MAT 280: Harmonic Analysis on Graphs & Networks
Lecture 9: Graph Construction from Given Datasets

Naoki Saito

Department of Mathematics
University of California, Davis

October 24, 2019

Outline

1. Motivation: How to Construct a Graph from a Given Dataset
2. Simple Graph Construction Strategies
3. Optimization Strategy by Daitch-Kelner-Spielman

Outline

1. Motivation: How to Construct a Graph from a Given Dataset
2. Simple Graph Construction Strategies
3. Optimization Strategy by Daitch-Kelner-Spielman

From Datasets to Graphs

- Suppose we have deployed a set of sensors at scattered (i.e., unequally-spaced) locations and each sensor records time series (or waveform) data measuring certain properties of interest, e.g.,
  - seismometers to monitor volcanic activities;
  - acoustic sensors to detect underground tunnels;
  - biosensors to detect pathogens, . . .
- In order to analyze such data in an efficient manner not only along the time axis but also *across* the sensors (i.e., the spatial directions), we need to construct a *graph* whose vertices represent the spatial locations where the sensors are placed.
- Then an important question is how to define *edges* among these vertices to form a graph.

From Datasets to Graphs

- Suppose we have deployed a set of sensors at scattered (i.e., unequally-spaced) locations and each sensor records time series (or waveform) data measuring certain properties of interest, e.g.,
  - seismometers to monitor volcanic activities;
  - acoustic sensors to detect underground tunnels;
  - biosensors to detect pathogens, . . .
- In order to analyze such data in an efficient manner not only along the time axis but also *across* the sensors (i.e., the spatial directions), we need to construct a *graph* whose vertices represent the spatial locations where the sensors are placed.
- Then an important question is how to define *edges* among these vertices to form a graph.

From Datasets to Graphs

- Suppose we have deployed a set of sensors at scattered (i.e., unequally-spaced) locations and each sensor records time series (or waveform) data measuring certain properties of interest, e.g.,
  - seismometers to monitor volcanic activities;
  - acoustic sensors to detect underground tunnels;
  - biosensors to detect pathogens, ...
- In order to analyze such data in an efficient manner not only along the time axis but also *across* the sensors (i.e., the spatial directions), we need to construct a *graph* whose vertices represent the spatial locations where the sensors are placed.
- Then an important question is how to define *edges* among these vertices to form a graph.

From Datasets to Graphs

- Suppose we have deployed a set of sensors at scattered (i.e., unequally-spaced) locations and each sensor records time series (or waveform) data measuring certain properties of interest, e.g.,
  - seismometers to monitor volcanic activities;
  - acoustic sensors to detect underground tunnels;
  - biosensors to detect pathogens, . . .
- In order to analyze such data in an efficient manner not only along the time axis but also *across* the sensors (i.e., the spatial directions), we need to construct a *graph* whose vertices represent the spatial locations where the sensors are placed.
- Then an important question is how to define *edges* among these vertices to form a graph.

From Datasets to Graphs

- Suppose we have deployed a set of sensors at scattered (i.e., unequally-spaced) locations and each sensor records time series (or waveform) data measuring certain properties of interest, e.g.,
  - seismometers to monitor volcanic activities;
  - acoustic sensors to detect underground tunnels;
  - biosensors to detect pathogens, . . .
- In order to analyze such data in an efficient manner not only along the time axis but also *across* the sensors (i.e., the spatial directions), we need to construct a *graph* whose vertices represent the spatial locations where the sensors are placed.
- Then an important question is how to define *edges* among these vertices to form a graph.

From Datasets to Graphs

- Suppose we have deployed a set of sensors at scattered (i.e., unequally-spaced) locations and each sensor records time series (or waveform) data measuring certain properties of interest, e.g.,
  - seismometers to monitor volcanic activities;
  - acoustic sensors to detect underground tunnels;
  - biosensors to detect pathogens, . . .
- In order to analyze such data in an efficient manner not only along the time axis but also *across* the sensors (i.e., the spatial directions), we need to construct a *graph* whose vertices represent the spatial locations where the sensors are placed.
- Then an important question is how to define *edges* among these vertices to form a graph.

From Datasets to Graphs . . .

Unless the restrictive and predefined cases (e.g., each sensor/vertex is physically forced to connect to only a handful of its neighbors), we need to answer the following questions:

- Should we connect each vertex to every other vertex to make a *complete* graph?
- Or should we create a *sparse* graph for efficiency without deteriorating the performance of the task at hand (e.g., detection, classification, regression, missing data recovery, etc.)?
- What *weight* should we assign to each edge?

These questions are also important in completely different and more general scenarios where each vertex represents not the sensor location but simply a *vector* in $\mathbb{R}^d$ (e.g., an image patch for denoising and feature extraction, etc.)

From Datasets to Graphs . . .

Unless the restrictive and predefined cases (e.g., each sensor/vertex is physically forced to connect to only a handful of its neighbors), we need to answer the following questions:

- Should we connect each vertex to every other vertex to make a *complete* graph?
- Or should we create a *sparse* graph for efficiency without deteriorating the performance of the task at hand (e.g., detection, classification, regression, missing data recovery, etc.)?
- What *weight* should we assign to each edge?

These questions are also important in completely different and more general scenarios where each vertex represents not the sensor location but simply a *vector* in $\mathbb{R}^d$ (e.g., an image patch for denoising and feature extraction, etc.)

From Datasets to Graphs . . .

Unless the restrictive and predefined cases (e.g., each sensor/vertex is physically forced to connect to only a handful of its neighbors), we need to answer the following questions:

- Should we connect each vertex to every other vertex to make a *complete* graph?
- Or should we create a *sparse* graph for efficiency without deteriorating the performance of the task at hand (e.g., detection, classification, regression, missing data recovery, etc.)?
- What *weight* should we assign to each edge?

These questions are also important in completely different and more general scenarios where each vertex represents not the sensor location but simply a *vector* in $\mathbb{R}^d$ (e.g., an image patch for denoising and feature extraction, etc.)

From Datasets to Graphs ...

Unless the restrictive and predefined cases (e.g., each sensor/vertex is physically forced to connect to only a handful of its neighbors), we need to answer the following questions:

- Should we connect each vertex to every other vertex to make a *complete* graph?
- Or should we create a *sparse* graph for efficiency without deteriorating the performance of the task at hand (e.g., detection, classification, regression, missing data recovery, etc.)?
- What *weight* should we assign to each edge?

These questions are also important in completely different and more general scenarios where each vertex represents not the sensor location but simply a *vector* in $\mathbb{R}^d$ (e.g., an image patch for denoising and feature extraction, etc.)

From Datasets to Graphs ...

Unless the restrictive and predefined cases (e.g., each sensor/vertex is physically forced to connect to only a handful of its neighbors), we need to answer the following questions:

- Should we connect each vertex to every other vertex to make a *complete* graph?
- Or should we create a *sparse* graph for efficiency without deteriorating the performance of the task at hand (e.g., detection, classification, regression, missing data recovery, etc.)?
- What *weight* should we assign to each edge?

These questions are also important in completely different and more general scenarios where each vertex represents not the sensor location but simply a *vector* in $\mathbb{R}^d$ (e.g., an image patch for denoising and feature extraction, etc.)

Outline

1 Motivation: How to Construct a Graph from a Given Dataset

2 Simple Graph Construction Strategies

3 Optimization Strategy by Daitch-Kelner-Spielman

Criteria for Good Graphs

- What is a *good* graph?
- Obviously, this question depends on the task at hand (e.g., classification, regression, missing data recovery, . . .).
- Yet, the "goodness" of a graph should be measured in the following three criteria:

C1 Computational efficiency for constructing a graph from given data;

C2 Computational efficiency for processing data on the constructed graph;

C3 Performance of the tasks at hand using that graph.

