

THE METRIC NEARNESS PROBLEM*

JUSTIN BRICKELL[†], INDERJIT S. DHILLON[†], SUVRIT SRA[†], AND JOEL A. TROPP[‡]

Abstract. Metric nearness refers to the problem of optimally restoring metric properties to distance measurements that happen to be non-metric due to measurement errors or otherwise. Metric data can be important in various settings, for example in clustering, classification, metric-based indexing, query processing and graph theoretic approximation algorithms. This paper formulates and solves the *metric nearness problem*: Given a set of pairwise dissimilarities, find a “nearest” set of distances that satisfy the properties of a metric—principally the triangle inequality. For solving this problem, the paper develops efficient *triangle fixing* algorithms that are based on an iterative projection method. An intriguing aspect of the metric nearness problem is that a special case turns out to be equivalent to the All Pairs Shortest Paths problem. The paper exploits this equivalence and develops a new algorithm for the latter problem using a primal-dual method. Applications to graph clustering are provided as an illustration. We include experiments that demonstrate the computational superiority of triangle-fixing over general-purpose convex programming software. Finally, we conclude by suggesting various useful extensions and generalizations to metric nearness.

Key words. matrix nearness problems, metric, distance matrix, metric nearness, all pairs shortest paths, triangle inequality

AMS subject classifications. 05C12, 05C85, 54E35, 65Y20, 90C06, 90C08

1. Introduction. Most applications make some assumptions about the properties that the input data should satisfy. Due to measurement errors, noise, or an inability to gather data completely, an application may receive data that does not conform to its requirements. For example, imagine taking measurements as a part of some experiment. The theory suggests that the quantities measured should represent distance values amongst points in a discrete metric space. However, measurements being what they are, one ends up with a set of numbers that do not represent actual distance values, primarily because they fail to satisfy the triangle inequality. It might be beneficial to somehow optimally massage the measurements to obtain a set of “nearest” distance values that obey the properties of a metric.

It could also happen that experimental expenses and difficulties prevent one from making all the measurements. Before this incomplete set of measurements can be used in an application it might need to be tweaked, preferably minimally. As before, obtaining a “nearest” set of distance values (measurements) seems to be desirable.

Both scenarios above lead to the *metric nearness problem*: Given a set of input distances, find a “nearest” set of output distances that satisfy the properties of a metric. The notion of nearness is quantified by the function that measures distortion between the input and output distances.

Matrix nearness problems [10] offer a natural framework for pursuing the above-mentioned ideas. If there are n points, we may collect the measurements into an $n \times n$ symmetric matrix whose (j, k) entry represents the distance between points j and k . Then, we seek to approximate this matrix by another (say \mathbf{M}) whose entries satisfy the triangle inequalities. That is, $m_{ij} \leq m_{ik} + m_{kj}$ for every triple (i, k, j) . Any such matrix will represent the distances among n points in some metric space.

* This research was supported by NSF grant CCF-0431257, NSF Career Award ACI-0093404, and NSF-ITR award IIS-0325116. A preliminary version of this work appeared at NIPS 2004, Vancouver, Canada

[†]Dept. of Computer Sciences. The University of Texas at Austin. Austin, TX, 78712.

[‡]Dept. of Mathematics. University of Michigan at Ann Arbor. Ann Arbor, MI, 48109.

We calculate approximation error with a distortion measure that depends on how the corrected matrix should relate to the input matrix. For example, one might prefer to change a few entries significantly or to change all the entries a little. This paper considers metric nearness problems that use vector norms for characterizing distortion.

There is no analytic solution to the metric nearness problem. Fortunately this problem lends itself to a convex formulation, whereby developing algorithms for solving it becomes much easier. However, despite the natural convexity of the formulations, the large number of triangle inequality constraints can make traditional approaches or general purpose convex programming software much too slow. This paper provides solutions to the metric nearness problem that exploit its inherent structure for efficiency gains.

The remainder of this paper is structured as follows. Section 1.1 highlights the principal contributions of this paper. Section 2 develops a convex formulation of the metric nearness problem. Following that, Section 3 provides efficient triangle fixing algorithms for solving the metric nearness problems described in Section 2. An interesting connection of metric nearness with the All Pairs Shortest Paths (APSP) problem is studied in Section 4. This connection leads to a curious new primal-dual algorithm for APSP (Algorithm 4.3).

Applications of metric nearness to clustering are discussed in Section 5. Experiments highlighting running time studies and comparisons against the CPLEX software are given in Section 6.1, whereas experiments illustrating the behavior of the primal-dual metric nearness algorithm are the subject of Section 6.2.

Section 7.1 discusses some variations to the metric nearness problem that may also be studied. Section 7.2 describes possible future work and extensions to this paper, while two open problems are mentioned in Section 7.3. Finally, Section 7.4 summarizes related work and concludes this paper.

1.1. Contributions of this paper. In preliminary work [7], the authors presented the basic ideas about convex formulations of metric nearness and *triangle fixing* algorithms. However, many of the details necessary for understanding and actually implementing the triangle fixing algorithms were missing. This paper fills that gap by presenting a detailed derivation for ℓ_1 (consequently ℓ_∞), and ℓ_2 norm based metric nearness problems. Pseudocode for both the ℓ_1 and ℓ_2 problems is given along with the derivations.

When one allows only decreasing changes to the input, then metric nearness becomes equivalent to the All Pairs Shortest Paths (APSP) problem [22]. This paper studies this decrease-only version of metric nearness, and consequently obtains a new primal-dual algorithm for solving the APSP problem. This algorithm possesses some interesting characteristics related to its convergence behavior that are discussed in this paper.

The paper discusses applications to the MAX-CUT problem. We also developed efficient C++ code for metric nearness that outperforms CPLEX by factors of up to 30, and it may be requested from the authors.

2. Problem Formulation. We begin our formulation with a few basic definitions. We define a *dissimilarity matrix* to be a symmetric, nonnegative matrix with a zero diagonal. Such matrices are used to represent pairwise proximity data between objects of a certain type. A *distance matrix* is defined to be a dissimilarity matrix

whose entries satisfy the triangle inequalities. Specifically, \mathbf{M} is a distance matrix if

$$\begin{aligned} m_{ij} &\geq 0, & m_{ii} &= 0, & m_{ij} &= m_{ji}, \\ \text{and } m_{ij} &\leq m_{ik} + m_{kj} & \text{for distinct triples } &(i, k, j). \end{aligned}$$

We remark that symmetry, while part of the definition of a metric, is not crucial to our algorithms; asymmetry can be handled at the expense of doubling the running time and storage.

The distance matrices studied in this paper are assumed to arise from measuring inter-point distances between n points in a pseudo-metric space (i.e., two distinct points can lie at zero distance from each other). Consequently, distance matrices contain $N = \binom{n}{2}$ parameters, and we denote the set of all $n \times n$ distance matrices as \mathcal{M}_N . We observe that the set \mathcal{M}_N is a closed, convex polyhedral cone.

Assume that the input is a dissimilarity matrix \mathbf{D} . Metric nearness seeks a distance matrix \mathbf{M} that is closest to \mathbf{D} , with respect to some measure of ‘‘closeness.’’ Formally, we seek a matrix \mathbf{M} so that

$$\mathbf{M} \in \operatorname{argmin}_{\mathbf{X} \in \mathcal{M}_N} \|\mathbf{X} - \mathbf{D}\|, \quad (2.1)$$

where $\|\cdot\|$ is a norm. Though it is possible to use any norm in the metric nearness problem (2.1), we restrict our attention to the *vector* ℓ_p norms, wherein we treat the strict upper triangular part of our matrices as vectors.

THEOREM 2.1 (Attainment of minimum). *The functional $f(\mathbf{X}) = \|\mathbf{X} - \mathbf{D}\|$ always attains its minimum on \mathcal{M}_N . Moreover, every local minimum is a global minimum.*

Proof. The latter claim follows immediately from the convexity of f . It remains to show that $f(\mathbf{X})$ always attains its minimum on the cone \mathcal{M}_N . For convenience, we pass to the function $g(\mathbf{Y}) = \|\mathbf{Y}\|$. Notice that if g attains a minimum on $\mathcal{M}_N - \mathbf{D}$, then $f(\mathbf{X})$ attains a minimum on \mathcal{M}_N . The function g is a closed convex function, and it is homogeneous of degree one, so we can compute its recession function as

$$(g^0)^+(\mathbf{Y}) = \lim_{h \rightarrow 0} (g(h\mathbf{Y}) - g(\mathbf{0}))/h = \lim_{h \rightarrow 0} g(h\mathbf{Y})/h = g(\mathbf{Y}).$$

But g is non-negative, so its only directions of recession are directions in which it is constant. Since $\mathcal{M}_N - \mathbf{D}$ is a closed, *polyhedral* cone, we may apply [23, Theorem 27.3] to conclude that g attains a minimum on this cone, whereby f attains its minimum on \mathcal{M}_N . \square

2.1. Metric Nearness for the ℓ_2 norm. We start with a formulation for the vector ℓ_2 norm based metric nearness problem. Given the input dissimilarity matrix $\mathbf{D} = [d_{ij}]$ (where $d_{ij} = d_{ji}$), we wish to obtain a distance matrix \mathbf{X} that minimizes the squared error

$$\frac{1}{2} \sum_{i < j} (x_{ij} - d_{ij})^2.$$

Note that the sum above ranges over $i < j$, since the involved matrices are symmetric and have a zero diagonal.

