Graph Constructions for Machine Learning
Optimization and Combinatorial Properties

Christopher Musco
Advisor: Daniel Spielman

December 09, 2011

Abstract

Increased relevance of graph based clustering, regression, and classification has motivated an interest in constructing graphs to fit vector data points, a more typical raw input for machine learning algorithms than naturally occurring graphs. At least in certain problems, it seems that graphs provide a particularly natural tool for capturing both local and global structure in high-dimensional vector data. One recent proposal from a paper by S. Daitch, J. Kelner, and my project advisor, D. Spielman, suggests constructing undirected graphs with edge weights chosen to best approximate each vector as a weighted sum of its neighbors. This objective is intuitively natural since functions on graphs are often approximated by taking a weighted average of the function’s value on a vertex’s neighbors. Additionally, the optimal graphs, which can be found by solving quadratic programs, seem practically useful as well as combinatorially interesting. This semester I explored the construction of such graphs for large datasets and considered some interesting questions related to sparseness, uniqueness, and planarity of the graphs. My later work eventually led to the construction of a counter-example for the conjecture that the optimal graph is unique for points in general position. The details and properties of this counter-example, which were worked out in collaboration with Cameron Musco, will be included in a forthcoming, jointly written report.

1 Background

1.1 Fitting Graphs to Vector Data

Graphs are intuitively useful for clustering and labeling in models that support a notion of edges. Natural connections exist between the nodes in social networks, linked sets of web pages, paper citation corpuses, etc. By representing these connections in a weighted or unweighted graph it is possible to predict a node’s characteristics from its neighbors or to cluster data based on the graph’s structure.

However, researchers have also attempted to extend the use of graphs in order to understand data that is not naturally connected. For example, a problem might center around a set of many-dimensional vectors - a more common form of data in machine learning. Without edge information the task becomes two-fold:

1. Create a graph by constructing edges between data points based on (potentially high-dimensional) spatial information.

2. Use the ‘artificial’ graph to perform learning operations on the data (cluster data using spectral techniques, label unlabeled vectors based on neighbors, etc.)

Since research has primarily focused on the second step, fairly simple algorithms are typically used for graph construction [Tal09]. For example, exponentially weighted $k$-nearest neighbor graphs and graphs that construct edges between all vertices within a given threshold distance are popular [Zhu05].
More recently, attempts have been made to improve initial graph construction. [MH08] This project focuses on one system in particular that was presented in a 2009 paper titled “Fitting a Graph to Vector Data,” by Samuel Daitch, Jonathan Kelner, and my project advisor, Daniel Spielman [DKS09]. In this paper an attempt is made to find a natural graph by constructing edges in such a way that vertices are spatially well approximated by their neighbors. Intuitively, this aligns with the idea that a function on a graph can be approximated through its value on a vertex’s neighborhood.

1.2 Daitch, Kelner, & Spielmans’s Proposal

Consider a problem with \( n \) data vectors, \( \{ \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \} \). We’ll construct an undirected, weighted graph, \( G \), with weight \( w_{i,j} \geq 0 \) on edge \( e_{i,j} \) between vertices \( i \) and \( j \). There are no self-loops so \( w_{i,i} = 0 \) \( \forall \ i \). The authors seek the graph that minimizes the objective function:

\[
 f(G) = \sum_{i=1}^{n} \left( \deg_i \mathbf{x}_i - \sum_{j=1}^{n} w_{i,j} \mathbf{x}_j \right)^2
\]

where \( \deg_i = \sum_{j \neq i} w_{i,j} \), which is the weighted degree of the graph vertex corresponding to vector \( \mathbf{x}_i \). We will also write this function as \( f(\mathbf{w}) \), where \( \mathbf{w} \) is an \( \binom{n}{2} \) length vector of all the edge weights in \( G \) (excluding self-loops, since they always have weight 0). Notice that:

\[
 \deg_i \mathbf{x}_i - \sum_{j \neq i} w_{i,j} \mathbf{x}_j = \deg_i \left( \mathbf{x}_i - \sum_{j \neq i} \frac{w_{i,j}}{\deg_i} \mathbf{x}_j \right)
\]

Thus, \( f(\mathbf{w}) \) computes a sum of the squared distances between each vertex and the weighted average of its neighbors. The sum is weighted by \( \deg_i \), so higher degree vertices are more important to the objective function.