Criteria for Good Graphs

- What is a *good* graph?
- Obviously, this question depends on the task at hand (e.g., classification, regression, missing data recovery, ...).
- Yet, the "goodness" of a graph should be measured in the following three criteria:

C1 Computational efficiency for constructing a graph from given data;

C2 Computational efficiency for processing data on the constructed graph;

C3 Performance of the tasks at hand using that graph.

Criteria for Good Graphs

- What is a *good* graph?
- Obviously, this question depends on the task at hand (e.g., classification, regression, missing data recovery, . . .).
- Yet, the "goodness" of a graph should be measured in the following three criteria:

C1 Computational efficiency for constructing a graph from given data;

C2 Computational efficiency for processing data on the constructed graph;

C3 Performance of the tasks at hand using that graph.

Criteria for Good Graphs

- What is a *good* graph?
- Obviously, this question depends on the task at hand (e.g., classification, regression, missing data recovery, . . .).
- Yet, the "goodness" of a graph should be measured in the following three criteria:

C1 Computational efficiency for constructing a graph from given data;

C2 Computational efficiency for processing data on the constructed graph;

C3 Performance of the tasks at hand using that graph.

Criteria for Good Graphs

- What is a *good* graph?
- Obviously, this question depends on the task at hand (e.g., classification, regression, missing data recovery, . . .).
- Yet, the "goodness" of a graph should be measured in the following three criteria:

C1 Computational efficiency for constructing a graph from given data;

C2 Computational efficiency for processing data on the constructed graph;

C3 Performance of the tasks at hand using that graph.

Criteria for Good Graphs

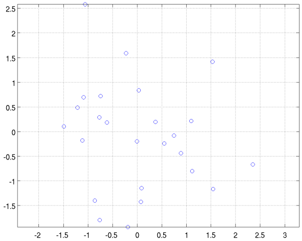
- What is a *good* graph?
- Obviously, this question depends on the task at hand (e.g., classification, regression, missing data recovery, . . .).
- Yet, the "goodness" of a graph should be measured in the following three criteria:

C1 Computational efficiency for constructing a graph from given data;

C2 Computational efficiency for processing data on the constructed graph;

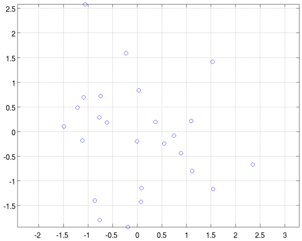
C3 Performance of the tasks at hand using that graph.

Three Possibilities

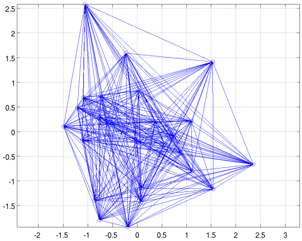


(a) Sensor locations

Three Possibilities

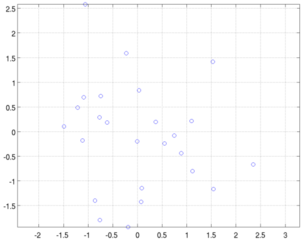


(a) Sensor locations

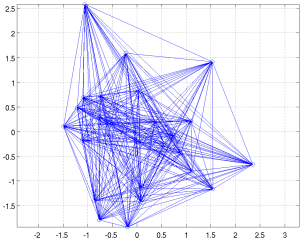


(b) Complete graph

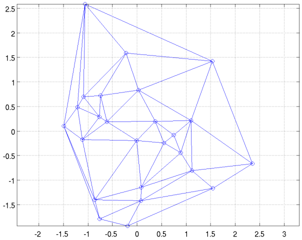
Three Possibilities



(a) Sensor locations

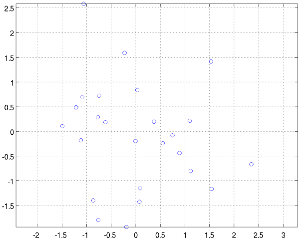


(b) Complete graph

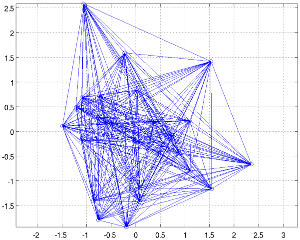


(c) Delaunay graph

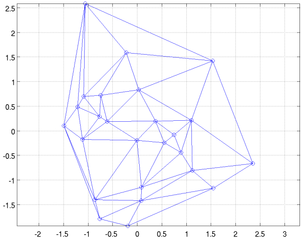
Three Possibilities



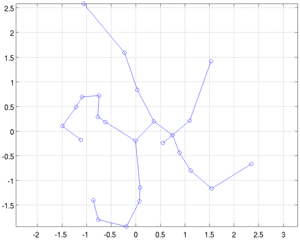
(a) Sensor locations



(b) Complete graph



(c) Delaunay graph



(d) Minimum spanning tree

Complete Graphs

- Construct a *complete* graph $K(V) = K_n$ by mutually connecting all the vertices $v_1, \ldots, v_n$.
- Often the Gaussian weights are used for the edge weights, i.e., for $w_{ij} = \exp(-\mathrm{dist}(v_i, v_j)^2/\epsilon^2)$ where $\mathrm{dist}(\cdot,\cdot)$ is an appropriate distance function (e.g., ℓ^2-distance), and ϵ is an appropriate scale parameter, which is often difficult to choose (more about it in the next lecture).
- This is easy and good in the sense of Criterion 1.
- Hence, many people in fact have been constructing and using this strategy more or less mindlessly.
- The number of its edges, however, is of course quite large, i.e., $|E(K_n)| = n(n-1)/2$, which may hinder it from being good in Criterion 2.

Complete Graphs

- Construct a *complete* graph $K(V) = K_n$ by mutually connecting all the vertices $v_1, \ldots, v_n$.
- Often the Gaussian weights are used for the edge weights, i.e., for $w_{ij} = \exp(-\mathrm{dist}(v_i, v_j)^2/\epsilon^2)$ where $\mathrm{dist}(\cdot,\cdot)$ is an appropriate distance function (e.g., ℓ^2-distance), and ϵ is an appropriate scale parameter, which is often difficult to choose (more about it in the next lecture).
- This is easy and good in the sense of Criterion 1.
- Hence, many people in fact have been constructing and using this strategy more or less mindlessly.
- The number of its edges, however, is of course quite large, i.e., $|E(K_n)| = n(n-1)/2$, which may hinder it from being good in Criterion 2.

Complete Graphs

- Construct a *complete* graph $K(V) = K_n$ by mutually connecting all the vertices $v_1, \ldots, v_n$.
- Often the Gaussian weights are used for the edge weights, i.e., for $w_{ij} = \exp(-\operatorname{dist}(v_i, v_j)^2/\epsilon^2)$ where $\operatorname{dist}(\cdot,\cdot)$ is an appropriate distance function (e.g., ℓ^2-distance), and ϵ is an appropriate scale parameter, which is often difficult to choose (more about it in the next lecture).
- This is easy and good in the sense of Criterion 1.
- Hence, many people in fact have been constructing and using this strategy more or less mindlessly.
- The number of its edges, however, is of course quite large, i.e., $|E(K_n)| = n(n-1)/2$, which may hinder it from being good in Criterion 2.

Complete Graphs

- Construct a *complete* graph $K(V) = K_n$ by mutually connecting all the vertices $v_1, \ldots, v_n$.
- Often the Gaussian weights are used for the edge weights, i.e., for $w_{ij} = \exp(-\operatorname{dist}(v_i, v_j)^2/\epsilon^2)$ where $\operatorname{dist}(\cdot,\cdot)$ is an appropriate distance function (e.g., ℓ^2-distance), and ϵ is an appropriate scale parameter, which is often difficult to choose (more about it in the next lecture).
- This is easy and good in the sense of Criterion 1.
- Hence, many people in fact have been constructing and using this strategy more or less mindlessly.
- The number of its edges, however, is of course quite large, i.e., $|E(K_n)| = n(n-1)/2$, which may hinder it from being good in Criterion 2.