Let T_n be the set of $3\binom{n}{3}$ triples, each of which corresponds to a triangle inequality that the entries of an $n \times n$ distance matrix must satisfy. Formally,

$$T_n = \{(i, j, k), (j, k, i), (k, i, j) : 1 \leq i < k < j \leq n\}, \quad (2.2)$$

where the triple (i, k, j) corresponds to the triangle inequality

$$x_{ij} \leq x_{ik} + x_{kj}.$$

With the introduction of an auxiliary matrix $\mathbf{E} = \mathbf{X} - \mathbf{D}$ that represents the changes to the original dissimilarities, the ℓ_2 metric nearness problem can be rewritten as the following quadratic program:

$$\text{Minimize}_{e_{ij}} \frac{1}{2} \sum_{i < j} e_{ij}^2, \quad (2.3)$$

$$\text{subject to } e_{ij} - e_{ik} - e_{kj} \leq d_{ik} + d_{kj} - d_{ij} = v_{ikj} \quad \forall (i, k, j) \in T_n. \quad (2.4)$$

The triangle inequality constraints are encoded by (2.4). Since the ℓ_2 norm is strictly convex, the solution to (2.3) is unique. The variable v_{ikj} quantifies the *violation* in the (i, k, j) triangle inequality. Note that non-negativity of x_{ij} need not be enforced explicitly as it is implied by the triangle inequalities.

2.2. Metric Nearness for the ℓ_1 and ℓ_∞ norms. When measuring approximation error using the ℓ_1 norm, we wish to minimize

$$\sum_{i < j} |e_{ij}|, \quad (2.5)$$

where $e_{ij} = x_{ij} - d_{ij}$ as in the previous section. However, to write the problem as a linear program, we need to introduce additional variables $f_{ij} = |e_{ij}|$. The resulting problem is the following linear program:

$$\text{Minimize}_{e_{ij}, f_{ij}} \sum_{i < j} (1 \cdot f_{ij} + 0 \cdot e_{ij}), \quad (2.6)$$

$$\text{subject to } \begin{array}{lll} e_{ij} - e_{ik} - e_{kj} & \leq & v_{ikj} \quad \forall (i, k, j) \in T_n, \\ -e_{ij} - f_{ij} & \leq & 0 \quad 1 \leq i < j \leq n, \\ e_{ij} - f_{ij} & \leq & 0 \quad 1 \leq i < j \leq n. \end{array} \quad (2.7)$$

The fact that $f_{ij} = |e_{ij}|$ is accomplished by the last two sets of inequalities in (2.7).

Similarly, for the ℓ_∞ nearness problem, we introduce a variable $\zeta = \max_{ij} |e_{ij}|$ that represents the vector ℓ_∞ norm of \mathbf{E} . The ℓ_∞ nearness problem becomes:

$$\text{Minimize}_{e_{ij}, \zeta} \zeta + \sum_{i < j} 0 \cdot e_{ij}, \quad (2.8)$$

$$\text{subject to } \begin{array}{lll} e_{ij} - e_{ik} - e_{kj} & \leq & v_{ikj} \quad \forall (i, k, j) \in T_n, \\ -e_{ij} - \zeta & \leq & 0 \quad 1 \leq i < j \leq n, \\ e_{ij} - \zeta & \leq & 0 \quad 1 \leq i < j \leq n. \end{array} \quad (2.9)$$

The last two sets of inequalities in (2.9) express the fact $|e_{ij}| \leq \zeta$ for all i and j .

2.3. Metric nearness for ℓ_p norms. Metric nearness may be easily formulated for ℓ_p norms, where $1 < p < \infty$. The problem is the following convex program:

$$\text{Minimize}_{e_{ij}} \frac{1}{p} \sum_{i < j} |e_{ij}|^p,$$

$$\text{subject to } e_{ij} - e_{ik} - e_{kj} \leq v_{ikj} \quad \forall (i, k, j) \in T_n.$$

Since the ℓ_p norms are strictly convex for $1 < p < \infty$, the associated metric nearness problems have unique solutions. There is a basic intuition for choosing p when solving the nearness problems. The ℓ_1 norm error is computed as the absolute sum of changes to the input matrix, while ℓ_∞ reflects only the maximum absolute change. The other ℓ_p norms interpolate between these two extremes. Thus, a small value of p typically results in a solution that prefers a few large changes to the original data, while a large p typically results in a solution with many small changes. In practice, however, the ℓ_1 , ℓ_2 , and ℓ_∞ problems are computationally easier to solve than those using arbitrary ℓ_p norms. Thus, we focus primarily on these three problems.

3. Triangle Fixing Algorithms. The previous section formulated the metric nearness problem as a quadratic program for the ℓ_2 norm, as a linear program for ℓ_1 and ℓ_∞ norms, and as a convex program for ℓ_p norms. Using off-the-shelf software for these formulations might appear to be an attractive way to solve the corresponding problems. However, it turns out that the computational time and storage requirements of such an approach can be prohibitive. An efficient algorithm must exploit the inherent structure offered by the triangle inequalities. In this section, we develop *triangle fixing algorithms*, which take advantage of this structure to efficiently solve the problem for ℓ_p norms. These algorithms iterate through the triangle inequalities, optimally enforcing any inequality that is not satisfied. While enforcing the triangle inequalities, one needs to introduce appropriate correction terms to guide the iterative algorithm to the globally optimal solution. The details are provided below.

3.1. Triangle fixing for ℓ_2 metric nearness. Our approach for solving (2.3) is iterative, and is based on the technique described in [2]. Collecting all the e_{ij} values into vector \mathbf{e} and the violation amounts v_{ijk} into \mathbf{v} , problem (2.3) may be rewritten as

$$\begin{aligned} \min_{\mathbf{e}} \quad & \frac{1}{2} \mathbf{e}^T \mathbf{e} \\ \text{subject to} \quad & \mathbf{A} \mathbf{e} \leq \mathbf{v}, \end{aligned} \tag{3.1}$$

where matrix \mathbf{A} encodes the triangle inequalities (2.4), whereby, each row of \mathbf{A} has one +1 entry, and two -1 entries.

The Lagrangian of (3.1) is

$$L(\mathbf{e}, \mathbf{z}) = \frac{1}{2} \mathbf{e}^T \mathbf{e} + \langle \mathbf{z}, \mathbf{A} \mathbf{e} - \mathbf{v} \rangle,$$

where \mathbf{z} is the dual vector. A necessary condition for optimality of (3.1) is

$$\frac{\partial L}{\partial \mathbf{e}} = 0 \quad \implies \quad \mathbf{e} = -\mathbf{A}^T \mathbf{z}, \quad \mathbf{z} \geq \mathbf{0}. \tag{3.2}$$

Using (3.2) we see that the dual problem corresponding to (3.1) is

$$\max_{\mathbf{z} \geq \mathbf{0}} \quad g(\mathbf{z}) = -\frac{1}{2} \mathbf{z}^T \mathbf{A} \mathbf{A}^T \mathbf{z} - \mathbf{z}^T \mathbf{v}. \tag{3.3}$$

We solve (3.1) iteratively, wherein we initialize both \mathbf{e} and \mathbf{z} to zero as this choice satisfies (3.2). At each subsequent iteration we update the dual vector \mathbf{z} one coordinate at a time, thereby resulting in a *dual coordinate ascent* procedure, while maintaining (3.2). Assume that the dual variable corresponding to inequality (i, k, j) is updated, i.e., $z'_{ikj} = z_{ikj} + \theta$. Then, the corresponding update to the primal is obtained via (3.2), i.e., $\mathbf{e}' = \mathbf{e} - \theta \mathbf{a}_{ikj}$ (using the fact that $\mathbf{e}' = -\mathbf{A}^T \mathbf{z}'$), where \mathbf{a}_{ikj} is

Algorithm 3.1: Metric Nearness for ℓ_2 norm.

```

METRIC_NEARNESS_L2( $\mathbf{D}$ ,  $\kappa$ )
Input: Dissimilarity matrix  $\mathbf{D}$ , tolerance  $\kappa$ 
Output:  $\mathbf{M} = \operatorname{argmin}_{\mathbf{X} \in \mathcal{M}_N} \|\mathbf{X} - \mathbf{D}\|_2$ .
{Initialize the primal and the dual variables}
 $e_{ij} \leftarrow 0$  for  $1 \leq i < j \leq n$ 
 $(z_{ijk}, z_{jki}, z_{kij}) \leftarrow 0$  for  $1 \leq i < k < j \leq n$ 
 $\delta \leftarrow 1 + \kappa$ 
while ( $\delta > \kappa$ )      {convergence test}
  foreach triangle inequality  $(i, k, j)$ 
     $v \leftarrow d_{ik} + d_{kj} - d_{ij}$                                 {Compute violation}
     $\theta^* \leftarrow \frac{1}{3}(e_{ij} - e_{ik} - e_{kj} - v)$                 (*)
     $\theta \leftarrow \max\{\theta^*, -z_{ikj}\}$                             {Stay within half-space of constraint}
     $e_{ij} \leftarrow e_{ij} - \theta, e_{ik} \leftarrow e_{ik} + \theta, e_{kj} \leftarrow e_{kj} + \theta$   (**)
     $z_{ikj} \leftarrow z_{ikj} + \theta$                                 {Update dual variable}
  end foreach
   $\delta \leftarrow$  sum of changes in the  $e_{ij}$  values
end while
return  $\mathbf{M} = \mathbf{D} + \mathbf{E}$ 