Of course this allows us to minimized the function trivially by setting all edge weights, and thus all degrees, to 0. So, the authors suggest two possible restrictions to avoid degenerate solutions. Most simply, we can restrict our solution such that \( \deg_i = \sum_{j \neq i} w_{i,j} \geq 1 \) for all vertices. The graph resulting from this constraint is called the “hard graph”. Alternatively, we can allow additional flexibility (some vertices with weighted degree below 1) by constraining the problem such that \( \sum_{i=1}^{n} (\max(0, 1 - \deg_i))^2 \leq \alpha n \) for some constant \( \alpha \). The graph corresponding to this second constraint is called the \( \alpha \)-soft graph.

It is helpful to notice that \( f(\mathbf{w}) \) is equivalent to \( \| LX \|_F^2 \), where \( L \) is the graph Laplacian and \( X \) is an \( n \)-by-\( d \) data matrix with each data vector as a row (\( d \) is the number of dimensions in our feature space). This formulation is perhaps the most intuitive way to think about the objective function and appears most in the code connected to this project and in my report. Each column of \( X \), and thus each column of \( LX \), corresponds to a dimension. Consider a vector \( \mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^T \). So, \( x_{ik} \) is the component of \( \mathbf{x}_i \) in the \( k \)th dimension. Then:

\[
 LX_{i,k} = \deg_i x_{i}^k - \sum_{j \neq i} w_{i,j} x_j^k
 = \left( \sum_{j \neq i} w_{i,j} \right) x_j^k - \sum_{j \neq i} w_{i,j} x_j^k
 = \sum_{j \neq i} w_{i,j} (x_j^k - x_i^k)
\]

Each element of the \( i \)th row of \( LX \) holds the prediction error for vector \( \mathbf{x}_i \) in a single dimension. So, row \( i \) is an error vector, scaled by \( \deg_i \), between the true value of \( \mathbf{x}_i \) and its value as predicted
by the weighted average of its neighbors. Thus, the Frobenius norm of $LX$ is simply the weighted sum of squared differences between each vector and its predicted value.

### 1.3 Alternative Formulation

However, optimizing a Frobenius norm over the Laplacian matrix is not especially natural, so Daitch et al. introduce notation that makes it possible to represent $\|LX\|_F$ as the vector norm of a matrix-vector product. Again, the details can be found in [DKS09] section 2.1, but briefly, it is possible to set $\|LX\|_F^2 = \|Mw\|^2$, where $w$ is our $\binom{n}{2}$ length vector of edge weights. $M$ is an $(nd)\times|E|$ matrix, that can be thought of as $d$ vertically stacked $n\times|E|$ matrices, each holding difference vectors for a single dimension. An expanded representation of $M$ is included below. Notice that each row corresponds to a vertex and only contains non-zero entries in columns corresponding to edges adjacent to that vertex.

$$
\begin{bmatrix}
(x_1^2 - x_2^2) & (x_1^2 - x_1^1) & \cdots & (x_1^2 - x_n^1) & 0 & \cdots & 0 & 0 & \cdots & 0 \\
(x_2^2 - x_1^2) & 0 & \cdots & 0 & (x_2^2 - x_1^1) & \cdots & (x_2^2 - x_n^1) & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & (x_n^2 - x_1^1) & 0 & \cdots & (x_n^2 - x_1^1) & 0 & \cdots & (x_n^2 - x_{n-1}^1) \\
(x_1^3 - x_2^3) & (x_1^3 - x_1^2) & \cdots & (x_1^3 - x_1^1) & 0 & \cdots & 0 & 0 & \cdots & 0 \\
(x_2^3 - x_1^3) & 0 & \cdots & 0 & (x_2^3 - x_2^1) & \cdots & (x_2^3 - x_2^1) & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & (x_n^3 - x_2^1) & 0 & \cdots & (x_n^3 - x_2^1) & 0 & \cdots & (x_n^3 - x_{n-1}^1) \\
(x_1^4 - x_2^4) & (x_1^4 - x_1^3) & \cdots & (x_1^4 - x_1^2) & 0 & \cdots & 0 & 0 & \cdots & 0 \\
(x_2^4 - x_1^4) & 0 & \cdots & 0 & (x_2^4 - x_2^3) & \cdots & (x_2^4 - x_2^3) & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & (x_n^4 - x_2^2) & 0 & \cdots & (x_n^4 - x_2^2) & 0 & \cdots & (x_n^4 - x_{n-1}^2)
\end{bmatrix}
$$

Thus, the $(i + n(k - 1))^{th}$ element of $Mw = \sum_{j=1}^{n} w_{i,j} (x_i^k - x_j^k)$, and $\|LX\|_F^2 = \|Mw\|^2$ as desired. This matrix was most important in both the optimization code used to compute the graphs in [DKS09] and in thinking about combinatorial properties of these graphs.