Complete Graphs

- Construct a *complete* graph $K(V) = K_n$ by mutually connecting all the vertices $v_1, \ldots, v_n$.
- Often the Gaussian weights are used for the edge weights, i.e., for $w_{ij} = \exp(-\operatorname{dist}(v_i, v_j)^2/\epsilon^2)$ where $\operatorname{dist}(\cdot,\cdot)$ is an appropriate distance function (e.g., ℓ^2-distance), and ϵ is an appropriate scale parameter, which is often difficult to choose (more about it in the next lecture).
- This is easy and good in the sense of Criterion 1.
- Hence, many people in fact have been constructing and using this strategy more or less mindlessly.
- The number of its edges, however, is of course quite large, i.e., $|E(K_n)| = n(n-1)/2$, which may hinder it from being good in Criterion 2.

Delaunay Graphs

- How to *sparsify* a complete graph to improve Criterion 2 while keeping Criterion 1 in mind?
- One of the possibilities may be the so-called *Delaunay graph*.
- If $v_i \in \mathbb{R}^2$, $i = 1, \dots, n$, then the *Delaunay triangulation* $DT(V)$ for V is a triangulation such that no vertex in V is inside the circumcircle of any triangle in $DT(V)$.

Delaunay Graphs

- How to *sparsify* a complete graph to improve Criterion 2 while keeping Criterion 1 in mind?
- One of the possibilities may be the so-called *Delaunay graph*.
- If $v_i \in \mathbb{R}^2$, $i = 1, \dots, n$, then the *Delaunay triangulation* $DT(V)$ for V is a triangulation such that no vertex in V is inside the circumcircle of any triangle in $DT(V)$.

Delaunay Graphs

- How to *sparsify* a complete graph to improve Criterion 2 while keeping Criterion 1 in mind?
- One of the possibilities may be the so-called *Delaunay graph*.
- If $v_i \in \mathbb{R}^2$, $i = 1, \ldots, n$, then the *Delaunay triangulation* $DT(V)$ for V is a triangulation such that no vertex in V is inside the circumcircle of any triangle in $DT(V)$.

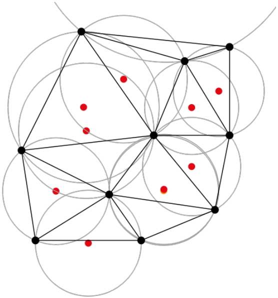


Figure: From Wikipedia

Delaunay Graphs . . .

- $DT(V)$ maximizes the minimum angle of all the angles of the triangles in the triangulations $\Longrightarrow$ tends to avoid extremely skinny triangles
- Moreover, in $\mathbb{R}^2$, there is a fast algorithm to construct $DT(V)$ with $O(n \log n)$ cost.
- A graph representing such Delaunay triangulation of V is called the *Delaunay graph* of V, and denoted by $DG(V)$.
- By considering circumscribed spheres, the notion of Delaunay triangulation can extend to three and higher dimensions.
- The computational cost to construct $DG(V)$ for higher dimension, however, can be high: $O(n \log n + n^{\lceil d/2 \rceil})$ if $V \subset \mathbb{R}^d$.
- Hence, the Delaunay graph may be useful if the vertices represent the physical sensor locations/coordinates in $\mathbb{R}^2$ or at most $\mathbb{R}^3$.
- In more general situations where each vertex directly represent a high dimensional vector in $\mathbb{R}^d$ with $d > 3$, then this may not be a good approach in terms of Criterion 1.

Delaunay Graphs . . .

- $DT(V)$ maximizes the minimum angle of all the angles of the triangles in the triangulations $\Longrightarrow$ tends to avoid extremely skinny triangles
- Moreover, in $\mathbb{R}^2$, there is a fast algorithm to construct $DT(V)$ with $O(n \log n)$ cost.
- A graph representing such Delaunay triangulation of V is called the *Delaunay graph* of V, and denoted by $DG(V)$.
- By considering circumscribed spheres, the notion of Delaunay triangulation can extend to three and higher dimensions.
- The computational cost to construct $DG(V)$ for higher dimension, however, can be high: $O(n \log n + n^{\lceil d/2 \rceil})$ if $V \subset \mathbb{R}^d$.
- Hence, the Delaunay graph may be useful if the vertices represent the physical sensor locations/coordinates in $\mathbb{R}^2$ or at most $\mathbb{R}^3$.
- In more general situations where each vertex directly represent a high dimensional vector in $\mathbb{R}^d$ with $d > 3$, then this may not be a good approach in terms of Criterion 1.

Delaunay Graphs ...

- $DT(V)$ maximizes the minimum angle of all the angles of the triangles in the triangulations $\Longrightarrow$ tends to avoid extremely skinny triangles
- Moreover, in $\mathbb{R}^2$, there is a fast algorithm to construct $DT(V)$ with $O(n \log n)$ cost.
- A graph representing such Delaunay triangulation of V is called the *Delaunay graph* of V, and denoted by $DG(V)$.
- By considering circumscribed spheres, the notion of Delaunay triangulation can extend to three and higher dimensions.
- The computational cost to construct $DG(V)$ for higher dimension, however, can be high: $O(n \log n + n^{\lceil d/2 \rceil})$ if $V \subset \mathbb{R}^d$.
- Hence, the Delaunay graph may be useful if the vertices represent the physical sensor locations/coordinates in $\mathbb{R}^2$ or at most $\mathbb{R}^3$.
- In more general situations where each vertex directly represent a high dimensional vector in $\mathbb{R}^d$ with $d > 3$, then this may not be a good approach in terms of Criterion 1.

Delaunay Graphs . . .

- $DT(V)$ maximizes the minimum angle of all the angles of the triangles in the triangulations $\Longrightarrow$ tends to avoid extremely skinny triangles
- Moreover, in $\mathbb{R}^2$, there is a fast algorithm to construct $DT(V)$ with $O(n\log n)$ cost.
- A graph representing such Delaunay triangulation of V is called the *Delaunay graph* of V, and denoted by $DG(V)$.
- By considering circumscribed spheres, the notion of Delaunay triangulation can extend to three and higher dimensions.
- The computational cost to construct $DG(V)$ for higher dimension, however, can be high: $O(n\log n + n^{\lceil d/2 \rceil})$ if $V \subset \mathbb{R}^d$.
- Hence, the Delaunay graph may be useful if the vertices represent the physical sensor locations/coordinates in $\mathbb{R}^2$ or at most $\mathbb{R}^3$.
- In more general situations where each vertex directly represent a high dimensional vector in $\mathbb{R}^d$ with $d > 3$, then this may not be a good approach in terms of Criterion 1.

Delaunay Graphs . . .

- $DT(V)$ maximizes the minimum angle of all the angles of the triangles in the triangulations $\Longrightarrow$ tends to avoid extremely skinny triangles
- Moreover, in $\mathbb{R}^2$, there is a fast algorithm to construct $DT(V)$ with $O(n \log n)$ cost.
- A graph representing such Delaunay triangulation of V is called the *Delaunay graph* of V, and denoted by $DG(V)$.
- By considering circumscribed spheres, the notion of Delaunay triangulation can extend to three and higher dimensions.
- The computational cost to construct $DG(V)$ for higher dimension, however, can be high: $O(n \log n + n^{\lceil d/2 \rceil})$ if $V \subset \mathbb{R}^d$.
- Hence, the Delaunay graph may be useful if the vertices represent the physical sensor locations/coordinates in $\mathbb{R}^2$ or at most $\mathbb{R}^3$.
- In more general situations where each vertex directly represent a high dimensional vector in $\mathbb{R}^d$ with $d > 3$, then this may not be a good approach in terms of Criterion 1.

Delaunay Graphs . . .