```

the column vector containing the entries of the (i, j, k) row of \mathbf{A} . Recall that \mathbf{a}_{ikj} has only three non-zero entries corresponding to the edges (i, j) , (i, k) and (k, j) . Thus, the update to \mathbf{e} amounts to “fixing” (enforcing) one triangle inequality at a time, hence the name of our procedure. The parameter θ is computed by solving

$$\begin{aligned} & \max_{\theta} g(\mathbf{z} + \theta \mathbf{1}_{ikj}), \\ & \text{subject to } z_{ikj} + \theta \geq 0, \end{aligned} \quad (3.4)$$

where $\mathbf{1}_{ikj}$ indicates the standard basis vector that is zero in all positions except the ikj entry, which equals unity. Using (3.2) and (3.3), we may rewrite (3.4) as

$$\begin{aligned} & \max_{\theta} g(\mathbf{z}) - \frac{1}{2} \|\mathbf{a}_{ikj}\|^2 \theta^2 + (\mathbf{a}_{ikj}^T \mathbf{e} - v_{ikj}) \theta, \\ & \text{subject to } \theta \geq -z_{ikj}. \end{aligned} \quad (3.5)$$

Consider optimizing (3.5) in an unconstrained manner. It is easily seen that

$$\theta^* = \frac{1}{\|\mathbf{a}_{ikj}\|^2} (\mathbf{a}_{ikj}^T \mathbf{e} - v_{ikj}) = \frac{1}{3} (\mathbf{a}_{ikj}^T \mathbf{e} - v_{ikj}), \quad (3.6)$$

is the maximum. If $\theta^* \geq -z_{ikj}$ we are done, else the maximum of (3.5) will be achieved at $\theta = -z_{ikj}$. Thus, we obtain $\theta = \max\{\theta^*, -z_{ikj}\}$ as the answer to (3.5). Algorithm 3.1 puts together all these ideas to give the complete iterative triangle fixing procedure.

Remarks. The procedure derived above ensures that at each iteration $g(\mathbf{z}') \geq g(\mathbf{z})$, i.e., it is a *dual coordinate ascent* procedure. Following [2], it can be shown that in the limit, the $\mathbf{A}\mathbf{e} \leq \mathbf{v}$ constraints are satisfied. Since (3.2) is also maintained at each step, the KKT conditions, which are necessary and sufficient for this problem, are satisfied in the limit. Thus, the triangle fixing procedure converges to the optimal solution of (3.1). In fact, Algorithm 3.1 is an efficient version of Bregman’s method for minimizing a convex function subject to linear inequality constraints [2]. Our algorithm exploits the structure of the problem to obtain its efficiency.

3.2. Triangle fixing for ℓ_1 and ℓ_∞ . Triangle fixing is somewhat less direct for the ℓ_1 and ℓ_∞ problems. The reason these norms pose an additional challenge is because they are not strictly convex; the convergence of the basic triangle fixing procedure depends on the strict convexity of the norm used. We illustrate only the ℓ_1 case; the development for ℓ_∞ takes the same course.

With the introduction of vector and matrix notation, the ℓ_1 matrix nearness problem may be rewritten as

$$\begin{aligned} \min_{\mathbf{e}, \mathbf{f}} \quad & \mathbf{0}^T \mathbf{e} + \mathbf{1}^T \mathbf{f}, \\ \text{subject to} \quad & \mathbf{A}\mathbf{e} \leq \mathbf{v}, \quad -\mathbf{e} - \mathbf{f} \leq \mathbf{0}, \quad \mathbf{e} - \mathbf{f} \leq \mathbf{0}. \end{aligned} \tag{3.7}$$

The auxiliary variable \mathbf{f} is interpreted as the element-wise absolute value of \mathbf{e} . The violations to the triangle inequalities are again given by the vector \mathbf{v} .

To solve the linear program (3.7) without sacrificing the advantages of triangle fixing we replace it with an equivalent quadratic program. This replacement hinges upon a connection between linear and quadratic programs that may be motivated by the observation,

$$\operatorname{argmin}_{\mathbf{g}} \|\mathbf{g} + \epsilon^{-1}\mathbf{c}\|^2 = \operatorname{argmin}_{\mathbf{g}} (\mathbf{g}^T \mathbf{g} + 2\epsilon^{-1}\mathbf{g}^T \mathbf{c} + \epsilon^{-2}\mathbf{c}^T \mathbf{c}) \approx \operatorname{argmin}_{\mathbf{g}} \mathbf{g}^T \mathbf{c},$$

if ϵ is chosen to be sufficiently small (so that the $2\epsilon^{-1}\mathbf{g}^T \mathbf{c}$ term dominates the objective function). The following theorem, which follows from a result of [17, Theorem 2.1-a-i], makes the above connection concrete.

THEOREM 3.1 (ℓ_1 Metric Nearness). *Let $\mathbf{g} = [\mathbf{e}; \mathbf{f}]$ and $\mathbf{c} = [\mathbf{0}; \mathbf{1}]$ be partitioned conformally. If (3.7) has a solution, then there exists an $\epsilon_0 > 0$, such that for all $\epsilon \leq \epsilon_0$,*

$$\operatorname{argmin}_{\mathbf{g} \in G} \|\mathbf{g} + \epsilon^{-1}\mathbf{c}\|_2 = \operatorname{argmin}_{\mathbf{g} \in G^*} \|\mathbf{g}\|_2, \tag{3.8}$$

where G is the feasible set for (3.7) and G^* is the set of optimal solutions to (3.7). The minimizer of (3.8) is unique.

From (3.7) one can see that the triangle inequality constraints involve only \mathbf{e} and not \mathbf{f} . This circumstance permits us to use triangle fixing once again. As before, we go through the constraints one by one. The first $3\binom{n}{3}$ constraints are triangle constraints and are handled by triangle fixing. The remaining $2\binom{n}{2}$ absolute value constraints are very simple and thus are enforced easily.

For the ℓ_2 case, the dual variables (corresponding to each constraint) were represented by the vector \mathbf{z} . For (3.8), we let the dual variables be $[\mathbf{z}; \boldsymbol{\lambda}; \boldsymbol{\mu}]$; vector \mathbf{z} corresponds to the triangle inequalities, while vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ correspond to $-\mathbf{e} - \mathbf{f} \leq \mathbf{0}$ and $\mathbf{e} - \mathbf{f} \leq \mathbf{0}$, respectively. Together, non-negative values of \mathbf{z} , $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ correspond to the feasible set G alluded to by Theorem 3.1.

Our augmented triangle-fixing procedure is as follows. First we initialize \mathbf{e} , \mathbf{f} , \mathbf{z} , $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ so that the first order optimality conditions derived from (3.8) are initially true. Thereafter, we enforce constraints one by one to ensure the the dual functional corresponding to (3.8) is increasing and first order optimality conditions are maintained. Written out as Algorithm 3.2, this procedure becomes an efficient adaptation of Bregman's method, thereby, after a sufficient number of iterations, it converges to the globally optimal solution.

Algorithm 3.2: Metric nearness for ℓ_1 norm.

```

METRIC_NEARNESS_L1( $\mathbf{D}$ ,  $\epsilon$ ,  $\kappa$ )
Input: Dissimilarity matrix  $\mathbf{D}$ ; tolerance  $\kappa$ ;  $\ell_1$  parameter  $\epsilon$ 
Output:  $\mathbf{M} \in \{\operatorname{argmin}_{\mathbf{X} \in \mathcal{M}_N} \|\mathbf{X} - \mathbf{D}\|_1\}$ 
{Initialize primal & dual variables}
 $e_{ij} \leftarrow 0$ ;  $f_{ij} = -\epsilon^{-1}$  for  $1 \leq i < j \leq n$       {Primal variables}
 $(z_{ijk}, z_{jki}, z_{kij}) \leftarrow 0$  for  $1 \leq i < k < j \leq n$   {Dual variables – triangles}
 $\lambda_{ij} \leftarrow \pi_{ij} \leftarrow 0$  for  $1 \leq i < j \leq n$       {Dual variables – Other}
 $\delta \leftarrow 1 + \kappa$ 
while ( $\delta > \kappa$ )      {convergence test}
  Do triangle fixing on the  $e_{ij}$  as in Algorithm 3.1
  {Enforce  $-\mathbf{e} - \mathbf{f} \leq \mathbf{0}$  and  $\mathbf{e} - \mathbf{f} \leq \mathbf{0}$  as follows}
   $\boldsymbol{\mu} \leftarrow \frac{1}{2}(\mathbf{e} + \mathbf{f})$       {Projection parameters}
   $\boldsymbol{\theta} \leftarrow \min\{\boldsymbol{\mu}, \boldsymbol{\lambda}\}$       {Update amount}
   $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \boldsymbol{\theta}$       {Update dual vector corr. to  $-\mathbf{e} - \mathbf{f} \leq \mathbf{0}$ }
   $\mathbf{e} \leftarrow \mathbf{e} - \boldsymbol{\theta}$ ;  $\mathbf{f} \leftarrow \mathbf{f} - \boldsymbol{\theta}$  {Update primal variables}
   $\boldsymbol{\nu} \leftarrow \frac{1}{2}(\mathbf{f} - \mathbf{e})$ 
   $\boldsymbol{\theta} \leftarrow \min\{\boldsymbol{\nu}, \boldsymbol{\pi}\}$       {Update amount}
   $\boldsymbol{\pi} \leftarrow \boldsymbol{\pi} - \boldsymbol{\theta}$       {Update dual vector corr. to  $\mathbf{e} - \mathbf{f} \leq \mathbf{0}$ }
   $\mathbf{e} \leftarrow \mathbf{e} + \boldsymbol{\theta}$ ;  $\mathbf{f} \leftarrow \mathbf{f} - \boldsymbol{\theta}$  {Update primal variables}
  {Update convergence test parameter}
   $\delta \leftarrow$  sum of absolute changes in  $e_{ij}$ .
end.