### 2 Algorithmic Considerations

#### 2.1 Original Algorithm

As demonstrated in [DKS09], finding an optimal graph is a convex optimization problem and Daitch et al. show how to formulate it as a quadratic program, for hard constraints, or more specifically, a nonnegative least squares problem, for $\alpha$-soft constraints.

Unfortunately, direct optimization becomes intractable as $n$ increases since we have to solve for the weight of every potential edge in the graph, leaving $\binom{n}{2}$ free variables. However, as discussed in more detail later (Section 3.2), it turns out that the objective function proposed leads to sparse graphs. Every set of vertices has an optimal soft and hard graph with $\leq (d + 1)n$ edges, where $d$ is the dimensionality of the data vectors [DKS09].
This realization led Daitch et al. to introduce an iterative method for finding the optimal graph: it is possible to solve the quadratic program or nonnegative least squares problem on a subset of the potential edges in \( G \), setting all other edge weights to 0. For example, the program might start by only allowing non-zero weight on the edges in a quickly computed \( k \)-nearest neighbors graph. Next, the algorithm considers the derivative of the Lagrange function (or least squares function) at the current solution with respect to each \( w_{i,j} \) that was forced to zero. If the derivative is nonnegative for all \( w_{i,j} \), we have found a solution to the program. Otherwise, it is possible to reduce the objective function by adding weight to some \( w_{i,j} \). So, the algorithm simply re-solves a slightly larger program, allowing previously excluded edges with the lowest derivatives to vary as well.

By continuing in this way the program will eventually find a solution, hopefully without having to consider all possible edges. To avoid overwhelming the solver, the number of edges added in each iteration of the algorithm is either limited directly, or limited such that only a certain number edges can be added for each vertex (bounding how much the degree of each vertex is allowed to grow in each step).

The plots below demonstrate how a solution might evolve over time. Cyan indicates possible edges in the graph, black indicates non-zero weight edges included in the current restricted solution, and blue indicates additional edges to be considered in the next iteration. At least for this small example, only a fraction of the total edges were considered before converging to a final solution:

Figure 1: 30 Random Vectors in 2 Dimensions

(a) Initial k-NN Edge Mask  
(b) Iteration 2

(c) Iteration 3  
(d) Final Graph
It is interesting to note that this iterative method is, on a high level, similar to the popular active set nonnegative least squares algorithm introduced in [LH74], which solves on a subset of variables before adding restricted, negative-derivative variables to the problem after each round. It would be interesting to see whether similar convergence results to those demonstrated by Lawson and Hanson could be established for the algorithm introduced in [DKS09], even if only a selection of candidate negative-derivative edges are added in each round.

2.2 Soft Graph Algorithm

One advantage of the soft objective function is that $\alpha$-graphs can be found by solving a nonnegative least squares problem of the form:

$$\min_{x \geq 0} \|Ex - f\|$$

The nonnegativity constraint on $x$ simplifies the problem considerably in comparison to the linear $Aw - 1 \geq 0$ constraint used when solving for hard graphs. A more detailed explanation can be found in [DKS09], but to find an $\alpha$-graph, it is necessary to optimize:

$$\min_{w, s} \left[ \|Mw\|^2 + \mu \|1 - Aw + s\|^2 : w, s \geq 0 \right]$$

$A$ is the absolute value of the vertex-edge transfer matrix, $U$, introduced in [DKS09]. So, $Aw = d$, where $d$ is an $n$-length vector containing the weighted degrees of each vertex. $s$ is a vector of $n$ slack variables allowed to take on any nonnegative value, and thus $\|1 - Aw + s\|^2 = \|1 - d + s\|^2 = \sum_{i=0}^n (\max(0, 1 - deg_i))^2$, which is exactly the degree constraint function for the soft graph. Thus, $\mu$ acts as a softness parameter that changes inversely with $\alpha$. To obtain a lower value for $\alpha$, we simply increase $\mu$, and to find the graph for a particular $\alpha$, we can perform a binary search over $\mu$ [DKS09].