- $DT(V)$ maximizes the minimum angle of all the angles of the triangles in the triangulations $\Longrightarrow$ tends to avoid extremely skinny triangles
- Moreover, in $\mathbb{R}^2$, there is a fast algorithm to construct $DT(V)$ with $O(n \log n)$ cost.
- A graph representing such Delaunay triangulation of V is called the *Delaunay graph* of V, and denoted by $DG(V)$.
- By considering circumscribed spheres, the notion of Delaunay triangulation can extend to three and higher dimensions.
- The computational cost to construct $DG(V)$ for higher dimension, however, can be high: $O(n \log n + n^{\lceil d/2 \rceil})$ if $V \subset \mathbb{R}^d$.
- Hence, the Delaunay graph may be useful if the vertices represent the physical sensor locations/coordinates in $\mathbb{R}^2$ or at most $\mathbb{R}^3$.
- In more general situations where each vertex directly represent a high dimensional vector in $\mathbb{R}^d$ with $d > 3$, then this may not be a good approach in terms of Criterion 1.

Delaunay Graphs . . .

- $DT(V)$ maximizes the minimum angle of all the angles of the triangles in the triangulations $\Longrightarrow$ tends to avoid extremely skinny triangles
- Moreover, in $\mathbb{R}^2$, there is a fast algorithm to construct $DT(V)$ with $O(n \log n)$ cost.
- A graph representing such Delaunay triangulation of V is called the *Delaunay graph* of V, and denoted by $DG(V)$.
- By considering circumscribed spheres, the notion of Delaunay triangulation can extend to three and higher dimensions.
- The computational cost to construct $DG(V)$ for higher dimension, however, can be high: $O(n \log n + n^{\lceil d/2 \rceil})$ if $V \subset \mathbb{R}^d$.
- Hence, the Delaunay graph may be useful if the vertices represent the physical sensor locations/coordinates in $\mathbb{R}^2$ or at most $\mathbb{R}^3$.
- In more general situations where each vertex directly represent a high dimensional vector in $\mathbb{R}^d$ with $d > 3$, then this may not be a good approach in terms of Criterion 1.

Minimum Spanning Trees

- Can construct the so-called *minimum spanning tree* from either a complete graph or a Delaunay graph.
- A *spanning tree* of a given connected graph $G(V, E)$ is a subgraph of G that is a tree and connects all the vertices in V together.
- In general, G can have many different spanning trees, and the *minimum spanning tree* $MST(G)$ of G is a spanning tree whose total edge weights (i.e., the sum of the edge weights in that tree) are less than or equal to those of every other spanning tree.

Minimum Spanning Trees

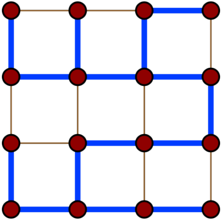
- Can construct the so-called *minimum spanning tree* from either a complete graph or a Delaunay graph.
- A *spanning tree* of a given connected graph $G(V,E)$ is a subgraph of G that is a tree and connects all the vertices in V together.
- In general, G can have many different spanning trees, and the *minimum spanning tree* $MST(G)$ of G is a spanning tree whose total edge weights (i.e., the sum of the edge weights in that tree) are less than or equal to those of every other spanning tree.

Minimum Spanning Trees

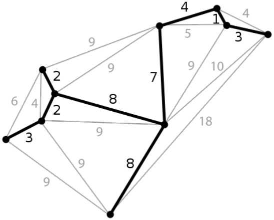
- Can construct the so-called *minimum spanning tree* from either a complete graph or a Delaunay graph.
- A *spanning tree* of a given connected graph $G(V, E)$ is a subgraph of G that is a tree and connects all the vertices in V together.
- In general, G can have many different spanning trees, and the *minimum spanning tree* $MST(G)$ of G is a spanning tree whose total edge weights (i.e., the sum of the edge weights in that tree) are less than or equal to those of every other spanning tree.

Minimum Spanning Trees

- Can construct the so-called *minimum spanning tree* from either a complete graph or a Delaunay graph.
- A *spanning tree* of a given connected graph $G(V, E)$ is a subgraph of G that is a tree and connects all the vertices in V together.
- In general, G can have many different spanning trees, and the *minimum spanning tree* $MST(G)$ of G is a spanning tree whose total edge weights (i.e., the sum of the edge weights in that tree) are less than or equal to those of every other spanning tree.



(a) MST(a lattice)



(b) MST(a weighted graph)

Figure: From Wikipedia

Minimum Spanning Trees

- It is well known that

$$MST(K(V)) \subset DG(V) \subset K(V)$$

- $MST(G)$ is the sparsest graph that connects all the vertices without redundancy; but it may not be unique for a given G.
- Computing $MST(G)$ for a given $G = G(V, E)$ is good in Criterion 1: there exists fast algorithms with the cost $O(|E(G)| \log n)$.
- This does not depends on the dimension of the vectors d at the vertices. Hence, $MST(G)$ is easier to compute than $DG(V)$ in general.
- For the details of the computational algorithms for MST as well as its history, see the references provided at the course reference webpage.

Minimum Spanning Trees

- It is well known that

$$MST(K(V)) \subset DG(V) \subset K(V)$$

- $MST(G)$ is the sparsest graph that connects all the vertices without redundancy; but it may not be unique for a given G.
- Computing $MST(G)$ for a given $G = G(V,E)$ is good in Criterion 1: there exists fast algorithms with the cost $O(|E(G)| \log n)$.
- This does not depends on the dimension of the vectors d at the vertices. Hence, $MST(G)$ is easier to compute than $DG(V)$ in general.
- For the details of the computational algorithms for MST as well as its history, see the references provided at the course reference webpage.

Minimum Spanning Trees

- It is well known that

$$MST(K(V)) \subset DG(V) \subset K(V)$$

- $MST(G)$ is the sparsest graph that connects all the vertices without redundancy; but it may not be unique for a given G.
- Computing $MST(G)$ for a given $G = G(V,E)$ is good in Criterion 1: there exists fast algorithms with the cost $O(|E(G)| \log n)$.
- This does not depends on the dimension of the vectors d at the vertices. Hence, $MST(G)$ is easier to compute than $DG(V)$ in general.
- For the details of the computational algorithms for MST as well as its history, see the references provided at the course reference webpage.

Minimum Spanning Trees

- It is well known that

$$MST(K(V)) \subset DG(V) \subset K(V)$$

- $MST(G)$ is the sparsest graph that connects all the vertices without redundancy; but it may not be unique for a given G.
- Computing $MST(G)$ for a given $G = G(V,E)$ is good in Criterion 1: there exists fast algorithms with the cost $O(|E(G)| \log n)$.
- This does not depends on the dimension of the vectors d at the vertices. Hence, $MST(G)$ is easier to compute than $DG(V)$ in general.
- For the details of the computational algorithms for MST as well as its history, see the references provided at the course reference webpage.

Minimum Spanning Trees

- It is well known that

$$MST(K(V)) \subset DG(V) \subset K(V)$$

- $MST(G)$ is the sparsest graph that connects all the vertices without redundancy; but it may not be unique for a given G.
- Computing $MST(G)$ for a given $G = G(V,E)$ is good in Criterion 1: there exists fast algorithms with the cost $O(|E(G)| \log n)$.
- This does not depends on the dimension of the vectors d at the vertices. Hence, $MST(G)$ is easier to compute than $DG(V)$ in general.
- For the details of the computational algorithms for MST as well as its history, see the references provided at the course reference webpage.

k-Nearest Neighbor Graphs

- Yet another popular approach to construct a graph from a given dataset is the so-called *k-nearest neighbor graph*.
- Its construction is simple: connect v_i with v_j if v_j is among the k-nearest neighbors of v_i.
- This definition leads to a directed graph. To make it undirected, there are two approaches:
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *or* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called *the k-nearest neighbor graph*.
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *and* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called the *mutual k-nearest neighbor graph*.
- Of course, the important questions are:
  - what *distance* among the vertices (or vectors) should be used?
  - what value of k should be used?

