with majority of existing state-of-the-art methods, ElPiGraph exploits the top-down approach to data approximation based on growing the optimal structure of the graph *G*, by systematic application of some pre-defined set of graph grammar operations. This allows to explore many graph topologies, in the space of all possible graph topologies, using a gradient descent-like algorithm.

Graph grammar approach for determining the optimal graph topology

Any construction the graph-based data approximator should deal simultaneously with two inter-related aspects: determining the most optimal topological structure of the graph and determining the optimal map for embedding this graph topology into the multidimensional space. An exhaustive approach would be to consider all possible graph topologies (or, from a certain class, e.g., all possible trees), find the best mapping of all them into the data space, and select the best one. In practice, due to combinatorial explosion, testing all possible graph topologies is feasible only for a restricted number of nodes in the graph, or under restrictive constraints (e.g., only trivial linear graphs, or assuming a restricted set of topologies with a pre-defined number and types of branching). Determining a globally optimal embedment of a graph with a given topology is usually challenging, because of the energy landscape complexity. This means that in practice one has to use an optimization approach in which both graph topology and the mapping function should be learnt simultaneously [3].

A graph grammar-based approach for constructing such an algorithm was suggested in [3]. This top-down approach starts from a simple graph G_0 and a simple map $\phi_0(G_0)$. A set of grammar operations is defined which can transform both the graph topology and the map, starting from a given pair $\{G_i, \phi_i(G_i)\}$. Each grammar operation Ψ^p produces a set of new graph embedments possibly taking into account the dataset *X*:

$$\{\{D^k, \phi(D^k)\}, k = 1 \dots s\} = \Psi^p\left(\{G_i , \phi_i (G_i)\}, X\right) .$$

Given a pair $\{G_i, \phi_i(G_i)\}$ and a set of *r* different graph operations $\{\Psi^1, \dots, \Psi^r\}$ (which we call a "graph grammar") and an energy function $\overline{U}^{\phi}(X, G)$, at each step of the algorithm the most energetically optimal candidate graph embedment is selected:

$$\{G_{i+1}, \phi_{i+1}(G_{i+1})\} = \arg min_{\{D^k,\phi(D^k)\}}\{\overline{U}^{\phi(D^k)}(D^k, X) : \{D^k, \phi(D^k)\} \in \bigcap_{p=1\dots r} \Psi^p\left(\{G_i , \phi_i (G_i)\}, X\right)\} ,$$

where $\{D^k, \phi(D^k)\}$ is supposed to be optimized (fit to data) after application of a graph grammar, using ALGORITHM 1 with initialization suggested by the graph grammar application (see below).

The pseudocode for this algorithm is provided below:

ALGORITHM 2: GRAPH GRAMMAR BASED OPTIMIZATION OF GRAPH STRUCTURE AND EMBEDMENT

1. Initialize current graph embedment by some graph topology and some initial map $\{G_0, \phi_0(G_0)\}$.
2. For the current graph embedment $\{G_i, \phi_i(G_i)\}$, apply all grammar operations from a grammar $\{\Psi^1, \ldots, \Psi^r\}$, and generate a set of s candidate graph embedments $\{D^k, \phi(D^k), k = 1 \ldots s\}$.
3. Further optimize each candidate graph embedment using ALGORITHM 1, and obtain a set of s energy values $\{\bar{U}^{\phi(D^k)}(D^k)\}$.
4. Among all candidate graph embedments, select an embedment with the minimum energy $\{G_{i+1}, \phi_{i+1}(G_{i+1})\}$.
5. Repeat 2-4 until the graph contains a required number of nodes.

Note that the energy function $\bar{U}^{\phi}(X, G)$ used to select the optimal graph structure is not necessary the same energy as (1-3) and can include various penalties to give less priority to certain graph configurations (such as those having excessive branching as described below). Separating energy functions $U^{\phi}(X, G)$ used for fitting a fixed graph structure to the data and $\bar{U}^{\phi}(X, G)$ used to select the most favorable graph configuration allows achieving flexibility in defining the strategy of selecting the graph topologies.

Simple graph grammar operations

Let us define in details two base grammar operations “bisect an edge” and “add a node to a node”.