The above minimization is solved as a nonnegative least squares problem by setting $E$ equal to:

$$E = \begin{bmatrix} M & 0_{n \times n} \\ \sqrt{\mu}A & -I_{n \times n} \end{bmatrix}$$

Where $0_{n \times n}$ and $I_{n \times n}$ are the $n$-by-$n$ all-zeros and identity matrices. $f$ is set to:

$$n \times d = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -\sqrt{\mu} \\ \vdots \\ -\sqrt{\mu} \end{bmatrix}$$

Finally, $x = \begin{bmatrix} w \\ s \end{bmatrix}$.

Again, the nonnegativity constraint is simpler than the linear constraint in the hard graph problem, so the quadratic program is much faster to solve. Thus, profiling and code testing focused on the soft graph algorithm and specifically on the degree-constrained edge addition method implemented in degClose1nnSearcher.m.
2.3 Nonnegative Least Squares

Initial work focused on the actual nonnegative least squares (NNLSQ) optimization problem that is solved at each iteration of the edge-addition algorithm. In the original degCloseInnSearcher.m code used by Daitch et al., a binary search is performed over different values of $\mu$ until $\|1-A\hat{w}+\hat{s}\|^2 = \sum_{i=0}^{n}(\max(0,1-deg_i))^2 \approx \alpha n$ for the desired value of $\alpha$. To avoid incurring the full cost of running the edge-addition algorithm (implemented in degCloseInnFaster.m) for each value of $\mu$ tested, the quadratic program is ‘warm-started’ by initially solving over only non-zero edges in the graph for the previous value of $\mu$. As $\mu$ converges, the set of edges included in the graph also converges, even if the weights on these edges change. For example, a graph that required 10-20 edge additions (and thus an equal number of NNLSQ optimizations) when first solved usually requires < 5 iterations on subsequent calls to degCloseInnFaster.m, which falls to 1-2 iterations for the last several adjustments of $\mu$. In general the approach is very effective.

We extend this idea in degCloseInnSearcher_hot.m by not only carrying over the ‘edge mask’ between iterations, but also by explicitly maintaining previous values of $x = \left[ \begin{array}{c} w \\ s \end{array} \right]$ for use as a starting point in our edge-addition algorithm. Additionally, when adding edges within each run of the edge-addition algorithm, each call to NNLSQ was warm-started using the previous value of $x$. The code for this inner warm-starting can be found in degCloseInnFaster_hot.m. Results were fairly positive for smaller graphs, with an average runtime improvement of about 15% for data-sets with fewer than 1000 vertices, including the IRIS, IONOSPHERE, PIMA, SONAR, and VEHICLE data sets initially tested in [DF9].

However, improvements were considerably less noticeable on large problems. For example, on the ABALONE data set (the largest data-set tested by Daitch et al.), we only saw runtime decrease by 6%. Improvements were even smaller for larger data-sets from the UCI Machine Learning Library, including MUSK ($n = 6598$, $d = 166$), ISOLET ($n = 6328$, $d = 618$), and DIGITS ($n = 10992$, $d = 16$). [FA10]

Using MATLAB’s built in code profiler, it became apparent that this decrease was due to the fact that, for large data-sets, the dominance of the NNLSQ optimization in the runtime of degCloseInnSearcher.m decreases. In fact, as $n$ increases, most time is spent computing the $\binom{n}{2}$ edge-derivatives required for each edge-addition and, more surprisingly, in constructing the objective function matrix, $M$. For example, the computation of $M$ and the edge derivatives accounts for nearly 44% of computation time in both the ABALONE and VEHICLE datasets. For the MUSK data-set, which contains 6,598 vectors with 166 dimensions each, the total increases to 94%.

2.4 Scaling with Problem Size

This finding suggests two possible improvements to the algorithm: 1) Build and maintain $M$ iteratively instead of recomputing it each time the quadratic program is solved. 2) When adding new edges to the program, it may be possible to find edges with large negative derivatives without consider all possible edges. Additionally, it may be possible to add natural pairs or sets of edges together, even if increasing their weights individually would not improve the objective function significantly.