k-Nearest Neighbor Graphs

- Yet another popular approach to construct a graph from a given dataset is the so-called *k-nearest neighbor graph*.
- Its construction is simple: connect v_i with v_j if v_j is among the k-nearest neighbors of v_i.
- This definition leads to a directed graph. To make it undirected, there are two approaches:
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *or* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called *the k-nearest neighbor graph*.
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *and* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called the *mutual k-nearest neighbor graph*.
- Of course, the important questions are:
  - what *distance* among the vertices (or vectors) should be used?
  - what value of k should be used?

k-Nearest Neighbor Graphs

- Yet another popular approach to construct a graph from a given dataset is the so-called *k-nearest neighbor graph*.
- Its construction is simple: connect v_i with v_j if v_j is among the k-nearest neighbors of v_i.
- This definition leads to a directed graph. To make it undirected, there are two approaches:
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *or* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called *the k-nearest neighbor graph*.
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *and* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called the *mutual k-nearest neighbor graph*.
- Of course, the important questions are:
  - what *distance* among the vertices (or vectors) should be used?
  - what value of k should be used?

k-Nearest Neighbor Graphs

- Yet another popular approach to construct a graph from a given dataset is the so-called *k-nearest neighbor graph*.
- Its construction is simple: connect v_i with v_j if v_j is among the k-nearest neighbors of v_i.
- This definition leads to a directed graph. To make it undirected, there are two approaches:
    - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *or* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called *the k-nearest neighbor graph*.
    - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *and* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called the *mutual k-nearest neighbor graph*.
- Of course, the important questions are:
    - what *distance* among the vertices (or vectors) should be used?
    - what value of k should be used?

k-Nearest Neighbor Graphs

- Yet another popular approach to construct a graph from a given dataset is the so-called *k-nearest neighbor graph*.
- Its construction is simple: connect v_i with v_j if v_j is among the k-nearest neighbors of v_i.
- This definition leads to a directed graph. To make it undirected, there are two approaches:
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *or* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called *the k-nearest neighbor graph*.
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *and* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called the *mutual k-nearest neighbor graph*.
- Of course, the important questions are:
  - what *distance* among the vertices (or vectors) should be used?
  - what value of k should be used?

k-Nearest Neighbor Graphs

- Yet another popular approach to construct a graph from a given dataset is the so-called *k-nearest neighbor graph*.
- Its construction is simple: connect v_i with v_j if v_j is among the k-nearest neighbors of v_i.
- This definition leads to a directed graph. To make it undirected, there are two approaches:
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *or* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called *the k-nearest neighbor graph*.
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *and* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called the *mutual k-nearest neighbor graph*.
- Of course, the important questions are:
  - what *distance* among the vertices (or vectors) should be used?
  - what value of k should be used?

k-Nearest Neighbor Graphs

- Yet another popular approach to construct a graph from a given dataset is the so-called *k-nearest neighbor graph*.
- Its construction is simple: connect v_i with v_j if v_j is among the k-nearest neighbors of v_i.
- This definition leads to a directed graph. To make it undirected, there are two approaches:
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *or* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called *the k-nearest neighbor graph*.
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *and* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called the *mutual k-nearest neighbor graph*.
- Of course, the important questions are:
  - what *distance* among the vertices (or vectors) should be used?
  - what value of *k* should be used?

k-Nearest Neighbor Graphs

- Yet another popular approach to construct a graph from a given dataset is the so-called *k-nearest neighbor graph*.
- Its construction is simple: connect v_i with v_j if v_j is among the k-nearest neighbors of v_i.
- This definition leads to a directed graph. To make it undirected, there are two approaches:
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *or* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called *the k-nearest neighbor graph*.
  - Create an edge $e = (v_i, v_j)$ if v_i is among the k-nearest neighbors of v_j *and* if v_j is among the k-nearest neighbors of v_i. The resulting graph is usually called the *mutual k-nearest neighbor graph*.
- Of course, the important questions are:
  - what *distance* among the vertices (or vectors) should be used?
  - what value of *k* should be used?

The ε-Neighborhood Graph

- This graph is created by connecting all vertices whose pairwise distances are smaller than $\varepsilon > 0$.
- Since the distances between all connected vertices are roughly of the same scale (at most ε), weighting the edges would not incorporate more information about the data to the graph.
- Hence, the ε-neighborhood graph is usually viewed as an *unweighted* graph.
- Again the important questions to ask are the *distance measure* between vertices and the value of ε.

The ε-Neighborhood Graph

- This graph is created by connecting all vertices whose pairwise distances are smaller than $\varepsilon > 0$.
- Since the distances between all connected vertices are roughly of the same scale (at most ε), weighting the edges would not incorporate more information about the data to the graph.
- Hence, the ε-neighborhood graph is usually viewed as an *unweighted* graph.
- Again the important questions to ask are the *distance measure* between vertices and the value of ε.

The ε-Neighborhood Graph

- This graph is created by connecting all vertices whose pairwise distances are smaller than $\varepsilon > 0$.
- Since the distances between all connected vertices are roughly of the same scale (at most ε), weighting the edges would not incorporate more information about the data to the graph.
- Hence, the ε-neighborhood graph is usually viewed as an *unweighted* graph.
- Again the important questions to ask are the *distance measure* between vertices and the value of ε.

The ε-Neighborhood Graph

- This graph is created by connecting all vertices whose pairwise distances are smaller than $\varepsilon > 0$.
- Since the distances between all connected vertices are roughly of the same scale (at most ε), weighting the edges would not incorporate more information about the data to the graph.
- Hence, the ε-neighborhood graph is usually viewed as an *unweighted* graph.
- Again the important questions to ask are the *distance measure* between vertices and the value of ε.

Difficulty of Assessing Criterion 3

- So far, we have not discussed Criterion 3, i.e., the dependency of the task performance on constructed graphs.
- Unfortunately, there is no general theory to automatically construct a graph to optimize a given task.
- We need to examine and compare the performance in each case.
- This also depends on what *distance* (or *weight*) we should assign for each edge, which will be discuss in the next lecture.

Difficulty of Assessing Criterion 3

- So far, we have not discussed Criterion 3, i.e., the dependency of the task performance on constructed graphs.
- Unfortunately, there is no general theory to automatically construct a graph to optimize a given task.
- We need to examine and compare the performance in each case.
- This also depends on what *distance* (or *weight*) we should assign for each edge, which will be discuss in the next lecture.

Difficulty of Assessing Criterion 3

- So far, we have not discussed Criterion 3, i.e., the dependency of the task performance on constructed graphs.
- Unfortunately, there is no general theory to automatically construct a graph to optimize a given task.
- We need to examine and compare the performance in each case.
- This also depends on what *distance* (or *weight*) we should assign for each edge, which will be discuss in the next lecture.

Difficulty of Assessing Criterion 3

- So far, we have not discussed Criterion 3, i.e., the dependency of the task performance on constructed graphs.
- Unfortunately, there is no general theory to automatically construct a graph to optimize a given task.
- We need to examine and compare the performance in each case.
- This also depends on what *distance* (or *weight*) we should assign for each edge, which will be discuss in the next lecture.

Outline

1. Motivation: How to Construct a Graph from a Given Dataset
2. Simple Graph Construction Strategies
3. Optimization Strategy by Daitch-Kelner-Spielman

The Daitch-Kelner-Spielman Construction

- Given a collection of vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$, we want to fit a good, weighted, and undirected graph to them.
- Viewing these vectors as vertices in a graph, it boils down to the following question: how to determine the weight $a_{ij} \geq 0$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$?
- No self-loop is allowed, i.e., $a_{ii} = 0$.
- Let $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix.
- DKS proposed to find the weighted adjacency matrix $A \in \mathbb{R}^{n \times n}_{\geq 0}$ and $A^{\mathsf{T}} = A$ such that

$$\min_{A \in \mathbb{R}^{n \times n}_{\geq 0};\, A^{\mathsf{T}} = A} \left\| L X^{\mathsf{T}} \right\|_F^2 = \min_{a_{ij} \geq 0} \sum_{i=1}^{n} \left\| \sum_{j=1}^{n} a_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_2^2 .$$

- The above objective function looks quite natural since a_{ij} becomes small if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are far apart.