```

Remarks. Algorithm 3.2 depends on the parameter ϵ that governs convergence to the true optimal solution. It is an open problem to obtain an ϵ that guarantees convergence. However, upon experimentation with random dissimilarity matrices we found that setting $\epsilon^{-1} \approx \max_{ij} d_{ij}$, worked well, i.e., led to convergence, for Algorithm 3.2. Furthermore, from Theorem 3.1 we know that there exists a range within which ϵ can lie, and in practice running Algorithm 3.2 a small number (2–3) of times (with early stopping to save time) helps to determine a suitable value for ϵ for an arbitrary input matrix.

3.3. Triangle fixing for other ℓ_p norms. We can go a step further and extend triangle fixing to solve the metric nearness problem for all ℓ_p ($1 < p < \infty$) norms. The problem may be compactly stated as

$$\min_{\mathbf{e}} \frac{1}{p} \|\mathbf{e}\|_p^p \quad \text{subject to} \quad \mathbf{A}\mathbf{e} \leq \mathbf{v}. \quad (3.9)$$

Recall that for ℓ_2 metric nearness, at each iterative step we obtained \mathbf{e}' from \mathbf{e} by solving (3.2) after updating the dual variables \mathbf{z} in a single coordinate. This update to \mathbf{e} may be viewed as the result of an orthogonal projection of \mathbf{e} onto the hyperplane defined by $\langle \mathbf{a}_{ikj}, \mathbf{e}' \rangle = v_{ikj}$ (ignoring inequalities for the moment). For the ℓ_p norm problem, we must instead perform a generalized projection, called a *Bregman projection*, which involves solving the following problem

$$\min_{\mathbf{e}'} \varphi(\mathbf{e}') - \varphi(\mathbf{e}) - \langle \nabla \varphi(\mathbf{e}), \mathbf{e}' - \mathbf{e} \rangle \quad \text{such that} \quad \langle \mathbf{a}_{ikj}, \mathbf{e}' \rangle = v_{ikj}, \quad (3.10)$$

where $\varphi(\mathbf{x}) = \frac{1}{p} \|\mathbf{x}\|_p^p$. We use $(\nabla \varphi(\mathbf{x}))_i = \operatorname{sgn}(x_i) |x_i|^{p-1}$ to determine the projection (3.10) by solving

$$\nabla \varphi(\mathbf{e}') = \nabla \varphi(\mathbf{e}) + \boldsymbol{\mu} \mathbf{a}_{ikj} \quad \text{so that} \quad \langle \mathbf{a}_{ikj}, \mathbf{e}' \rangle = v_{ikj}. \quad (3.11)$$

Since \mathbf{a}_{ikj} has only three nonzero entries, once again \mathbf{e} needs to be updated in only three components. Therefore, in Algorithm 3.1 we may replace (\star) by an appropriate numerical computation of the parameter μ , and replace $(\star\star)$ by the computation of the new value of \mathbf{e} as resulting from (3.11). As before, each iteration maintains the necessary condition $\partial L(\mathbf{e}, \mathbf{z})/\partial \mathbf{e} = 0$ while correcting the dual vector \mathbf{z} , and the overall algorithm converges to the optimum of (3.9).

4. Metric Nearness and APSP. The All Pairs Shortest Paths (APSP) problem [3] is an important and well-studied problem in graph theory that still continues to interest researchers. For a given weighted graph G , APSP computes an associated matrix of distances M whose entry m_{ij} gives the weight of a shortest path between vertices i and j . Optionally, shortest paths between all pairs of vertices corresponding to these distances are also obtained.

On the surface, APSP appears to have no connection with the metric nearness problem. However, it turns out that APSP can be viewed as a special case of metric nearness. We develop this connection below. Note that in the previous sections we considered only symmetric matrices. However, in this section we consider asymmetric distance matrices, which are more natural for the APSP problem, as they correspond to directed graphs.

4.1. The relation of Metric Nearness to APSP. Let the input be a weighted complete directed graph. We represent this graph by the (non-symmetric) matrix \mathbf{D} , where d_{ij} denotes the edge weight of edge (i, j) . On \mathbf{D} we perform a restricted version of metric nearness that permits only decreasing changes to the d_{ij} values. Curiously this decrease only version of metric nearness is equivalent to APSP.

LEMMA 4.1 (Decrease only metric nearness is APSP). *Let $\mathbf{M}^A \in \mathcal{M}_N$ be the APSP solution for \mathbf{D} . Then, \mathbf{M}^A is also the nearest “decrease only” metric solution. In fact, any metric solution $\mathbf{M} \in \mathcal{M}_N$ that is element-wise smaller than \mathbf{D} , is also smaller than \mathbf{M}^A , i.e., $\forall \mathbf{M} \in \mathcal{M}_N$, if $\mathbf{M} \leq \mathbf{D}$ then $\mathbf{M} \leq \mathbf{M}^A$.*

A proof of this lemma may be found in Appendix A.1. This connection between APSP and decrease only metric nearness (DOMN) suggests that the latter may be solved by using any off-the-shelf algorithm for APSP. More interestingly, one can turn the problem around and obtain a new method to solve APSP by solving the DOMN problem. In this section, we present a new algorithm for APSP based on solving a linear programming formulation of DOMN.

APSP for dense graphs is commonly performed using the Floyd-Warshall algorithm, which has a complexity of $\Theta(n^3)$. Unlike the Floyd-Warshall algorithm that proceeds by fixing the triangles of the graph in a predetermined order, our DOMN algorithm fixes triangles in a data-dependent order. Empirically, our algorithm converges more quickly to the solution than Floyd-Warshall, despite having the same asymptotic worst-case behavior.

4.2. The linear programming formulation of DOMN and its dual.

Lemma 4.1 suggests that APSP solves the decrease only metric nearness problem regardless of the norm used to measure the error. We, however, focus on the ℓ_1 norm problem along with its linear programming formulation. The linear program is interesting both because it is a novel formulation for solving APSP, and also because its dual allows us to construct shortest paths, if desired. We apply the primal-dual technique for solving the resulting linear programs and obtain a new APSP algorithm as a consequence.

4.2.1. Formulation. Let \mathbf{X} represent a decrease-only distance matrix corresponding to the input matrix \mathbf{D} . Then the entries of \mathbf{X} must satisfy,

$$x_{ij} \leq d_{ij} \quad \text{for all } (i, j), \quad (4.1)$$

$$x_{ij} \leq x_{ik} + x_{kj} \quad \text{for all } (i, k, j). \quad (4.2)$$

Finding the matrix with the least ℓ_1 perturbation requires solving the problem

$$\underset{x_{ij}}{\text{minimize}} \quad \sum_{ij} (d_{ij} - x_{ij}) \quad \text{subject to (4.1) and (4.2).}$$

Note that we are dealing directly with the values x_{ij} rather than the error values $e_{ij} = d_{ij} - x_{ij}$, as we did in sections 2.1 and 2.2. Since the d_{ij} are fixed we may replace this minimization by the equivalent problem

$$\begin{aligned} & \underset{x_{ij}}{\text{maximize}} \quad \sum_{ij} x_{ij} \\ & \text{subject to} \quad \begin{aligned} x_{ij} & \leq d_{ij} & \text{for all } (i, j), \\ x_{ij} - x_{ik} - x_{kj} & \leq 0 & \text{for all } (i, k, j). \end{aligned} \end{aligned} \quad (4.3)$$

The dual problem corresponding to (4.3) is

$$\begin{aligned} & \underset{\pi_{ij}}{\text{minimize}} \quad \sum_{ij} \pi_{ij} d_{ij} \\ & \text{subject to} \quad \begin{aligned} \pi_{ij} + \sum_{k \neq i, j} (\gamma_{ikj} - \gamma_{ijk} - \gamma_{kij}) & = 1 & \text{for all } (i, j), \\ \pi_{ij} & \geq 0 & \text{for all } (i, j), \\ \gamma_{ikj} & \geq 0 & \text{for all } (i, k, j), \end{aligned} \end{aligned} \quad (4.4)$$

where the dual variables π_{ij} and γ_{ikj} correspond to the decrease-only constraints (4.1), and the triangle inequality constraints (4.2), respectively.

It is illustrative to cast the linear program (4.4) as a network flow problem, in which we must satisfy a demand for a single unit of flow between every pair of vertices i and j . We can accomplish this either by sending the flow *directly* via the edge (i, j) (which corresponds to setting $\pi_{ij} = 1$) or by *bypassing* the edge (i, j) and routing through some other vertex k (which corresponds to setting $\gamma_{ikj} = 1$); in the latter case, we increase the demand for flow between (i, k) and (k, j) by 1.

We note that while there is a unique optimal solution to the linear program (4.3), the linear program (4.4) has several optimal solutions, some of which involve non-integral assignments to the γ_{ikj} variables. This non-uniqueness is not unexpected, because while there is only one value that the shortest distance between two nodes in M can attain, whereas there can be several shortest paths that achieve this distance value (paths which may contain many intermediate nodes, each of which allows a γ_{ikj} variable to assume a positive assignment).