The first improvement, while not implemented, would have the most significantly impact on the runtime of degCloseInnSearcher.m, since the computation of $M$ dominates the derivative computation as the size of the problem grows:
Additionally, with MATLAB’s sparse matrix library, improving the time and space efficiency of the $M$ computation should be fairly straightforward, if tedious. As we can see in the expanded representation of $M$ included in the background section, it is possible to simply add and remove columns as edges are included or excluded from the quadratic program.

While perhaps less practically important, the second possible improvement is a bit more interesting because it suggests that local connection properties of our graphs could help improve the runtime of the algorithm. If we could locally decide which edges are most useful, a global search across all possible edges might be unnecessary, at least in the early stages of the optimization.

### 2.5 Localilty

It is possible that local changes ‘ripple’ through large portions of a graph. For example, increasing the weight on a new candidate edge would likely require adjustments to weights on neighboring edges. Since optimal graphs are almost always connected, such changes could chain across long paths of edges. However, it still seems that changes might remain somewhat local, especially in less structured and less symmetric graphs. To test this intuition, consider roughly clustering a set of points before computing an optimal soft graph for each cluster. When the graph is computed for the entire set of points, ideally it will differ from the subgraphs only on the boundaries of the clusters, with most edges in the subgraphs remaining in the global graph. For a randomly generated set of vectors, rough clustering followed by local optimization might produce the following set of subgraphs, which are displayed along side the globally optimal graph:
Below we also include a plot indicating which edges are shared between the two plots above. Cyan indicates a shared edge while darker blue edges were only included in either the local or global optimum.

As expected, a large proportion of edges in the final graph where included in the local subgraphs and differences are concentrated near the borders of the subgraphs. In total, the graphs share 4,108 edges total, with 670 extra edges in the local graphs, and 886 new edges in the global graph. Finally, we include a plot indicating weight differences to confirms that changes are concentrated near subgraph borders. Dark blue indicates a weight difference of $> .05$ while black indicates a difference of $> .08$. 

Empirical evidence of locality largely motivated an interest in exploring the topic further. Specifically, as discussed in Section 3, understanding local properties of the graphs seems closely related to the question of whether or not the optimal graphs are unique. Additionally, the extension of ideas about sparseness and planarity to subgraphs might have implications for more intelligently choosing edges to add to our quadratic programs.

### 2.6 Final Caveat

Before moving on to a more theoretical discussion of the objective functions underlying these graphs, it is worth noting that numerical issues can significantly affect computation accuracy and runtime. For example, consider the soft graphs computed over 1000 random points below. The graph on the left was constructed using `degCloseInnSearcher.m` without rescaling; the points were left concentrated within the range $[0, 1]^2$ when optimizing. The graph on the right was generated using the same function, but after scaling each point by a factor of 10. Without scaling, numerical factors lead to a graph that is non-planar and contains over 10,000 low weight edges. On the other hand, the graph to the right contains just over 5,000 edges. Furthermore, since the size of our quadratic program depends on the number of edges with non-zero weight, computation time for the unscaled vectors was more than double the computation time for the scaled vectors, with `degCloseInnFaster.m` called 19 times instead of just 6.
3 Exploration of Uniqueness

3.1 Motivation for Conjecture

It was originally conjectured in [DKS09] that both the hard and soft objective functions would lead to unique graphs, at least under arbitrarily small point perturbations. It is necessary to assume general position because highly symmetric point sets do lead to multiple solutions. For example, the original paper introduces a simple example based on the vertices of the 5-dimensional hypercube. Specifically, it was observed that the set of points in \( \{0, 1\}^5 \) with even-parity leads to multiple solutions of the hard and \( \alpha \)-soft minimization problems. However, under arbitrarily small perturbations, this example and similar symmetric examples collapse and admit a single solution. Additionally, it seems that actual and randomly generated data-sets always result in unique solutions (fortunately, after solving the required quadratic program or nonnegative least squares problem, it is possible to check whether or not a solution is unique). For some more intuition about how symmetry can lead to non-uniqueness, I have included the simplest set of points I found that leads to multiple \( f(w) \) minimizing graphs:

Figure 7: Hard Graph in 2 Dimensions

(a) Solution 1

(b) Solution 2
However, the non-uniqueness is fragile, relying on the fact that the central point lies precisely on the diagonals of the square. Thus, each corner of the square is just as well approximated by a linear combination of its adjacent corners as it is by the center point alone. However, under small random perturbations (each point shifted < .01 in distance), the symmetry collapses and we are left with a single optimal graph:

Figure 8: Solution Under Perturbation

3.2 Sparseness vs. Symmetry

The question of uniqueness is closely related to ideas on sparseness developed in [DKS09]. Specifically, Daitch et al. demonstrate that every set of points has a hard and α-soft solution with < (d+1)n edges. It turns out that, for the many of the symmetric examples found (unfortunately, not including the simple one above), it is possible to demonstrate non-uniqueness by finding a dense symmetric solution that has > (d+1)n edges, and therefore differs from the sparse solution.

Daitch et al.’s proof relies on the fact, that when an optimal graph has greater than (d+1)n edges (w has ≥ (d+1)n non-zero elements) it should be possible to find some vector e with non-zero values restricted to locations where w is > 0 such that [M

\[ e = 0 \rightarrow Me = 0, Ae = 0. \]

This follows from the fact that [M

A

] has just dn + n rows and thus rank ≤ (d+1)n. In fact, we were recently able to tighten this rank bound, and thus decrease the upper bound on the minimum edge density of an optimal graph [Mus11]. Now, we can find some constant c such that w + ce ≥ 0, and specifically, if we set c = \[ \frac{\|w\|_{\infty}}{2}, \] w + ce equals zero for at least one element that was positive in w (where w/e is an element wise division). Now,

\[ \|M(w + ce)\|^2 = \|Mw + cMe\|^2 = \|Mw + 0\|^2 = \|Mw\|^2 \]

Thus, if w minimized our objective function, so does w + ce. Additionally, A(w + ce) = Aw = d, so the graph with weights w + ce has the same weighted degrees as the graph with weights w. Therefore, w + ce won’t violate any degree constraints, so we have a new solution with at least
one fewer edge. Since a sparser optimal solution can be constructed from any weight vector with \( \geq (d + 1)n \) positive entries, there must be some solution with \( < (d + 1)n \) positive entries.

### 3.3 Characterizing Multiple Solutions

We can extend this idea a bit further by arguing that any two optimal solutions must differ by some vector \( e \) such that \( Me = 0 \). This follows from the fact that the average of any two solution vectors will not violate our degree constraints, for either the hard or soft objective functions.

**Lemma 3.1.** For any two solutions \( w \) and \( \hat{w} \) of the hard or \( \alpha \)-soft optimization problem, \( \frac{w + \hat{w}}{2} \) describes a valid hard or \( \alpha \)-soft graph, in that \( A \left[ \frac{w + \hat{w}}{2} \right] \) does not violate the appropriate degree constraints.

**Proof.** Let \( Aw = d \) and \( A\hat{w} = \hat{d} \). Now, consider \( \frac{w + \hat{w}}{2} \). In the case of the hard graph, both \( d \) and \( \hat{d} \) must be \( \geq 1 \) and thus it follows trivially that \( A \left[ \frac{w + \hat{w}}{2} \right] = \frac{d + \hat{d}}{2} \geq 1 \).

Next, consider the soft graph constraint, \( \sum_{i=0}^{n} \left( \max(0, 1 - d_i) \right)^2 \leq \alpha n \). To simplify things, as explained in [DKS09], it must actually be that

\[
\sum_{i=0}^{n} \left( \max(0, 1 - d_i) \right)^2 = \alpha n
\]

at any optimal solution. Otherwise, we could uniformly scale down the solution without violating the constraint, which would strictly decrease \( f(\cdot) \). Now, let \( c = \max(0, d) \). So, for each \( i \in \{1, \ldots, n\}, c_i = \max(0, 1 - d_i) \). Similarly, let \( \hat{c} = \max(0, \hat{d}) \). By the equality above, \( ||c||^2 = ||\hat{c}||^2 = \alpha n \).

Consider the vector \( \hat{d} = \frac{d + \hat{d}}{2} = A \left[ \frac{w + \hat{w}}{2} \right] \), which is the degree vector corresponding to the graph with weights \( \frac{w + \hat{w}}{2} \). We would like to show that \( ||\hat{c}||^2 = ||\max(0, \hat{d})||^2 \leq \alpha n \), which follows from the fact that \( \hat{c} \leq (\hat{c} + \hat{c})/2 \).