GRAPH GRAMMAR OPERATION “BISECT AN EDGE”

Applicable to: any edge of the graph
Update of the graph structure: for a given edge {A,B}, connecting nodes A and B, remove {A,B} from the graph, add a new node C, and introduce two new edges {A,C} and {C,B}.
Update of the elasticity matrix: the elasticity of edges {A,C} and {C,B} equals elasticity of {A,B}.
Update of the graph embedment: C is placed in the mean position between A and B embedments.

GRAPH GRAMMAR OPERATION “ADD NODE TO A NODE”

Applicable to: any node of the graph
Update of the graph structure: for a given node A, add a new node C, and introduce a new edge {A,C}
Update of the elasticity matrix:
```
if A is a leaf node (not a star center) then
        the elasticity of the edge {A,C} equals to the edge connecting A and its neighbor,
        the elasticity of the new star with the center in A equals to the elasticity of the star centered in the neighbor of A
else
        the elasticity of the edge {A,C} is the mean elasticity of all edges in the star with the center in A,
        the elasticity of a star with the center in A does not change.
```
Update of the graph embedment:
```
if A is a leaf node (not a star center) then
        C is placed at the same distance and the same direction as the edge connecting A and its neighbor,
else
        C is placed in the mean point of all data points for which A is the closest node
```

ALGORITHM 2 with a graph grammar containing only ‘bisect an edge’ operation, and initializing the graph by two nodes connected by a single edge, produces an *elastic principal curve.*

ALGORITHM 2 with a graph grammar containing only ‘bisect an edge’ operation, and initializing the graph by four nodes connected by four edges without branching, produces a *closed elastic principal curve* (called elastic principal circle, for simplicity).

ALGORITHM 2 with the growing graph grammar containing both operations 'bisect an edge' and 'add a node to a node', and initializing the graph by two nodes connected by a single edge produces an *elastic principal tree*. In the case of a tree or other complex graphs, it is advantageous to improve the ALGORITHM 2 by providing an opportunity to 'roll back' the changes of the graph structure. This gives an opportunity to get rid of unnecessary branching or to merge split branches created in the history of graph optimization, if this is energetically justified (see Figure GraphGrammar). This possibility can be achieved by introducing a shrinking grammar. In the case of tree, the shrinking grammar consists of two operations 'remove a leaf node' and 'shrink internal edge' (defined below). Then the graph growth can be achieved by alterating two steps of application of the growing grammar with one step of application of the shrinking grammar. In each such a cycle, one node will be added to the graph.

GRAPH GRAMMAR OPERATION "REMOVE A LEAF NODE"

Applicable to: node A of the graph of connectivity one
Update of the graph structure: for a given edge {A,B}, connecting nodes A and B, remove {A,B} and A from the graph
Update of the elasticity matrix:
```
if B is the center of a 2-star then
        put zero for the elasticity of the star for B (B becomes a leaf)
else
        do not change the elasticity of the star for B
```
Update of the graph embedment: all nodes besides A keep their positions.

GRAPH GRAMMAR OPERATION "SHRINK INTERNAL EDGE"

Applicable to: any edge {A,B} such that both A and B has connectivity more than 1.
Update of the graph structure: for a given edge {A,B}, connecting nodes A and B, remove {A,B}, reattach all edges connecting A with its neigbours to B, remove A from the graph.
Update of the elasticity matrix:
The elasticity of the new star with the center in B becomes average elasticity of the previously existing stars with the centers in A and B
Update of the graph embedment: B is placed in the mean position between A and B embedments.

Robust local version of elastic principal graphs

Since the objective function used by ElPiGraph contains a quadratic data approximation term, it can be very sensitive to the presence of points located at large distance from a graph node. In practice it means that even moderate background noise distorts significantly the structure of the principal graph. In order to deal with this, a simple "data-driven" trimming approach can be used by a simple modification of the data approximation term, and defining the objective function as

$$U^{\phi}(X,G) = \frac{1}{\sum w_i}\sum_{j=1}^{|V|}\sum_{K(i)=j} w_i \min\{||X_i - \phi(V_j)||^2, {R_0}^2\} \quad + U^{\phi}(G).$$

In other words, data points located more distantly than R_0 (a parameter called "trimming radius") from a graph node position do not contribute, for a given data point partitioning, to the optimization equation at the maximization step. However, these data points might appear at the distance smaller than R_0 at the next algorithm iteration: therefore, it is not equivalent to permanent pre-filtering "outliers". This approach to constructing a robust local version of elastic principal graphs was described previously in (Gorban et al, Archives of Data Science, 2017), with many examples. The approach is similar to the "data-driven" trimming suggested in (Gordaliza, 1991).