4.3. A primal-dual algorithm for DOMN/APSP. We apply the primal-dual method [19, 16] to solve the linear programs for DOMN, and thereby obtain a new algorithm for APSP. Most treatments of the primal-dual method have a minimization of the primal problem and a maximization of the dual problem. Thus we will call (4.3) as the dual problem, and (4.4) as the primal problem. The primal-dual method begins with a feasible solution to the dual that is improved at each step by optimizing an *associated restricted primal* problem. In our case, we find it easier to optimize the associated restricted dual, whereby our method proceeds as follows:

1. Begin with a feasible solution to the dual problem. One such feasible solution is to set each x_{ij} to the smallest d_{ij} value.
2. Find the set P consisting of those constraints that do not have any additional slack. The decrease-only constraint $x_{ij} \leq d_{ij}$ (corresponding to dual variable π_{ij}) will be in P iff $x_{ij} = d_{ij}$, and the triangle constraint $x_{ij} - x_{ik} - x_{kj} \leq 0$ (corresponding to dual variable γ_{ikj}) will be in P iff $x_{ij} = x_{ik} + x_{kj}$.
3. Find a solution to the associated restricted dual

$$\begin{aligned}
& \text{maximize} && \sum_{ij} u_{ij} \\
& \text{subject to} && u_{ij} \leq 0 \quad \text{if } \pi_{ij} \in P \\
& && u_{ij} - u_{ik} - u_{kj} \leq 0 \quad \text{if } \gamma_{ikj} \in P \\
& && u_{ij} \leq 1 \quad \text{for all } (i, j)
\end{aligned} \tag{4.5}$$

The solution u_{ij} to the associated restricted dual identifies which variables can be increased while maintaining dual feasibility.

4. If $u_{ij} = 0$ then the current value of x_{ij} is the optimal dual variable assignment. Otherwise, improve the x_{ij} assignment by adding ϵu_{ij} to x_{ij} , where ϵ is as large as possible while still maintaining dual feasibility. Return to Step 2 with the new x_{ij} assignment.

By characterizing the solution of the associated restricted dual and the calculation of ϵ for the DOMN problem, we can give an efficient primal-dual algorithm. Observe that the solution to the associated restricted dual is that $u_{ij} = 1$ if the edge (i, j) is *increasable*, and 0 otherwise. Computing ϵ is equivalent to determining which of the increasable edges has the least capacity for increase. Rather than use linear programming to determine the increasable edge set and computing ϵ explicitly, we can track upper bounds u_{ij} in addition to the lower bounds tracked by the x_{ij} variables. These upper bounds start as the d_{ij} values, but are reduced as edges become triangle constrained. Then the increasable set is simply the set of edges for which $x_{ij} < u_{ij}$, and ϵ is the difference between the lower bound of edges in the increasable set and the largest upper bound. Algorithm 4.3 implements these optimizations. I , the set of increasable edges, is the complement of P .

4.4. Priority Queue DOMN Algorithm. Algorithm 4.3 requires $O(n^4)$ time, but we can do better by noticing that the only time the lower bounds are used is to check the condition $u_e = l_e$. For edges (i, j) not in I , we have $u_{ij} = l_{ij}$, whereas all edges (i, j) in I have l_{ij} equal to the smallest upper bound. Therefore, we can replace I with a priority queue ordered by upper bound, and we do not need to keep track of lower bounds at all (even though the original dual variable values x_{ij} were lower bounds). Algorithm 4.4 implements these changes, and requires only $O(n^3)$ time when implemented using a Fibonacci heap. Like the Floyd-Warshall algorithm, Algorithm 4.4 considers all edges in some order, and then fixes all triangles involving that edge. However, the Floyd-Warshall algorithm used a fixed data-independent order, whereas our algorithm uses a data-dependent order. As a result, our algorithm converges to the APSP/DOMN solution more rapidly, even though it still requires $O(n^3)$ time to complete.

5. An Application to Clustering. The metric nearness problem can be used to develop efficient algorithms for clustering that provide guarantees on the quality of the output in comparison with the optimal clustering. The MAX-CUT problem offers an especially attractive example. A *cut* of a graph is a partition of the vertices into two disjoint sets, and the value of a cut is the total weight of all edges that cross the

Algorithm 4.3: Decrease-Only Metric Nearness: Simple $O(n^4)$ implementation

```

DOMN_ALG1( $\mathbf{D}$ )
Input: Dissimilarity matrix  $\mathbf{D}$ 
Output:  $\mathbf{M} = APSP(\mathbf{D})$ .
{Initialization}
 $u_{ij} \leftarrow d_{ij}$  for all  $i, j$                                 {Initial upper bounds}
 $x_{ij} \leftarrow \min_{e' \in E} u_{e'}$                                 {Initial lower bounds}
 $I \leftarrow E$                                                 {Initial set of increasable edges}
while ( $I \neq \emptyset$ )
  foreach  $(i, j) \in I$  with  $u_{ij} = x_{ij}$ 
     $I \leftarrow I - \{(i, j)\}$                                  $\{(i, j)$  is no longer increasable}
    foreach  $k \neq i, j$ 
       $u_{ik} = \min(u_{ik}, u_{ij} + u_{jk})$                     {Update upper bounds}
    end foreach
  end foreach
  foreach  $(i, j) \in I$ 
     $x_{ij} \leftarrow \min_{e' \in I} u_{e'}$                         {Update lower bounds}
  end foreach
end while
return  $\mathbf{M}$  where  $m_{ij} = x_{ij}$ 

```

Algorithm 4.4: Decrease-Only Metric Nearness: Improved $O(n^3)$ Implementation

```

DOMN_ALG2( $\mathbf{D}$ )
Input: Dissimilarity matrix  $\mathbf{D}$ 
Output:  $\mathbf{M} = APSP(\mathbf{D})$ .
{Initialization}
 $u_{ij} \leftarrow d_{ij}$  for all  $i, j$                                 {Initial upper bounds}
 $Q.ENQUEUE((i, j), u_{ij})$  for all  $(i, j)$                     {Put all edges in priority-queue}
while ( $Q \neq \emptyset$ )
   $(i, j) \leftarrow Q.FIRST()$                                 {Remove edge with lowest upper bound}
  foreach  $k \neq i, j$ 
     $u_{ik} = \min(u_{ik}, u_{ij} + u_{jk})$                     {Update upper bounds}
     $Q.UPDATEPRIORITY((i, k), u_{ik})$                     {Reorder priority queue}
  end foreach
end while
return  $\mathbf{M}$  where  $m_{ij} = u_{ij}$ 

```

partition. MAX-CUT simply asks for the cut of a graph with maximum value. If the size of each edge weight is proportional to the dissimilarity between the two vertices, solving MAX-CUT can be interpreted as finding the best clustering of the vertices into two sets.

For a general set of weights, MAX-CUT is hard enough [20] that the solution cannot be well-approximated in polynomial time (unless $P = NP$) [1]. On the other hand, for weights that do satisfy the triangle inequality, de la Vega and Kenyon have exhibited a randomized algorithm that can approximate the solution arbitrarily well in polynomial time [5]. That is, for a given $\varepsilon > 0$, their method can (with high probability) compute in polynomial time, a cut whose value is no smaller than $(1 - \varepsilon)$ times the value of the optimal cut. Of course, the time complexity grows quickly as

ε shrinks.

Metric nearness plays an important role here. First, we approximate the original graph by a metric graph. Then, we use the fast algorithm to produce a nearly optimal cut of the metric graph. The same cut of the original graph also has a nearly optimal value, which can be bounded in terms of the approximation error from the metric nearness problem.

THEOREM 5.1. *Suppose that \mathbf{D} is a dissimilarity matrix and that \mathbf{M} is a distance matrix. If \mathcal{S} is a cut of \mathbf{M} whose value exceeds $(1 - \varepsilon) \text{maxcut}(\mathbf{M})$, then we have the bounds*

$$\text{cut}_{\mathcal{S}}(\mathbf{D}) \geq (1 - \varepsilon) \text{maxcut}(\mathbf{D}) - (1 - \frac{\varepsilon}{2}) \|\mathbf{M} - \mathbf{D}\|_1 \quad \text{and} \quad (5.1)$$

$$\text{cut}_{\mathcal{S}}(\mathbf{D}) \geq \frac{1 - \varepsilon}{\|\mathbf{M}/\mathbf{D}\|_{\infty} \|\mathbf{D}/\mathbf{M}\|_{\infty}} \text{maxcut}(\mathbf{D}), \quad (5.2)$$

where ‘/’ represents element-wise division and $\|\cdot\|_{\infty}$ denotes the ℓ_{∞} norm that ignores the matrix diagonal. If $m_{jk} = d_{jk} = 0$, then the infinity norm also ignores the (j, k) entry of its argument.

To find the optimal \mathbf{M} for bound (5.1), we simply solve the ℓ_1 metric nearness problem. The optimal \mathbf{M} for (5.2) cannot be obtained without solving a non-convex optimization problem.

Proof. For a set of vertices \mathcal{S} , the value of the corresponding cut is computed by the linear function

$$\text{cut}_{\mathcal{S}}(\mathbf{D}) = \sum_{j \in \mathcal{S}} \sum_{k \notin \mathcal{S}} d_{jk}.$$

The maximum cut just optimizes this functional over all subsets \mathcal{S} of the vertex set $\{1, 2, \dots, n\}$:

$$\text{maxcut}(\mathbf{D}) = \max_{\mathcal{S}} \sum_{j \in \mathcal{S}} \sum_{k \notin \mathcal{S}} d_{jk}.$$

Obviously, $\text{cut}_{\mathcal{S}}(\mathbf{D}) \leq \text{maxcut}(\mathbf{D})$. It can be shown that $\text{maxcut}(\|\cdot\|)$ is a matrix norm. In particular, it satisfies the triangle inequality for norms. It is also clear that

$$\text{maxcut}(\|\mathbf{T}\|) \leq \frac{1}{2} \sum_{j \neq k} |t_{jk}| = \frac{1}{2} \|\mathbf{T}\|_1$$

for any symmetric matrix \mathbf{T} with a zero diagonal.