The Daitch-Kelner-Spielman Construction

- Given a collection of vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$, we want to fit a good, weighted, and undirected graph to them.
- Viewing these vectors as vertices in a graph, it boils down to the following question: how to determine the weight $a_{ij} \geq 0$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$?
- No self-loop is allowed, i.e., $a_{ii} = 0$.
- Let $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix.
- DKS proposed to find the weighted adjacency matrix $A \in \mathbb{R}^{n \times n}_{\geq 0}$ and $A^{\mathsf{T}} = A$ such that

$$\min_{A \in \mathbb{R}^{n \times n}_{\geq 0}; A^{\mathsf{T}} = A} \left\| L X^{\mathsf{T}} \right\|_F^2 = \min_{a_{ij} \geq 0} \sum_{i=1}^{n} \left\| \sum_{j=1}^{n} a_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_2^2 .$$

- The above objective function looks quite natural since a_{ij} becomes small if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are far apart.

The Daitch-Kelner-Spielman Construction

- Given a collection of vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$, we want to fit a good, weighted, and undirected graph to them.
- Viewing these vectors as vertices in a graph, it boils down to the following question: how to determine the weight $a_{ij} \geq 0$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$?
- No self-loop is allowed, i.e., $a_{ii} = 0$.
- Let $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix.
- DKS proposed to find the weighted adjacency matrix $A \in \mathbb{R}^{n \times n}_{\geq 0}$ and $A^{\mathsf{T}} = A$ such that

$$\min_{A \in \mathbb{R}^{n \times n}_{\geq 0};\, A^{\mathsf{T}} = A} \left\| L X^{\mathsf{T}} \right\|_F^2 = \min_{a_{ij} \geq 0} \sum_{i=1}^{n} \left\| \sum_{j=1}^{n} a_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_2^2 .$$

- The above objective function looks quite natural since a_{ij} becomes small if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are far apart.

The Daitch-Kelner-Spielman Construction

- Given a collection of vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$, we want to fit a good, weighted, and undirected graph to them.
- Viewing these vectors as vertices in a graph, it boils down to the following question: how to determine the weight $a_{ij} \geq 0$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$?
- No self-loop is allowed, i.e., $a_{ii} = 0$.
- Let $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix.
- DKS proposed to find the weighted adjacency matrix $A \in \mathbb{R}_{\geq 0}^{n \times n}$ and $A^{\mathsf{T}} = A$ such that

$$\min_{A \in \mathbb{R}_{\geq 0}^{n \times n}; A^{\mathsf{T}} = A} \left\| L X^{\mathsf{T}} \right\|_F^2 = \min_{a_{ij} \geq 0} \sum_{i=1}^{n} \left\| \sum_{j=1}^{n} a_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_2^2 .$$

- The above objective function looks quite natural since a_{ij} becomes small if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are far apart.

The Daitch-Kelner-Spielman Construction

- Given a collection of vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$, we want to fit a good, weighted, and undirected graph to them.
- Viewing these vectors as vertices in a graph, it boils down to the following question: how to determine the weight $a_{ij} \geq 0$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$?
- No self-loop is allowed, i.e., $a_{ii} = 0$.
- Let $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix.
- DKS proposed to find the weighted adjacency matrix $A \in \mathbb{R}^{n \times n}_{\geq 0}$ and $A^{\mathsf{T}} = A$ such that

$$\min_{A \in \mathbb{R}^{n \times n}_{\geq 0}; A^{\mathsf{T}} = A} \left\| L X^{\mathsf{T}} \right\|_F^2 = \min_{a_{ij} \geq 0} \sum_{i=1}^{n} \left\| \sum_{j=1}^{n} a_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_2^2 .$$

- The above objective function looks quite natural since a_{ij} becomes small if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are far apart.

The Daitch-Kelner-Spielman Construction

- Given a collection of vectors $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$, we want to fit a good, weighted, and undirected graph to them.
- Viewing these vectors as vertices in a graph, it boils down to the following question: how to determine the weight $a_{ij} \geq 0$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$?
- No self-loop is allowed, i.e., $a_{ii} = 0$.
- Let $X = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix.
- DKS proposed to find the weighted adjacency matrix $A \in \mathbb{R}^{n \times n}_{\geq 0}$ and $A^\mathsf{T} = A$ such that

$$\min_{A \in \mathbb{R}^{n \times n}_{\geq 0};\, A^\mathsf{T} = A} \left\| L X^\mathsf{T} \right\|_F^2 = \min_{a_{ij} \geq 0} \sum_{i=1}^n \left\| \sum_{j=1}^n a_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_2^2 .$$

- The above objective function looks quite natural since a_{ij} becomes small if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are far apart.

The Daitch-Kelner-Spielman Construction . . .

- If $a_{ij} \equiv 0$, $\forall j$, then $\boldsymbol{x}_i$ is isolated. In fact, if we allow the isolated vertices, then clearly, the minimum is 0, i.e., a graph with no edges is the minimizer.
- To prevent this, the constraints, $d_i = \sum_{j=1}^{n} a_{ij} > 0$, $i = 1, \ldots, n$, are added in the above minimization problem.
- Furthermore, define a *hard graph* of X to be a graph minimizing $\|LX^{\mathsf{T}}\|_F^2$ subject to $d_i \geq 1$, $i = 1, \ldots, n$.
- Since some vectors could be outliers, define an *α-soft graph* of X to be a graph minimizing $\|LX^{\mathsf{T}}\|_F^2$ subject to $\sum_i (\max(0, 1 - d_i))^2 \leq \alpha n$, which constrains the number of edges with small weights.

The Daitch-Kelner-Spielman Construction . . .

- If $a_{ij} \equiv 0$, $\forall j$, then $\boldsymbol{x}_i$ is isolated. In fact, if we allow the isolated vertices, then clearly, the minimum is 0, i.e., a graph with no edges is the minimizer.
- To prevent this, the constraints, $d_i = \sum_{j=1}^{n} a_{ij} > 0$, $i = 1, \dots, n$, are added in the above minimization problem.
- Furthermore, define a *hard graph* of X to be a graph minimizing $\|LX^{\mathsf{T}}\|_F^2$ subject to $d_i \geq 1$, $i = 1, \dots, n$.
- Since some vectors could be outliers, define an *α-soft graph* of X to be a graph minimizing $\|LX^{\mathsf{T}}\|_F^2$ subject to $\sum_i (\max(0, 1 - d_i))^2 \leq \alpha n$, which constrains the number of edges with small weights.

The Daitch-Kelner-Spielman Construction . . .

- If $a_{ij} \equiv 0$, $\forall j$, then $\boldsymbol{x}_i$ is isolated. In fact, if we allow the isolated vertices, then clearly, the minimum is 0, i.e., a graph with no edges is the minimizer.
- To prevent this, the constraints, $d_i = \sum_{j=1}^{n} a_{ij} > 0$, $i = 1, \ldots, n$, are added in the above minimization problem.
- Furthermore, define a *hard graph* of X to be a graph minimizing $\|LX^{\mathsf{T}}\|_F^2$ subject to $d_i \geq 1$, $i = 1, \ldots, n$.
- Since some vectors could be outliers, define an *α-soft graph* of X to be a graph minimizing $\|LX^{\mathsf{T}}\|_F^2$ subject to $\sum_i (\max(0, 1 - d_i))^2 \leq \alpha n$, which constrains the number of edges with small weights.

The Daitch-Kelner-Spielman Construction ...