The implementation of ElPiGraph includes this simple option for applying data-driven trimming. The robust version of the algorithm can tolerate significant amount of uniformly distributed background noise (see Figure RobustLocal) and even deal with self-intersecting data distributions, if the trimming radius is properly guessed. Choice of the trimming radius is critical for the success of constructing a principal graph robust to the presence of noise. ElPiGraph includes a function for estimating the coarse-grained radius of the data (see 'Determining the coarse-grained data radius'), which can be used as a good initial guess for the trimming radius.

In case of the application of the trimming option, the graph growing cannot be initialized in the same way as the global ElPiGraph algorithm. Good results are obtained instead by a rough estimation of local data density in a limited number of data points. The graph can be initialized afterwards by two nodes, one is placed into the data point characterized by the highest local density and another node is placed into the data point closest to the first one (but not coinciding).

In case of existence of several well-separable clusters in the data, with the distance between them larger than R_0, robust version of ElPiGraph can approximate the principal graph only for one of them, completely disregarding the rest of the data. In this case, the approximated part of the data can be removed and the robust ElPiGraph can be re-applied. For example, this procedure will construct a second (third, fourth, etc.) principal tree. Such an approach will approximate the data by disconnected "principal forest".

Potentially the trimming radius can vary along the principal graph. Alternative ways of constructing robust principal graphs include using piece-wise quadratic subquadratic error functions (PQSQ potentials), see (Gorban et al, 2016). It uses computationally efficient non-quadratic functions for approximating a dataset. These two approaches will be developed in the future versions of ElPiGraph.

Control for excessive branching

Principal trees can suffer from excessive branching in the regions of data space where the local effective dimension of data is far from one. Playing with elasticity parameters, especially λ allows partially control of the excessive branching: however, this changes other properties of the elastic principal graph.

ElPiGraph allows controling appearance of branches of higher orders independently on elasticity parameters. For this, a couple of parameters α and β can be used such that α penalizes appearance of higher-order stars through adding a fee for their edges, and β scales the higher-order star harmonicity penalties.

More precisely, in the Algorithm 2, ElPiGraph uses the following energy function for selecting the most optimal tree structure:

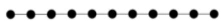
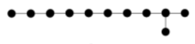
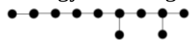
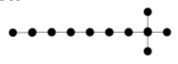
$$\bar{U}^{\Phi}(X,G) = MSE + \sum_{E_i}\left(\lambda_i + \alpha\left(\max(2, k_{E_i(0)}, k_{E_i(1)}) - 2\right)\right)\|\phi(E_i(0)) - \phi(E_i(1))\|^2 + \sum_{S_i}\mu_i(k_i)^{\beta}\left\|\phi(S_i(0)) - \frac{1}{k_i}\sum_{j=1}^{k_i}\phi(S_i(j))\right\|^2, \quad (5)$$

where k_i is the degree of the star S_i with its center $S_i(0)$ in the node i. If node i is not a center of any star then we assume $k_i = 0$.

In case of presence of excessive branching, it is recommended to set up the first branching control parameter α to a small value (e.g., 0.01). Using second parameter, β, remains experimental and much less efficient in typical scenarios.

Note that changing value of α from 0 to a large value (e.g., 1) allows gradual change from the absence of excessive branching penalty to effective interdiction of branching (thus, constructing a principal curve instead of a tree as a result).

In Figure BranchingControl, an example of application of the penalty α is shown using a simple example of a data distribution characterized by a "think turn" area.

Imagine various graph structures each having 11 nodes and 10 edges of equal unity length. Then, for example,  graph is characterized by 10λ contribution to the elastic energy from edges. Graph with one star  will be characterized by 10λ+3α penalty,  by 10λ+6α,  by 10λ+8α.