Let us begin with bound (5.1). Suppose that \mathcal{S} is a $(1 - \varepsilon)$ -optimal cut of \mathbf{M} . Then

$$\begin{aligned} \text{cut}_{\mathcal{S}}(\mathbf{D}) &= \text{cut}_{\mathcal{S}}(\mathbf{M}) + \text{cut}_{\mathcal{S}}(\mathbf{D} - \mathbf{M}) \\ &\geq (1 - \varepsilon) \text{maxcut}(\mathbf{M}) - \text{cut}_{\mathcal{S}}(\|\mathbf{D} - \mathbf{M}\|) \\ &\geq (1 - \varepsilon) \text{maxcut}(\mathbf{D} + (\mathbf{M} - \mathbf{D})) - \frac{1}{2} \|\mathbf{D} - \mathbf{M}\|_1 \\ &\geq (1 - \varepsilon) (\text{maxcut}(\mathbf{D}) - \text{maxcut}(\|\mathbf{M} - \mathbf{D}\|)) - \frac{1}{2} \|\mathbf{M} - \mathbf{D}\|_1 \\ &\geq (1 - \varepsilon) \text{maxcut}(\mathbf{D}) - (1 - \varepsilon/2) \|\mathbf{M} - \mathbf{D}\|_1. \end{aligned}$$

The proof for the bound (5.2) follows a similar outline. First, we implicitly define a relative error matrix \mathbf{E} with the relation $\mathbf{M} = \mathbf{D} \odot \mathbf{E}$. We assume that $m_{jk} = 0$

if and only if $d_{jk} = 0$ to ensure \mathbf{E} can be defined. If not, the resulting error bound would be trivial anyway. Let $r = \min\{e_{jk} : d_{jk} \neq 0\}$ and $R = \max\{e_{jk} : d_{jk} \neq 0\}$. For any zero entry of \mathbf{D} , take the corresponding entry of \mathbf{E} in the range $[r, R]$. In the sequel, we use ‘/’ for element-wise division.

Next, observe that

$$\begin{aligned} \text{cut}_{\mathcal{S}}(\mathbf{M}) &= \text{cut}_{\mathcal{S}}(\mathbf{D} \odot \mathbf{E}) = \sum_{j \in \mathcal{S}} \sum_{k \notin \mathcal{S}} d_{jk} e_{jk} \\ &\leq \max_{j \neq k} e_{jk} \sum_{j \in \mathcal{S}} \sum_{k \notin \mathcal{S}} d_{jk} \\ &\leq \|\mathbf{E}\|_{\infty} \text{cut}_{\mathcal{S}}(\mathbf{D}). \end{aligned}$$

Similarly,

$$\text{maxcut}(\mathbf{D}) = \text{maxcut}(\mathbf{M}/\mathbf{E}) \leq \|\mathbf{1}/\mathbf{E}\|_{\infty} \text{maxcut}(\mathbf{M}).$$

Then we compute

$$\begin{aligned} \text{cut}_{\mathcal{S}}(\mathbf{D}) &\geq \frac{\text{cut}_{\mathcal{S}}(\mathbf{M})}{\|\mathbf{E}\|_{\infty}} \\ &\geq \frac{1 - \varepsilon}{\|\mathbf{E}\|_{\infty}} \text{maxcut}(\mathbf{M}) \\ &\geq \frac{1 - \varepsilon}{\|\mathbf{E}\|_{\infty} \|\mathbf{1}/\mathbf{E}\|_{\infty}} \text{maxcut}(\mathbf{D}). \end{aligned}$$

This technique can be extended to other types of problems that are computationally easier for metric graphs [12]. Mettu and Plaxton have also considered fast algorithms for clustering “nearly metric” data, but their approach relies instead on weak versions of the triangle inequality [18]. Fast approximation algorithms for various other metric problems such as k -median, MAX-TSP, etc., are discussed in [13]; our method allows extending these approximation algorithms to non-metric data.

6. Experiments. We implemented metric nearness in C++ wherein we coded Algorithms 3.1 and 3.2. In this section we describe some experiments based on our implementation. All experiments were carried out in double precision on a P4/2.5GHz processor machine with 2GB RAM, running Linux.

6.1. Running Time. In this section we show some running time comparisons between CPLEX—a state of the art linear and quadratic optimization software—and our implementations of triangle fixing. Our results clearly indicate the superiority of triangle fixing over CPLEX. For these experiments, we used random dissimilarity matrices of dimensions up to 100×100 . The final values of the objective function achieved by CPLEX and our implementation agreed to five significant digits.

Figure 6.1 compares CPLEX quadratic programming to our implementation of ℓ_2 triangle fixing (see Algorithm 3.1). From the figure one can see that the triangle fixing procedure is up to thirty times faster than CPLEX’s fastest method for solving the metric nearness quadratic program. Our experiments suggest that the ℓ_2 triangle fixing procedure scales as $O(n^3)$.

For ℓ_1 metric nearness, we compared CPLEX’s fastest algorithm for metric nearness (determined by running all six choices available and selecting the fastest timing), and our implementation of the augmented triangle fixing procedure for solving the

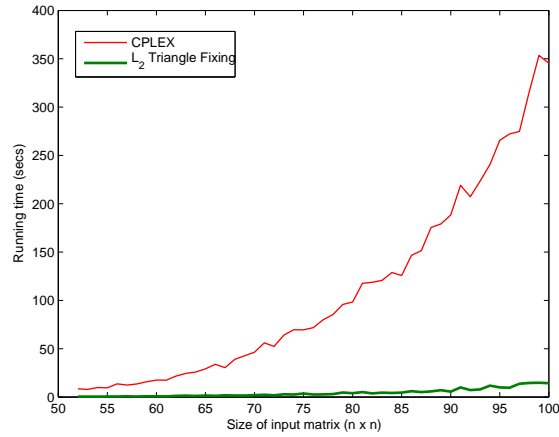


FIG. 6.1. Running time comparison between CPLEX and the ℓ_2 triangle fixing algorithm.

ℓ_1 metric nearness problem. Our implementation runs up to 15 times faster than CPLEX, as indicated by Figure 6.2. As suggested previously, we used $\epsilon = \max_{i,j} d_{ij}$ for our experiments.

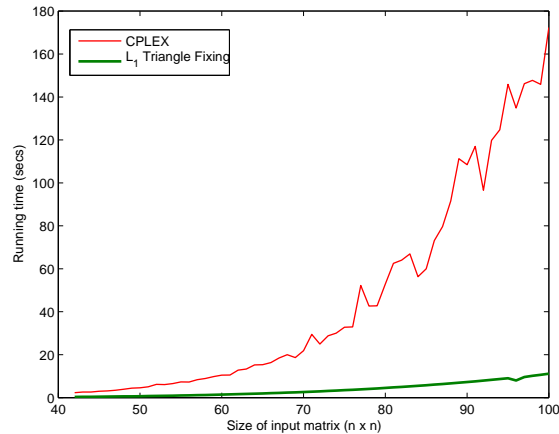


FIG. 6.2. Running time comparison between CPLEX and augmented triangle fixing (ℓ_1).

6.2. Decrease only metric nearness/APSP experiments. Although Floyd-Warshall and the primal-dual Algorithm 4.4 both have an asymptotic runtime of $O(n^3)$, the latter converges more quickly to the answer for certain classes of problems. Floyd-Warshall chooses an order of triangles to correct without any guidance, whereas the primal-dual algorithm prefers to correct triangles that include shorter edges. We can certainly imagine a problem instance where the violating triangles have longer edges, and in this case the preference for shorter edges does not help.

For randomly generated test cases, however, our primal-dual algorithm does converge more quickly than Floyd-Warshall. To illustrate this observation, we generated

random matrices of dimension 200×200 that had a zero diagonal and entries between 0.1 and 10. We then determined the correct answer before running both algorithms, halting the computation at each iteration to determine the distance between the current distance matrix and the final metric. Distance was computed as the l_1 vector distance. Figure 6.3 gives these results, which clearly show the primal-dual algorithm converging faster than Floyd-Warshall.

To determine the approximate time to convergence as a function of n , we generated $n \times n$ matrices for values of n from 25 to 225. Figure 6.4 plots the number of iterations required to converge for both Floyd-Warshall and the primal dual algorithm. The exponents in the big- O notation runtimes were approximated by fitting the curve to the best $a \cdot n^b$ approximation. While Floyd-Warshall takes the entire $O(n^3)$ time to converge, the primal-dual algorithm converges in about $O(n^{2.8})$ time. Even more striking is Figure 6.5, which plots the number of iterations the algorithms required to *nearly* converge (where nearly converging means being within $0.5 * n$ of the metric solution). Here Floyd-Warshall still required $O(n^3)$ time, but the primal-dual algorithm needed only about $O(n^{2.5})$ time.

