Graph Laplacian Regularization for Large-Scale Semidefinite Programming

Kilian Weinberger et al.
NIPS 2006

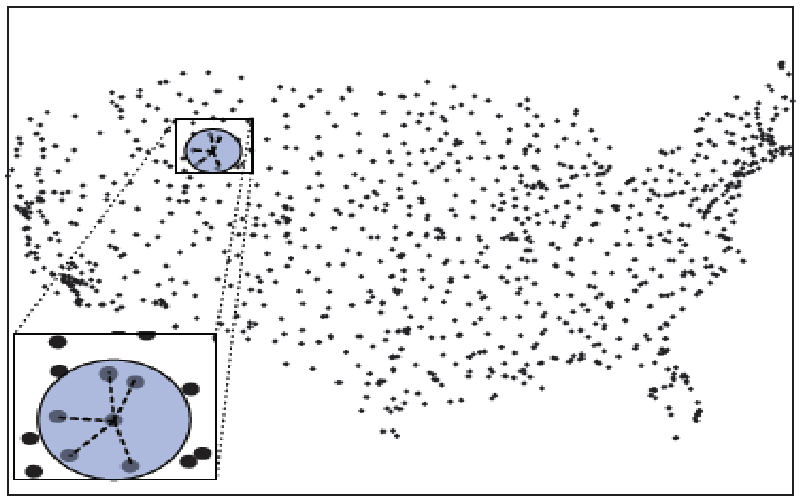
presented by Aggeliki Tsoli

Introduction

- ***Problem***
  - discovery of low dimensional representations of high-dimensional data
  - in many cases, local proximity measurements also available
  - e.g. computer vision, sensor localization
- Current Approach
  - semidefinite programs (SDPs) – convex optimization
  - Disadvantage: it doesn’t scale well for large inputs
- Paper Contribution
  - method for solving ***very large*** problems of the above type
  - much smaller/faster SDPs than those previously studied

Sensor localization

- Determine the 2D position of the sensors based on estimates of local distances between neighboring sensors
  - sensors i, j neighbors iff sufficiently close to estimate their pairwise distance via limited-range radio transmission
- ***Input***:
  - n sensors
  - d_{ij} : estimate of local distance between neighboring sensors i,j
- ***Output***:
  - $x_1, x_2, \ldots x_n \in R^2$: planar coordinates of sensors



Work so far…

- Minimize sum-of-squares loss function

$$\min_{\vec{x}_1,\ldots,\vec{x}_n} \sum_{i\sim j} \left(\|\vec{x}_i - \vec{x}_j\|^2 - d_{ij}^2 \right)^2 \qquad (1)$$

- Centering constraint (assuming no sensor location is known in advance)

$$\left\| \sum_i \vec{x}_i \right\|^2 = 0 \qquad (2)$$

- Optimization in (1) non convex
  → Likely to be trapped in local minima !

Convex Optimization

- **Convex function**
  - a real-valued function *f* defined on a domain C that for any two points *x* and *y* in *C* and any *t* in [0,1],
  - $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$
- **Convex optimization**

Standard form is the usual and most intuitive form of describing a convex optimization problem. It consists of the following three parts:

- A **convex function** $f_0(x) : \mathbb{R}^n \to \mathbb{R}$ to be minimized over the variable x
- **Inequality constraints** of the form $f_i(x) \leq 0$, where the functions f_i are convex
- **Equality constraints** of the form $h_i(x) = 0$, where the functions h_i are affine. In practice, the terminology "linear" and "affine" are generally equivalent and most often expressed in the form $Ax = b$, where A is a matrix and b is a vector.

A convex optimization problem is thus written as

minimize $f_0(x)$ subject to

$$f_i(x) \leq 0, \quad i = 1, \ldots, m$$
$$h_i(x) = 0, \quad i = 1, \ldots, p$$

Solution to convexity

- Define n x n inner-product matrix X
  - $X_{ij} = x_i \cdot x_j$
- Get convex optimization by relaxing the constraint that sensor locations x_i lie in the R^2 plane

$$\begin{array}{ll} \textbf{Minimize:} & \sum_{i\sim j}\left(X_{ii} - 2X_{ij} + X_{jj} - d_{ij}^2\right)^2 \\ \textbf{subject to:} & \textbf{(i)} \sum_{ij} X_{ij} = 0 \quad \textbf{and} \quad \textbf{(ii)}\ X \succeq 0. \end{array} \quad (3)$$

- x_i vectors will lie in a subspace with dimension equal to the rank of the solution X
  - ➔ Project x_i s into their 2D subspace of maximum variance to get planar coordinates