Implementations of the algorithm

The ElPiGraph method is implemented in several programming languages:

R: https://github.com/Albluca/rpgraph

principal developer: Luca Albergante, first release date: October 2017

MATLAB: https://github.com/auranic/Elastic-principal-graphs/wiki/Basic-use-of-Elastic-Principal-Graphs-Matlab-package

principal developers: Andrei Zinovyev and Eugene Mirkes, first release date: August 2017)

Matlab implementation also contains a wrapper to VDAOEngine library used to launch the Java code for ElPiGraph. However, usage of this wrapper is currently not advised and it is kept only for historical reference.

Java:

principal developer: Andrei Zinovyev, first release date: November 2007

Java implementation of ElPiGraph makes a part of VDAOEngine library (https://github.com/auranic/VDAOEngine/) for high-dimensional data analysis developed by Andrei Zinovyev. Developing Java code for ElPiGraph is not an active effort anymore, and implementation of ElPiGraph in Java does not scale as good as other implementations.

Python implementation is available at https://github.com/AlexiMartin/ElPiGraph.P.

Scala implementation is available from https://github.com/mraad/elastic-graph.

All implementations contain the core algorithm code described in this document. However, implementations differ in the set of functionalities improving the core algorithm such as robust local version of the algorithm, boosting up the algorithm performance by local optimization of candidate graphs, using the resulting graphs in various applications.

Conclusion

This text provides the most basic self-contained description of the ElPiGraph method for scalable constructing of principal graphs, based on using elastic energy functional and application of graph grammars.

A number of extensions of the basic algorithm is not described in this document and provided elsewhere, such as:

1) Estimating the coarse-grained radius of the dataset that can be used to estimate the robustness radius automatically
2) Boosting up the algorithm performance by local optimization of the candidate graph structures
3) Using resampling strategy to construct an ensemble of principal trees which can be used to assemble an elastic principal graph
4) Using the principal graphs for feature selection.

ElPiGraph is a highly parallel method and it currently allows constructing complex data approximators such as principal trees for datasets with millions of points in data spaces of hundreds of dimensions without preliminary drastic dimension reduction. Systematic use of subsampling should allow further improvement of performance.