- If $a_{ij} \equiv 0$, $\forall j$, then $\boldsymbol{x}_i$ is isolated. In fact, if we allow the isolated vertices, then clearly, the minimum is 0, i.e., a graph with no edges is the minimizer.
- To prevent this, the constraints, $d_i = \sum_{j=1}^{n} a_{ij} > 0$, $i = 1, \ldots, n$, are added in the above minimization problem.
- Furthermore, define a *hard graph* of X to be a graph minimizing $\|LX^{\mathsf{T}}\|_F^2$ subject to $d_i \geq 1$, $i = 1, \ldots, n$.
- Since some vectors could be outliers, define an *α-soft graph* of X to be a graph minimizing $\|LX^{\mathsf{T}}\|_F^2$ subject to $\sum_i (\max(0, 1 - d_i))^2 \leq \alpha n$, which constrains the number of edges with small weights.

The Daitch-Kelner-Spielman Construction ...

- To solve such optimization is not easy; need to use *quadratic programming*. The details can be found in their paper.
- Here, we will describe a couple of theorems on the properties of the hard and α-soft graphs.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^d$ has a hard and an α-soft graph with at most $(d+1)n$ edges. Consequently, the average degree of a vertex in such graphs is at most $2(d+1)$.

$\Longrightarrow$ Once such a graph is constructed, the average degree of that graph can be used for the measure of the *essential dimensionality* of the input data vectors, which could be much lower than the ambient dimension d.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^2$ has a hard and an α-soft graph that are planar (i.e., no edges cross each other when they are drawn on the plane).

The Daitch-Kelner-Spielman Construction . . .

- To solve such optimization is not easy; need to use *quadratic programming*. The details can be found in their paper.
- Here, we will describe a couple of theorems on the properties of the hard and α-soft graphs.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^d$ has a hard and an α-soft graph with at most $(d+1)n$ edges. Consequently, the average degree of a vertex in such graphs is at most $2(d+1)$.

$\Longrightarrow$ Once such a graph is constructed, the average degree of that graph can be used for the measure of the *essential dimensionality* of the input data vectors, which could be much lower than the ambient dimension d.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^2$ has a hard and an α-soft graph that are planar (i.e., no edges cross each other when they are drawn on the plane).

The Daitch-Kelner-Spielman Construction . . .

- To solve such optimization is not easy; need to use *quadratic programming*. The details can be found in their paper.
- Here, we will describe a couple of theorems on the properties of the hard and α-soft graphs.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^d$ has a hard and an α-soft graph with at most $(d+1)n$ edges. Consequently, the average degree of a vertex in such graphs is at most $2(d+1)$.

$\Longrightarrow$ Once such a graph is constructed, the average degree of that graph can be used for the measure of the *essential dimensionality* of the input data vectors, which could be much lower than the ambient dimension d.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^2$ has a hard and an α-soft graph that are planar (i.e., no edges cross each other when they are drawn on the plane).

The Daitch-Kelner-Spielman Construction ...

- To solve such optimization is not easy; need to use *quadratic programming*. The details can be found in their paper.
- Here, we will describe a couple of theorems on the properties of the hard and α-soft graphs.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^d$ has a hard and an α-soft graph with at most $(d+1)n$ edges. Consequently, the average degree of a vertex in such graphs is at most $2(d+1)$.

$\Longrightarrow$ Once such a graph is constructed, the average degree of that graph can be used for the measure of the *essential dimensionality* of the input data vectors, which could be much lower than the ambient dimension d.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^2$ has a hard and an α-soft graph that are planar (i.e., no edges cross each other when they are drawn on the plane).

The Daitch-Kelner-Spielman Construction ...

- To solve such optimization is not easy; need to use *quadratic programming*. The details can be found in their paper.
- Here, we will describe a couple of theorems on the properties of the hard and α-soft graphs.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^d$ has a hard and an α-soft graph with at most $(d+1)n$ edges. Consequently, the average degree of a vertex in such graphs is at most $2(d+1)$.

$\Longrightarrow$ Once such a graph is constructed, the average degree of that graph can be used for the measure of the *essential dimensionality* of the input data vectors, which could be much lower than the ambient dimension d.

Theorem (DKS, 2009)

For every $\alpha > 0$, every set of n vectors in $\mathbb{R}^2$ has a hard and an α-soft graph that are planar (i.e., no edges cross each other when they are drawn on the plane).

Some Results from the DKS paper

- As examples, DKS used the well-known datasets from the UCI Machine Learning Repository (Asuncion & Newman, 2007) or LIBSVM (Chang & Lin, 2001).
- Performed classification, regression, and clustering experiments on these datasets.
- Here, we show only their classification results.
- Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be the available vectors for a given classification problem, and let $T = \{\boldsymbol{x}_i\}_{i \in I_T}$ be a set of m labeled training vectors ($m < n$), and $I_T \subset N := \{1, \ldots, n\}$ is the index set for the training vectors, and $|I_T| = m$. For the 10-fold cross validation, $m \approx n/10$.
- Then, the classification problem is to build a classifier/predictor using the label information in T to predict a label of each vector in the test dataset $X \setminus T$.
- For a given classification problem, DKS used the whole dataset X to construct a graph using their optimization approach.

Some Results from the DKS paper

- As examples, DKS used the well-known datasets from the UCI Machine Learning Repository (Asuncion & Newman, 2007) or LIBSVM (Chang & Lin, 2001).
- Performed classification, regression, and clustering experiments on these datasets.
- Here, we show only their classification results.
- Let $X = \{x_1, \ldots, x_n\}$ be the available vectors for a given classification problem, and let $T = \{x_i\}_{i \in I_T}$ be a set of m labeled training vectors ($m < n$), and $I_T \subset N := \{1, \ldots, n\}$ is the index set for the training vectors, and $|I_T| = m$. For the 10-fold cross validation, $m \approx n/10$.
- Then, the classification problem is to build a classifier/predictor using the label information in T to predict a label of each vector in the test dataset $X \setminus T$.
- For a given classification problem, DKS used the whole dataset X to construct a graph using their optimization approach.

Some Results from the DKS paper

- As examples, DKS used the well-known datasets from the UCI Machine Learning Repository (Asuncion & Newman, 2007) or LIBSVM (Chang & Lin, 2001).
- Performed classification, regression, and clustering experiments on these datasets.
- Here, we show only their classification results.
- Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be the available vectors for a given classification problem, and let $T = \{\boldsymbol{x}_i\}_{i \in I_T}$ be a set of m labeled training vectors ($m < n$), and $I_T \subset N := \{1, \ldots, n\}$ is the index set for the training vectors, and $|I_T| = m$. For the 10-fold cross validation, $m \approx n/10$.
- Then, the classification problem is to build a classifier/predictor using the label information in T to predict a label of each vector in the test dataset $X \setminus T$.
- For a given classification problem, DKS used the whole dataset X to construct a graph using their optimization approach.

Some Results from the DKS paper

- As examples, DKS used the well-known datasets from the UCI Machine Learning Repository (Asuncion & Newman, 2007) or LIBSVM (Chang & Lin, 2001).
- Performed classification, regression, and clustering experiments on these datasets.
- Here, we show only their classification results.
- Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be the available vectors for a given classification problem, and let $T = \{\boldsymbol{x}_i\}_{i \in I_T}$ be a set of m labeled training vectors ($m < n$), and $I_T \subset N := \{1, \ldots, n\}$ is the index set for the training vectors, and $|I_T| = m$. For the 10-fold cross validation, $m \approx n/10$.
- Then, the classification problem is to build a classifier/predictor using the label information in T to predict a label of each vector in the test dataset $X \setminus T$.
- For a given classification problem, DKS used the whole dataset X to construct a graph using their optimization approach.