Maximum Variance Unfolding (MVU)

- The higher the rank of X, the greater the information loss after projection
- Add extra term to the loss function to favor solutions with high variance (or high trace)

$$\begin{array}{ll} \text{Maximize:} & \mathrm{tr}(\mathrm{X}) - \nu \sum_{i\sim j} \left(\mathrm{X}_{ii} - 2\mathrm{X}_{ij} + \mathrm{X}_{jj} - d_{ij}^2\right)^2 \\ \text{subject to:} & \text{(i)} \sum_{ij} \mathrm{X}_{ij} = 0 \quad \text{and} \quad \text{(ii)}\ \mathrm{X} \succeq 0. \end{array} \qquad (4)$$

  - **trace of square matrix X (tr(X)):** sum of the elements on X's main diagonal
- parameter v > 0 balances the trade-off between maximizing variance and preserving local distances (**maximum variance unfolding - MVU**)

Matrix factorization (1/2)

- G : neighborhood graph defined by the sensor network
- Assume location of sensors is a function defined over the nodes of G
- Functions on a graph can be approximated using eigenvectors of graph's Laplacian matrix as basis functions (spectral graph theory)
  - graph Laplacian l: $l_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ adjacent } v_j \\ 0 & \text{otherwise} \end{cases}$
  - eigenvectors of graph Laplacian matrix ordered by smoothness
- Approximate sensors' locations using the m bottom eigenvectors of the Laplacian matrix of G
  - $x_i \approx \Sigma_{\alpha=1}^{m} Q_{i\alpha} y_\alpha$
  - Q : n x m matrix with the m bottom eigenvectors of Laplacian matrix (precomputed)
  - y_α : m x 1 vector , $\alpha = 1, \ldots, m$ (unknown)

Matrix factorization (2/2)

- Define m x m inner-product matrix Y
  - $Y_{\alpha\beta} = y_\alpha \cdot y_\beta$
- Factorize matrix X
  - $X \approx QYQ^T$
- Get equivalent optimization
  - tr(Y) = tr(X), since Q stores mutually orthogonal eigenvectors
  - QYQ^T satisfies centering constraint (uniform eigenvector not included)

$$\begin{array}{ll} \textbf{Maximize:} & \mathrm{tr}(\mathrm{Y}) - \nu \sum_{i\sim j} \left[(\mathrm{QYQ}^\top)_{ii} - 2(\mathrm{QYQ}^\top)_{ij} + (\mathrm{QYQ}^\top)_{jj} - d_{ij}^2\right]^2 \\ \textbf{subject to:} & \mathrm{Y} \succeq 0 \end{array} \quad (5)$$

- Instead of the n x n matrix X, optimization is solved for the much smaller m x m matrix Y !

Formulation as SDP

- Approach for large input problems:
  - cast the required optimization as SDP over small matrices with few constraints
- Rewrite the previous formula as an SDP in standard form
  - ̣ є $^{m^2}$: vector obtained by concatenating all the columns of Y
  - Ạє $^{m^2 \times m^2}$: positive semidefinite matrix collecting all the quadratic coefficients in the objective function
  - ḅє ^{m2} : vector collecting all the linear coefficients in the objective function
  - l : lower bound on the quadratic piece of the objective function
    - Use Schur’s lemma to express this bound as a linear matrix inequality

$$\begin{array}{ll} \textbf{Maximize:} & b^\top \mathcal{Y} - \ell \\ \textbf{subject to:} & \text{(i) } \mathrm{Y} \succeq 0 \quad \text{and} \quad \text{(ii)} \begin{bmatrix} \mathrm{I} & \mathrm{A}^{\frac{1}{2}}\mathcal{Y} \\ (\mathrm{A}^{\frac{1}{2}}\mathcal{Y})^\top & \ell \end{bmatrix} \succeq 0. \end{array} \qquad (6)$$

Formulation as SDP

- Approach for large input problems:
  - cast the required optimization as SDP over small matrices with few constraints

$$
\begin{array}{ll}
\textbf{Maximize:} & b^\top \mathcal{Y} - \ell \\
\textbf{subject to:} & \textbf{(i) } \mathrm{Y} \succeq 0 \quad \textbf{and} \quad \textbf{(ii)} \begin{bmatrix} \mathrm{I} & \mathrm{A}^{\frac{1}{2}}\mathcal{Y} \\ (\mathrm{A}^{\frac{1}{2}}\mathcal{Y})^\top & \ell \end{bmatrix} \succeq 0.
\end{array}
\qquad (6)
$$