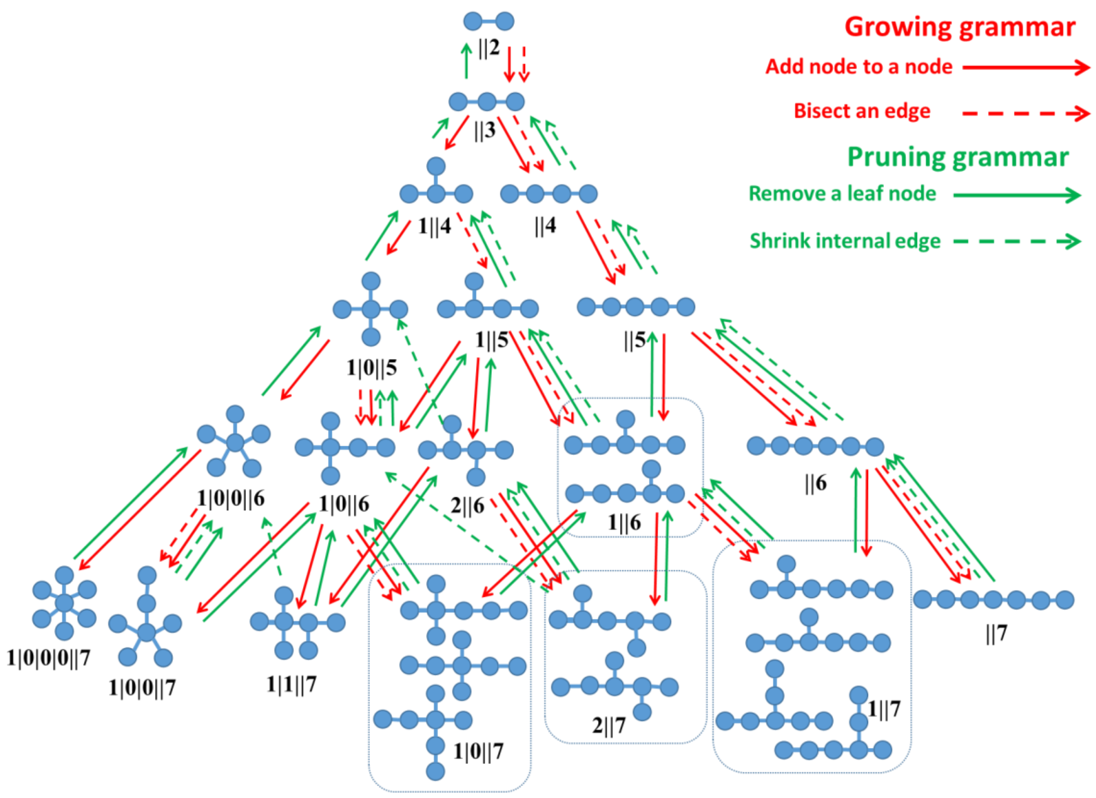


Figure GraphGrammar. **Principal tree structure optimization space for "add node to a node", "bisect an edge" growing graph grammar and "remove a leaf node", "shrink internal edge" shrinking graph grammar**. We start with a simple 2-node graph from which the optimization can grow larger trees up to certain number of nodes (7 in this case). Each topologically distinct graph structure is denoted by a code specifying the total number of nodes in the graph and the number of stars of various orders starting from the 3-stars. All applicable graph grammar operations are shown by different edge types. The process of optimizing the graph topology is a Markovian process on this 'topology transition' graph. From a given graph topology and its embedment, a step is done leading to the minimum total energy of the graph embedment $U^{\phi}(G,X)$ after fitting to the data X. Note that increasing topological complexity is created only by 'add a node' grammar operation, while bisecting an edge leads only to adjusting the tree branch lengths.

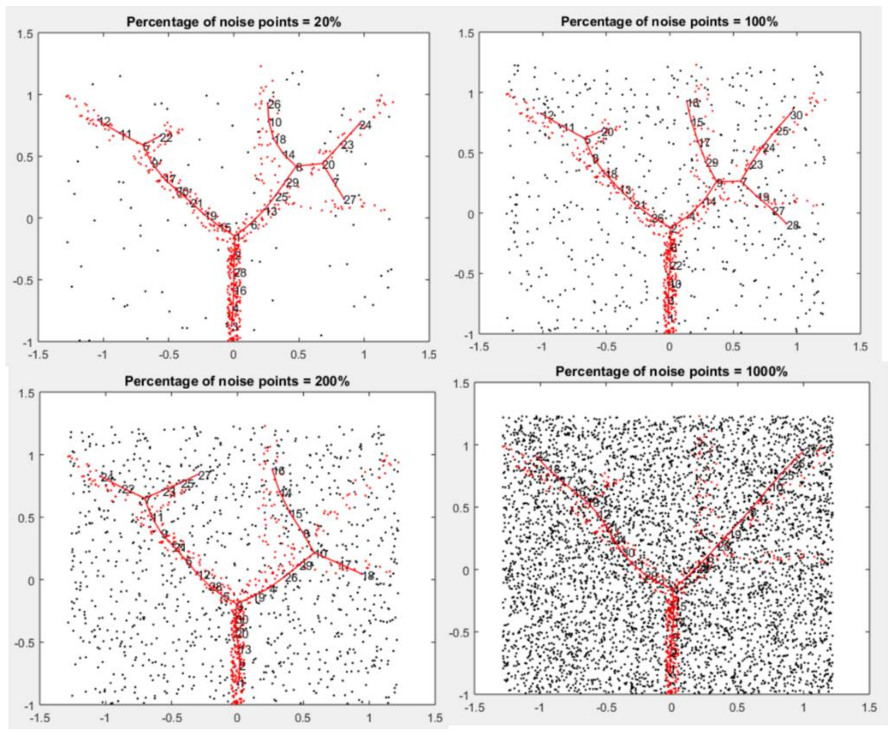
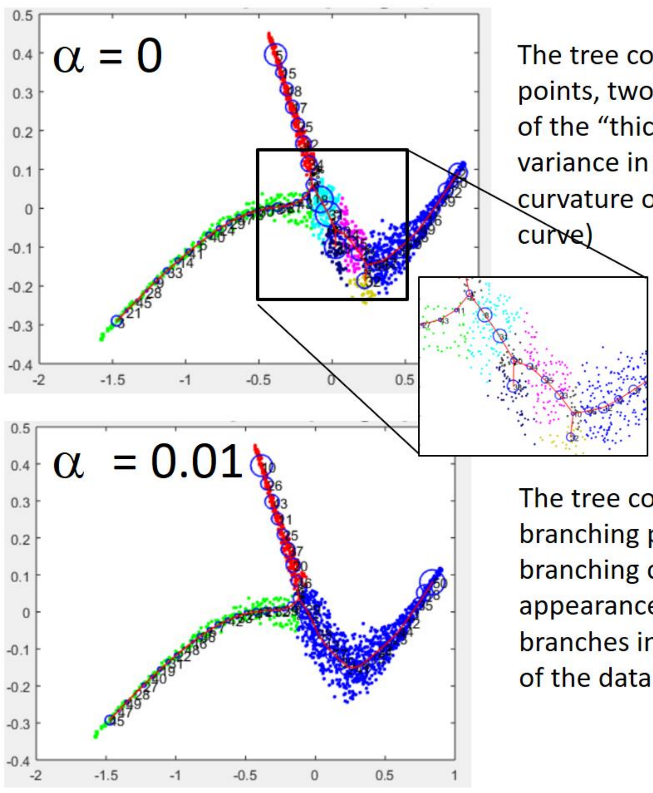