Some Results from the DKS paper

- As examples, DKS used the well-known datasets from the UCI Machine Learning Repository (Asuncion & Newman, 2007) or LIBSVM (Chang & Lin, 2001).
- Performed classification, regression, and clustering experiments on these datasets.
- Here, we show only their classification results.
- Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be the available vectors for a given classification problem, and let $T = \{\boldsymbol{x}_i\}_{i \in I_T}$ be a set of m labeled training vectors $(m < n)$, and $I_T \subset N := \{1, \ldots, n\}$ is the index set for the training vectors, and $|I_T| = m$. For the 10-fold cross validation, $m \approx n/10$.
- Then, the classification problem is to build a classifier/predictor using the label information in T to predict a label of each vector in the test dataset $X \setminus T$.
- For a given classification problem, DKS used the whole dataset X to construct a graph using their optimization approach.

Some Results from the DKS paper

- As examples, DKS used the well-known datasets from the UCI Machine Learning Repository (Asuncion & Newman, 2007) or LIBSVM (Chang & Lin, 2001).
- Performed classification, regression, and clustering experiments on these datasets.
- Here, we show only their classification results.
- Let $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be the available vectors for a given classification problem, and let $T = \{\boldsymbol{x}_i\}_{i \in I_T}$ be a set of m labeled training vectors $(m < n)$, and $I_T \subset N := \{1, \ldots, n\}$ is the index set for the training vectors, and $|I_T| = m$. For the 10-fold cross validation, $m \approx n/10$.
- Then, the classification problem is to build a classifier/predictor using the label information in T to predict a label of each vector in the test dataset $X \setminus T$.
- For a given classification problem, DKS used the whole dataset X to construct a graph using their optimization approach.

Some Results from the DKS paper . . .

- Their actual classification method is based on the simple algorithm of Zhu, Ghahramani, & Lafferty (2003). The two-class classifier can be described as follows:
    1. Construct a graph G from X via the DKS algorithm
    2. Let $\{c_i \in \{0,1\}\}_{i \in I_T}$ be the training dataset labels (either 0 or 1). Then, solve
    $$\widehat{\boldsymbol{y}} = \operatorname*{arg\,min}_{\boldsymbol{y} \in \mathbb{R}^n} \boldsymbol{y}^{\mathsf{T}} L(G) \boldsymbol{y} \quad \text{subject to } y_i = c_i \text{ if } i \in I_T.$$
    3. For each test vector $\boldsymbol{x}_j$, $j \in N \setminus I_T$, classify it according to the following rule:
    $$c_j = \begin{cases} 0 & \text{if } y_j < 1/2; \\ 1 & \text{otherwise.} \end{cases}$$
- One can generalize this for problems with more than two classes.

Some Results from the DKS paper . . .

- Their actual classification method is based on the simple algorithm of Zhu, Ghahramani, & Lafferty (2003). The two-class classifier can be described as follows:
  1. Construct a graph G from X via the DKS algorithm
  2. Let $\{c_i \in \{0,1\}\}_{i \in I_T}$ be the training dataset labels (either 0 or 1). Then, solve
  $$\widehat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathbb{R}^n}{\arg\min}\, \boldsymbol{y}^{\mathsf{T}} L(G) \boldsymbol{y} \quad \text{subject to } y_i = c_i \text{ if } i \in I_T.$$
  3. For each test vector $\boldsymbol{x}_j$, $j \in N \setminus I_T$, classify it according to the following rule:
  $$c_j = \begin{cases} 0 & \text{if } y_j < 1/2; \\ 1 & \text{otherwise.} \end{cases}$$
- One can generalize this for problems with more than two classes.

Some Results from the DKS paper . . .

- Their actual classification method is based on the simple algorithm of Zhu, Ghahramani, & Lafferty (2003). The two-class classifier can be described as follows:
  1. Construct a graph G from X via the DKS algorithm
  2. Let $\{c_i \in \{0,1\}\}_{i \in I_T}$ be the training dataset labels (either 0 or 1). Then, solve
  $$\widehat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathbb{R}^n}{\arg\min}\, \boldsymbol{y}^{\mathsf{T}} L(G) \boldsymbol{y} \quad \text{subject to } y_i = c_i \text{ if } i \in I_T.$$
  3. For each test vector $\boldsymbol{x}_j$, $j \in N \setminus I_T$, classify it according to the following rule:
  $$c_j = \begin{cases} 0 & \text{if } y_j < 1/2; \\ 1 & \text{otherwise.} \end{cases}$$
- One can generalize this for problems with more than two classes.

Some Results from the DKS paper . . .

- Their actual classification method is based on the simple algorithm of Zhu, Ghahramani, & Lafferty (2003). The two-class classifier can be described as follows:
  1. Construct a graph G from X via the DKS algorithm
  2. Let $\{c_i \in \{0,1\}\}_{i \in I_T}$ be the training dataset labels (either 0 or 1). Then, solve
  $$\widehat{\boldsymbol{y}} = \arg\min_{\boldsymbol{y} \in \mathbb{R}^n} \boldsymbol{y}^{\mathsf{T}} L(G) \boldsymbol{y} \quad \text{subject to } y_i = c_i \text{ if } i \in I_T.$$
  3. For each test vector $\boldsymbol{x}_j$, $j \in N \setminus I_T$, classify it according to the following rule:
  $$c_j = \begin{cases} 0 & \text{if } y_j < 1/2; \\ 1 & \text{otherwise.} \end{cases}$$
- One can generalize this for problems with more than two classes.

Some Results from the DKS paper ...

- Their actual classification method is based on the simple algorithm of Zhu, Ghahramani, & Lafferty (2003). The two-class classifier can be described as follows:
  1. Construct a graph G from X via the DKS algorithm
  2. Let $\{c_i \in \{0,1\}\}_{i \in I_T}$ be the training dataset labels (either 0 or 1). Then, solve
  $$\widehat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathbb{R}^n}{\arg\min}\, \boldsymbol{y}^{\mathsf{T}} L(G)\, \boldsymbol{y} \quad \text{subject to } y_i = c_i \text{ if } i \in I_T.$$
  3. For each test vector $\boldsymbol{x}_j$, $j \in N \setminus I_T$, classify it according to the following rule:
  $$c_j = \begin{cases} 0 & \text{if } y_j < 1/2; \\ 1 & \text{otherwise.} \end{cases}$$
- One can generalize this for problems with more than two classes.

Classification Results from the DKS paper

Table 2. Classification error (%), 10-fold cross validation. The best result for each data set is bold. The experiments that do not perform better than ours have a grey background.

DATA SET	HARD	0.1-SOFT	KNN	THRESH	LIBSVM	FBC	AODE	HGC	NB	C4.5	BP	SMO
GLASS	27.78	28.30	**26.92**	33.30	31.44	37.56	38.27	41.64	50.55	32.37	32.68	42.64
HEART	18.18	17.81	**16.05**	16.1	17.01	16.19	16.37	17.41	16.41	21.85	16.70	16.19
IONOSPHERE	**4.75**	5.57	18.50	6.34	6.20	9.20	8.26	6.60	17.83	10.26	12.93	12.07
IRIS	4.87	4.21	4.46	6.20	**3.87**	6.27	6.00	3.93	4.47	5.27	15.20	15.13
PIMA	26.64	26.61	24.54	26.45	23.24	25.15	23.43	24.08	24.25	25.51	22.96	**22.93**
SONAR	9.16	**8.64**	13.80	14.94	11.71	22.62	20.09	30.84	32.29	26.39	21.33	22.12
VEHICLE	23.03	22.47	27.70	29.98	**14.87**	25.77	28.35	31.90	55.32	27.72	18.89	25.92
VOWEL990	1.19	0.95	2.62	0.98	**0.64**	6.54	10.36	7.30	37.10	19.80	7.27	29.39
WINE	2.92	2.62	2.86	3.64	2.57				2.54	6.80	1.98	**1.24**

FBC: Full Bayes Classifier; AODE: Averaged One-Dependence Estimators; HGC: the Hill Climbing Bayesian network learning algorithm; NB: Naive Bayesian networks; C4.5: a decision tree algorithm; BP: Back Propagation; SMO: Sequential Minimal Optimization