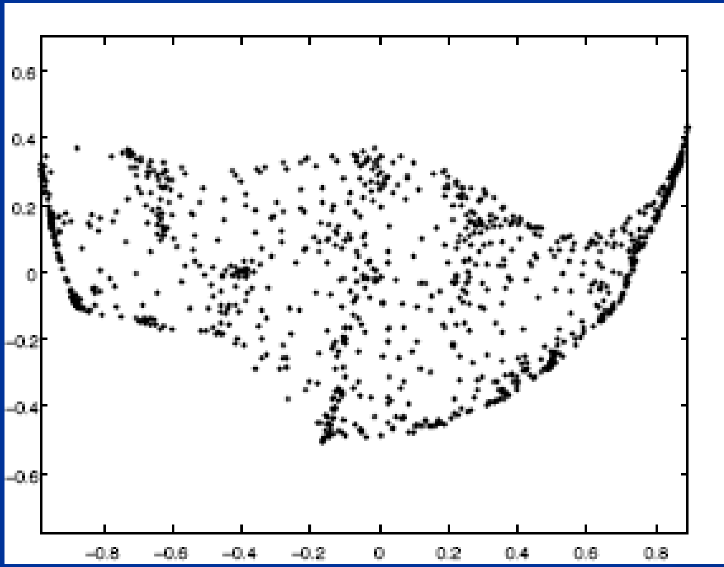
- *Unknown variables*: m(m+1)/2 elements of Y and scalar **l**
- *Constraints*: positive semidefinite constraint on Y and linear matrix inequality of size m^2 x m^2
- ***The complexity of the SDP does not depend on the number of nodes (n) or edges in the network!***

Gradient-based improvement

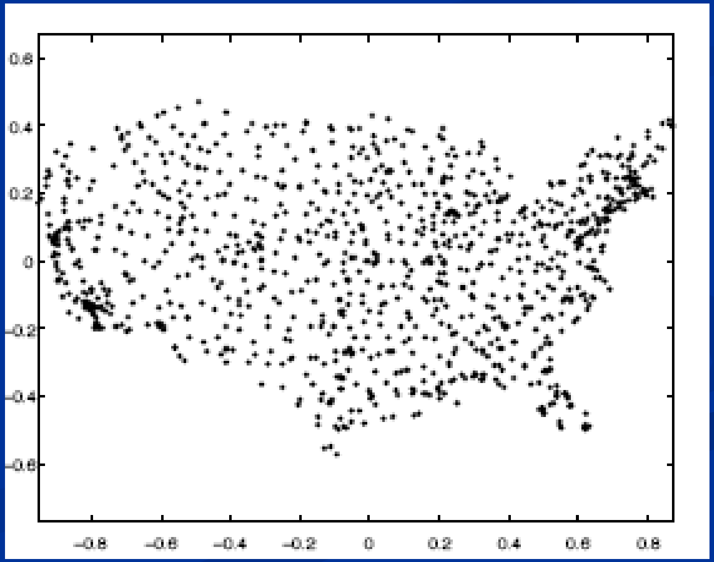
- 2-step process (optional):
  - Starting from the m-dimensional solution of eq. (6), use conjugate gradient methods to maximize the objective function in eq. (4)
  - Project the results from the previous step into the R^2 plane and use conjugate gradient methods to minimize the loss function in eq. (1)
    - **conjugate gradient method**: iterative method for minimizing a quadratic function where its Hessian matrix (matrix of second partial derivatives) is positive definite

Results (1/2)

- n = 1055 largest cities in continental US
- local distances up to 18 neighbors within radius r = 0.09
- local measurements corrupted by 10% Gaussian noise over the true local distance
- m = 10 bottom eigenvectors of graph Laplacian