Unfortunately, we cannot yet take advantage of this rapid convergence to improve the runtime of the APSP primal-dual algorithms. Ideally, we could terminate the algorithm when we had modified enough edges to cause the graph to be a metric. After the graph is a metric, there are no triangles in violation, so the additional steps of the algorithm do not modify the graph in any way. However, we are unaware of any computationally efficient way to solve the problem of *metricity*. That is, given a graph, return “true” if the graph is a metric, and “false” otherwise. One way to solve this is to run APSP on the graph, and then check to see if any edges were shortened. This observation yields an upper bound of $O(APSP)$ on the metricity problem. It follows that we cannot terminate the APSP primal-dual algorithms early, even if they have converged to the correct result, because testing for the termination condition has the same complexity as the problem itself.

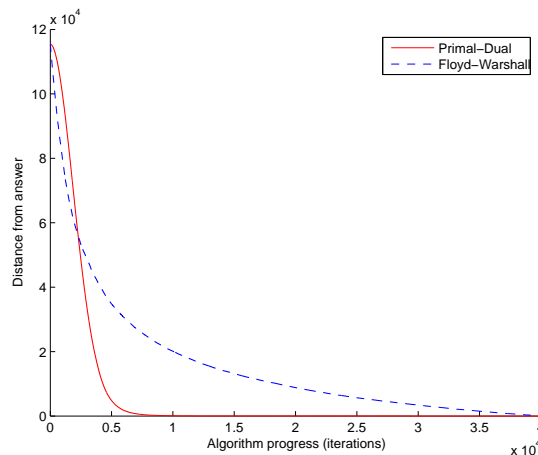


FIG. 6.3. *Convergence comparison of Floyd-Warshall and the primal-dual algorithm*

7. Discussion. Metric nearness is a rich problem. In this paper we formally introduced the problem and derived iterative algorithms for solving it for the vector ℓ_p norms. A special case of metric nearness was shown to be equivalent to the All

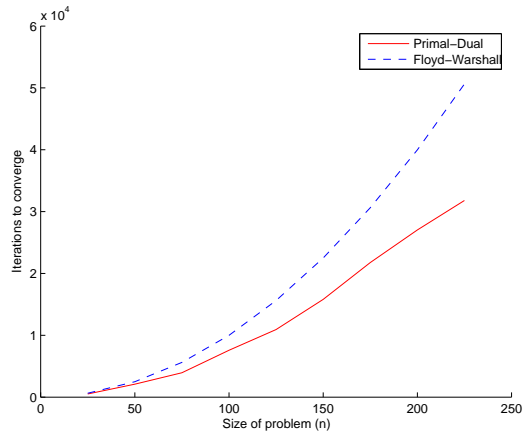


FIG. 6.4. *Iterations to converge for Floyd-Warshall and the primal-dual algorithm. Each iteration represents $O(n)$ computations.*

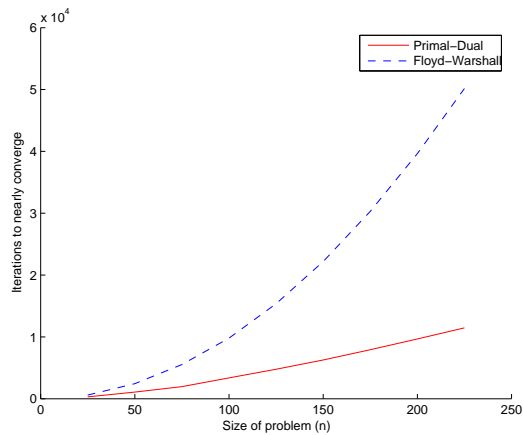


FIG. 6.5. *Iterations to nearly converge for Floyd-Warshall and the primal-dual algorithm. Each iteration represents $O(n)$ computations.*

Pairs Shortest Paths problem, which led to a new algorithm for APSP. We studied applications of metric nearness to MAX-CUT clustering. Experimental results illustrate the computational advantages of triangle fixing over generic optimization methods.

7.1. Variations. One may derive numerous variations of the metric nearness problem. The simplest of these involve the modification of the triangle inequality constraints in some interesting manners. These variations are all easily solved using our framework. Examples follow.

1. In Section 4 we discussed metric nearness with the restriction that permitted only decreasing changes to the entries of the input dissimilarity matrix. Similarly, one may also look at the problem where only increasing changes are permitted. Geometric or graph theoretic interpretations of this problem remain to be considered.
2. When performing metric nearness on non-symmetric input graphs, one can

choose either not to impose symmetry (as we did in the decrease-only section) or to impose symmetry. The latter case introduces additional constraints, but can be solved in our framework with only slight modifications.

3. Some applications may desire *rank* or order constraints to be enforced. That is, if the input satisfies $d_{ij} < d_{pq}$, then we also require $m_{ij} < m_{pq}$. Such a requirement can be useful in scenarios where the relative ordering of the dissimilarity values has a significance for the underlying application.
4. Box constraints, i.e., constraints of the type $l_{ij} \leq m_{ij} \leq u_{ij}$. Such constraints can be useful when a true metric, as opposed to a pseudo-metric, is desired (achieved by setting $l_{ij} > 0$). Upper bounds on the distance values may be utilized to prevent certain undesirable solutions.
5. Enforcement of λ -triangle inequalities that take the form $\lambda_{ij}m_{ij} \leq \lambda_{ik}m_{ik} + \lambda_{kj}m_{kj}$. Since the structure of the inequalities remains unaltered this problem can also be solved by triangle fixing.

Other variations involve generalization of the basic problem. The most important of such generalizations is one that introduces a weighting scheme to the problem. Here we propose to obtain a distance matrix \mathbf{M} such that

$$\mathbf{M} \in \operatorname{argmin}_{\mathbf{X} \in \mathcal{M}_N} \|\mathbf{W} \odot (\mathbf{X} - \mathbf{D})\|,$$

where $\|\cdot\|$ is a norm, \odot denotes the element-wise matrix product, and \mathbf{W} is a weighting matrix (a symmetric nonnegative matrix). The weight matrix reflects our confidence in the entries of \mathbf{D} . When each d_{ij} represents a measurement with variance σ_{ij}^2 , we might set $w_{ij} = 1/\sigma_{ij}^2$. If an entry of \mathbf{D} is missing, one can set the corresponding weight to zero (however, the resulting problem loses strict convexity, whereby one should set this weight to a small value instead of zero).

7.2. Future work. Metric nearness is a relatively new problem. Many aspects could form a basis for future work and further consideration. The most immediate concerns that interest us are:

1. Extensions to triangle fixing; for example, one may speed up the procedure by fixing all the independent triangle inequalities in parallel. One could also attempt to fix a few dependent triangle inequalities at the same time, and such an approach will result in a dual block coordinate ascent scheme [27].
2. Studying the convergence of the triangle fixing algorithms at least for the ℓ_2 case. If possible, it would be interesting to furnish a proof of convergence for our algorithms that is independent of the convergence of the more general Bregman's method.
3. Exploring applications of metric nearness, for e.g., applications to constrained clustering or various other applications that make use of proximity data and would profit from having metric data.

7.3. Open Problems. Two interesting open problems spring out of metric nearness. First is the metricity problem that seeks to verify if the input dissimilarity matrix is actually a distance matrix. Some related work that probabilistically tests metric properties of an input dissimilarity matrix can be found in [21]. Whether the metric verification problem has the same complexity as the metric nearness problem remains to be ascertained. Second is the search for faster algorithms for the general metric nearness problem. Along with faster algorithms, the possibility of guaranteed polynomial time (non-iterative procedures) algorithms still remains.

7.4. Related work. Metric nearness is a relatively new problem that was introduced by the authors, where preliminary work includes [6, 7].

The most relevant research appears in recent papers of Roth et al. [24, 25]. They observe that machine learning applications often require metric data, and they propose a technique for converting general dissimilarity data into metric data. Their method, constant-shift embedding, increases all the dissimilarities by an equal amount to produce a set of Euclidean distances (i.e., a set of numbers that can be realized as the pairwise distances among an ensemble of points in a Euclidean space). The size of the translation depends on the data, so the relative and absolute changes to the dissimilarity values can be large. Our approach is completely different. We seek a consistent set of distances that *deviates as little as possible* from the original measurements. In our approach, the resulting set of distances can arise from an arbitrary metric space; we do not restrict our attention to obtaining Euclidean distances. In consequence, we expect metric nearness to provide superior denoising. Moreover, our techniques can also learn distances that are missing entirely.

The technique of shifting the spectrum leads to an omission of the information carried by the negative eigenvalues of the input matrix. Laub and Müller [15] explore how the negative part of the spectrum could code for relevant features of the underlying data. Their method once again is based around computing an embedding, which is different from metric nearness, since the latter aims to only obtain a metric and constructs no embedding.

There is at least one other method for inferring a metric that proposes a technique for learning a Mahalanobis distance for data in \mathbb{R}^s [28]. That is, a metric $\text{dist}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{G} (\mathbf{x} - \mathbf{y})}$, where \mathbf{G} is an $s \times s$ positive semi-definite matrix. The user specifies that various pairs of points are similar or dissimilar. Then the matrix \mathbf{G} is computed by minimizing the total *squared* distances between similar points while forcing the total distances between dissimilar points to exceed one. The article provides explicit algorithms for the cases where \mathbf{G} is diagonal and where \mathbf{G} is an arbitrary positive semi-definite matrix. In comparison, the metric nearness problem is not restricted to Mahalanobis distances; it can learn a general discrete metric. It also allows us to use specific distance measurements and to indicate our confidence in those measurements (by means of a weight matrix), rather than forcing a binary choice of “similar” or “dissimilar.”