Thus, each element of \( \hat{c} \) is less than or equal to the corresponding element in \( \frac{c + \hat{c}}{2} \) and so, since the vectors are non-negative, \( ||\hat{c}||^2 \leq ||\frac{c + \hat{c}}{2}||^2 \). Additionally, by the triangle inequality, \( ||\frac{c + \hat{c}}{2}||^2 \leq ||\frac{c}{2}|| + ||\frac{\hat{c}}{2}|| = \frac{\sqrt{\alpha n}}{2} + \frac{\sqrt{\alpha n}}{2} = \sqrt{\alpha n} \).

Thus, \( ||\frac{c + \hat{c}}{2}||^2 \leq \alpha n \rightarrow ||\hat{c}||^2 \leq \alpha n \) as required, and thus \( \hat{d} = \frac{d + \hat{d}}{2} \) doesn’t violate the \( \alpha \)-soft degree constraint.

**Theorem 3.2.** \( M \left( w - \hat{w} \right) = 0 \) for any two solutions \( w \) and \( \hat{w} \) of the hard or \( \alpha \)-soft optimization problems.

**Proof.** Consider \( \left( w + \hat{w} \right)/2 \). Again, by the triangle inequality:

\[
\left\| M \left( \frac{w + \hat{w}}{2} \right) \right\| \leq \left\| M \left( \frac{w}{2} \right) \right\| + \left\| M \left( \frac{\hat{w}}{2} \right) \right\| = \frac{1}{2} \left\| Mw \right\| + \frac{1}{2} \left\| M\hat{w} \right\| = \left\| Mw \right\|
\]

With equality only holding if \( Mw = c(M\hat{w}) \). Since both weight vertices are minimums, \( c \) must equal 1, so we see that, if \( Mw \neq M\hat{w} \), \( \left\| M \left( \frac{w + \hat{w}}{2} \right) \right\|^2 < \left\| Mw \right\|^2 = \left\| M\hat{w} \right\|^2 \). But, by Lemma 3.1, we
know that \( \frac{w + \hat{w}}{2} \) describes a valid hard or \( \alpha \)-soft graph and thus this would contradict the fact that \( w \) and \( \hat{w} \) are constrained minima of \( f(\cdot) \). So, it must be that \( Mw = M\hat{w} \) and thus \( M(w - \hat{w}) = 0 \). \( \square \)

3.4 Column Rank of \( M \)

The above finding suggests that the null-space, and thus the rank of \( M \) is essentially to the question of uniqueness. Specifically, if we can find a weight vector \( e \) in the null-space of \( M \) with negative values restricted to the edges in our current solution and positive or zero elements elsewhere, it is possible to move to an alternative solution. A significant amount of time was spent trying to understand why such vectors seem difficult to find in most graphs and work in this direction will be presented in an upcoming report with Cameron Musco. However, for now, it is worth noting that we can simplify our problem as follows: Consider the (possibly infinite) set of optimal solutions, \( \{w_1, w_2, \ldots\} \), that correspond to graphs with edges sets, \( \{E_1, E_2, \ldots\} \). As demonstrated above, we can average all possible solutions to find a new valid solution \( \bar{w} \) with edge set \( \{E_1 \cup E_2 \cup \ldots\} \). Now, consider any \( e \) in the null-space of \( M \) such that \( \bar{w} + e \) is also a minimum point. It must be that the non-zero values of \( e \) are restricted to edges in \( \bar{w} \) or else we could find a solution with an edge not contained in \( \{E_1 \cup E_2 \cup \ldots\} \). Thus, to demonstrate that the graphs are unique, it is only necessary to show that there is no solution, \( \bar{w} \), with non-zero elements corresponding to linearly dependent columns of \( M \). This avoids the difficult of dealing with possible positive elements on edges not in a solution.

4 Conclusion and Future Work

The next steps in this project will hinge on some recent progress in the uniqueness problem. Despite evidence to contrary, we were able to develop a counter example that holds up under small point perturbations, unlike other symmetric examples considered in the past. Future work will includes evaluating the conditions under which non-unique graphs appear and trying to understand how these conditions might suggest changes to our objective function or construction techniques.

References