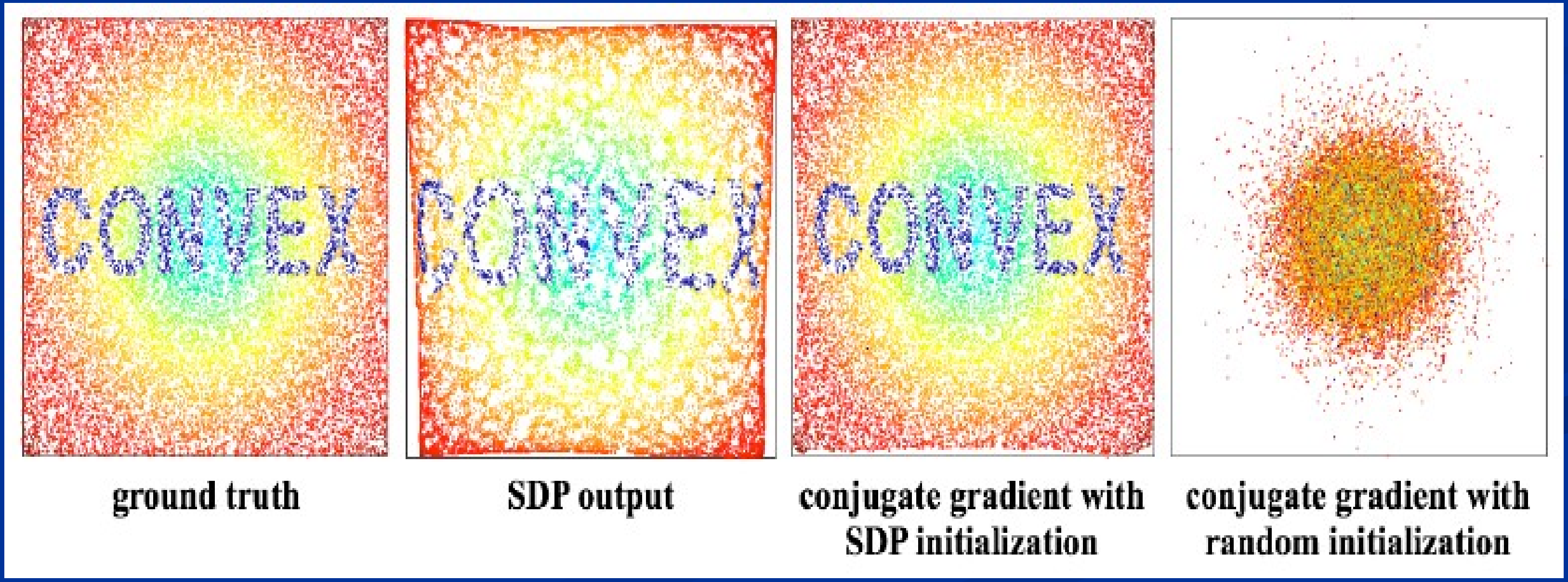
Result from SDP in (9) ~ 4s



Result after conjugate gradient descent

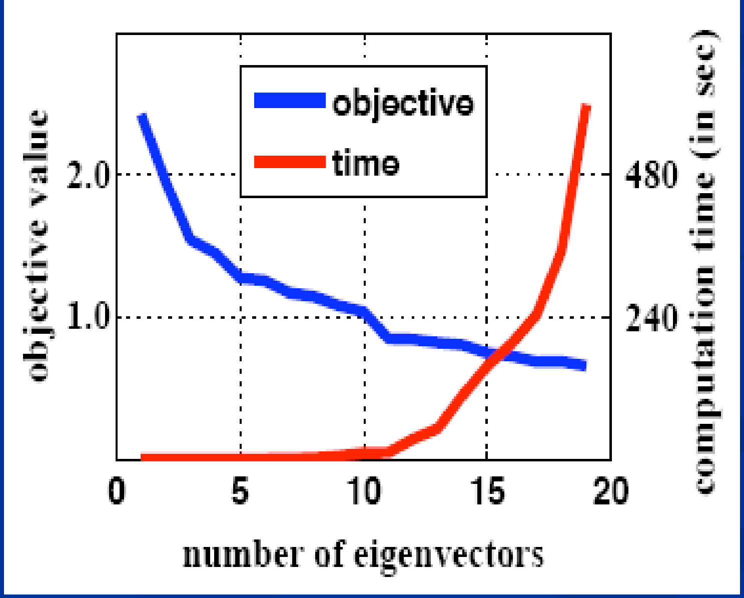
Results (2/2)

- n = 20,000 uniformly sampled points inside the unit square
- local distances up to 20 other nodes within radius r = 0.06
- m = 10 bottom eigenvectors of graph Laplacian
- 19s to construct and solve the SDP
- 52s for 100 iterations in conjugate gradient descent



Results (3/3)

- loss function in eq. (1) vs. number of eigenvectors
- computation time vs. number of eigenvectors
- “sweet spot” around m ≈ 10 eigenvectors



FastMVU on Robotics

- Control of a robot using sparse user input
  - e.g. 2D mouse position
- Robot localization
  - the robot's location is inferred from the high dimensional description of its state in terms of sensorimotor input

Conclusion

- Approach for inferring low dimensional representations from local distance constraints using MVU
- Use of matrix factorization computed from the bottom eigenvectors of the graph Laplacian
- Local search methods can refine solution
- Suitable for large input; its complexity does not depend on the input!