Figure RobustLocal. Example of ElPiGraph application to a branching data pattern (red points) corrupted by uniform background noise (black points). The algorithm is given unlabelled data distribution.



The tree contains three branching points, two of which appear because of the “thick turn” feature (increased variance in the area of the maximal curvature of a hypothetical principal curve)

The tree contains only one essential branching point, because branching control penalty supresses appearance of suboptimal small branches in the “thick turn” area of the data distribution

Figure BranchingControl. Using branching control penalty α (see Formula (5)) on appearance of excessive branching in the areas of data distribution where the local data dimension is significantly larger than one.

ElPiGraph bibliography

1. Gorban A, Kegl B, Wunch D, Zinovyev A. (eds.) Principal Manifolds for Data Visualisation and Dimension Reduction. 2008. Lecture Notes in Computational Science and Engeneering 58, p.340.
2. Gorban AN and Zinovyev AY. Principal Graphs and Manifolds. In Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods and Techniques (eds. Olivas E.S., Guererro J.D.M., Sober M.M., Benedito J.R.M., Lopes A.J.S.). Information Science Reference, September 4, 2009.
3. Gorban A., Sumner N., Zinovyev A. Topological grammars for data approximation. 2007. Applied Mathematics Letters 20(4), 382-386.
4. Gorban A., Mirkes E., Zinovyev A. Robust principal graphs for data approximation. Archives of Data Science 2(1):1:16, 2017.
5. Gorban A.N., Zinovyev A. 2010. Principal manifolds and graphs in practice: from molecular biology to dynamical systems. Int J Neural Syst 20(3):219-32.
6. Gorban A., Zinovyev A. Elastic Principal Graphs and Manifolds and their Practical Applications. 2005. Computing 75,359 -379.
7. Zinovyev A. and Mirkes E. Data complexity measured by principal graphs. 2013. Computers and Mathematics with Applications 65:1471-1482.
8. Gorban A, Sumner N, Zinovyev A. Beyond The Concept of Manifolds: Principal Trees, Metro Maps, and Elastic Cubic Complexes. 2008. In “Principal Manifolds for Data Visualisation and Dimension Reduction” (eds. Gorban A, Kegl B, Wunch D, Zinovyev A.), Lecture Notes in Computational Science and Engeneering 58: 223-240.
9. Gorban AN, Mirkes EM, Zinovyev A. Piece-wise quadratic approximations of arbitrary error functions for fast and robust machine learning. Neural Netw. 2016, 84:28-38.
10. Gordaliza A (1991) Best approximations to random variables based on trimming procedures. Journal of Approximation Theory 64(2):162–180