The Metric Nearness Problem may appear similar to metric Multi-Dimensional Scaling [14], but we emphasize that the two problems are *distinct*. The latter problem endeavors to find an ensemble of points in a *prescribed* metric space—usually a Euclidean space—such that the distances between these points are close to the set of input distances. In contrast, metric nearness does not seek an embedding—it does not impose any hypotheses on the underlying space other than requiring it to be a metric space. For more details on Euclidean Distance Matrices see [9, 10, 26].

Related to metrics are ultrametrics; a distance matrix \mathbf{M} is said to be an ultrametric if $m_{ij} \leq \max\{m_{ik}, m_{kj}\}$ for every distinct triple of indices (i, k, j) . It is known that finding the nearest (in ℓ_1 and ℓ_2 norms) ultrametric to a given input matrix is NP-Complete [4]. However, the ℓ_∞ -nearest ultrametric can be computed in $O(n^2)$ time [8]. Hubert *et al.* [11] consider the problem of representing a dissimilarity matrix by a sum of matrices having a particular form, including a form that restricts the matrices to be ultrametrics.

REFERENCES

- [1] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY, *Proof verification and the hardness of approximation problems*, J. Assoc. Comput. Mach., 45 (1998), pp. 501–555.
- [2] Y. CENSOR AND S. A. ZENIOS, *Parallel Optimization: Theory, Algorithms, and Applications*, Oxford University Press, 1997.
- [3] T. H. CORMEN, C. E. LEISERSON, R. RIVEST, AND C. STEIN, *Introduction to Algorithms*, The MIT Press, 2 ed., 2001.
- [4] W. H. E. DAY, *Computational complexity of inferring phylogenies from dissimilarity matrices*, Bulletin of Mathematical Biology, 49 (1987), pp. 461–467.
- [5] W. F. DE LA VEGA AND C. KENYON, *A randomized approximation scheme for Metric MAX-CUT*, J. Comput. Sys. and Sci., 63 (2001), pp. 531–541.
- [6] I. S. DHILLON, S. SRA, AND J. A. TROPP, *The Metric Nearness Problems with Applications*, Tech. Rep. TR-03-23, Computer Sciences, University of Texas at Austin, 2003.
- [7] ———, *Triangle fixing algorithms for the metric nearness problem*, in Advances in Neural Information Processing Systems 17, L. K. Saul, Y. Weiss, and L. Bottou, eds., Cambridge, MA, 2005, MIT Press.
- [8] M. FARACH, S. KANNAN, AND T. WARNOW, *A robust model for finding optimal evolutionary trees*, Algorithmica, 13 (1995).
- [9] J. C. GOWER, *Properties of Euclidean and non-Euclidean distance matrices*, Linear Algebra and its Applications, 67 (1985), pp. 81–97.
- [10] N. J. HIGHAM, *Matrix nearness problems and applications*, in Applications of Matrix Theory, M. J. C. Gower and S. Barnett, eds., Oxford University Press, 1989, pp. 1–27.
- [11] L. J. HUBERT, P. ARABIE, AND J. MEULMAN, *The representation of symmetric proximity data: Dimensions and classifications*, The Computer Journal, 41 (1998).
- [12] P. INDYK, *A Sublinear-time Approximation Scheme for Clustering in Metric Spaces*, in 40th Symposium on Foundations of Computer Science, 1999.
- [13] ———, *Sublinear time algorithms for metric space problems*, in 31st Symposium on Theory of Computing, 1999, pp. 428–434.
- [14] J. B. KRUSKAL AND M. WISH, *Multidimensional Scaling*, Sage Publications, 1978. Series: Quantitative Applications in the Social Sciences.
- [15] J. LAUB AND K.-R. MÜLLER, *Feature discovery in Non-metric pairwise data*, Journal of Machine Learning Research, 5 (2004), pp. 801–818.
- [16] D. G. LUENBERGER, *Linear and Nonlinear Programming*, Addison-Wesley Publishing Company, second ed., 1984.
- [17] O. L. MANGASARIAN, *Normal solutions of linear programs*, Mathematical Programming Study, 22 (1984), pp. 206–216.
- [18] R. R. METTU AND C. G. PLAXTON, *The online median problem*, SIAM J. Comput., 32 (2003), pp. 816–832.
- [19] C. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Dover, 2000.
- [20] C. PAPADIMITRIOU AND M. YANNAKAKIS, *Optimization, approximation and complexity classes*, J. Comput. Sys. and Sci., 43 (1991), pp. 425–440.
- [21] M. PARNAS AND D. RON, *Testing metric properties*, in STOC, 2001, pp. 276–285.
- [22] C. G. PLAXTON. Personal Communication, 2003–2004.
- [23] R. T. ROCKAFELLAR, *Convex Analysis*, Princeton Univ. Press, 1970.
- [24] V. ROTH, J. LAUB, J. M. BUHMANN, AND K.-R. MÜLLER, *Going metric: Denoising pairwise data*, in Advances in Neural Information Processing Systems (NIPS) 15, S. Becker, S. Thrun, and K. Obermayer, eds., 2003.
- [25] V. ROTH, J. LAUB, M. KAWANABE, AND J. M. BUHMANN, *Optimal cluster preserving embedding of non-metric proximity data*, IEEE Trans. Pattern Analysis and Machine Intelligence, 25 (2003).
- [26] I. J. SCHOENBERG, *Remarks to Maurice Fréchet’s article “Sur la définition axiomatique d’une classe d’espace distanciés vectoriellement applicable sur l’espace de Hilbert”*, Annals of Mathematics, 36 (1935), pp. 724–732.
- [27] P. TSENG, *Dual coordinate ascent methods for non-strictly convex minimization*, Mathematical Programming, 59 (1993), pp. 231–247.
- [28] E. P. KING, A. Y. NG, M. I. JORDAN, AND S. RUSSELL, *Distance metric learning, with application to clustering with side constraints*, in Advances in Neural Information Processing Systems (NIPS) 15, S. Becker, S. Thrun, and K. Obermayer, eds., 2003.

Appendix A. More on Metric Nearness. This appendix includes additional

informative material pertinent to metric nearness.

A.1. Metric Nearness and APSP. Lemma A.1 formalizes the equivalence between a “decrease only” version of metric nearness and APSP. This equivalence was originally suggested by [22].

LEMMA A.1 (Decrease only metric nearness is APSP). *Let $M^A \in \mathcal{M}_N$ be the APSP solution for D . Then, M^A is also the nearest “decrease only” metric solution. Furthermore, for any $M \in \mathcal{M}_N$, if $M \leq D$ then $M \leq M^A$.*

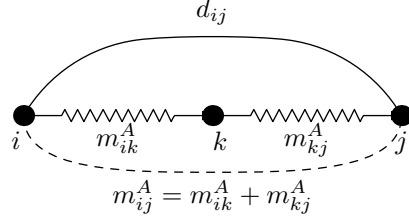


FIG. A.1. Shortest path between i and j via k

Proof. We prove the last statement of the lemma, noting that it immediately implies the rest.

Assume the edge weights m_{ij}^A of M^A are sorted in increasing order, and that the least-weighted edge for which M exceeds M^A is m_{ij} , i.e., $m_{ij} > m_{ij}^A$. Since M^A is an APSP solution for D , each edge weight m_{ij}^A either equals d_{ij} or is the sum of weights of edges involved in a shortest path of length less than d_{ij} , as shown in Figure A.1.

In the figure, k is some intermediate vertex on a shortest path from i to j . The zig-zag lines denote paths from $i \rightarrow k$ and $k \rightarrow j$. Since M is a metric solution, $m_{ij} \leq m_{ik} + m_{kj}$. Now $m_{ik}^A \leq m_{ij}^A$ and $m_{kj}^A \leq m_{ij}^A$, since $m_{ij}^A = m_{ik}^A + m_{kj}^A$. By our assumption $m_{ij} > m_{ij}^A$ is the first place where a component of M exceeds a component of M^A (taken in sorted order), hence $m_{ik} \leq m_{ik}^A$ and $m_{kj} \leq m_{kj}^A$, which in turn implies that $m_{ij} \leq m_{ij}^A$. We have arrived at a contradiction to our initial assumption and that completes the proof of our claim. \square

A.1.1. Equivalence of APSP to DOMN. Lemma A.1 shows that the optimal assignment of the x_{ij} variables in linear program (4.3) is the same as the *distances* given by the APSP solution. In this section, we will investigate an equivalence between the optimal assignment of the π_{ij} and γ_{ikj} variables in linear program (4.4) and the *paths* given by the APSP solution.

DOMN from APSP. Given an APSP solution, we construct an optimal solution to the DOMN problem (4.3) using the following procedure:

- If the edge (i, j) is used by n shortest paths, then set $\pi_{ij} = n$.
- If the edge (i, j) is used by n shortest paths *en route* to node k , then set $\gamma_{ijk} = n$.

Feasibility of the above assignment. Clearly the non-negativity constraints $\pi_{ij} \geq 0$ and $\gamma_{ikj} \geq 0$ are satisfied. We must show that the constraint

$$\pi_{ij} + \sum_{k \neq i, j} (\gamma_{ikj} - \gamma_{ijk} - \gamma_{kij}) = 1$$

is satisfied for all edges (i, j) .

For vertex pairs i and j in which the shortest path from i to j is the edge (i, j) , this assignment will be consistent with the constraint because all shortest paths involving the edge (i, j) fall into one of three categories: the path from i to j , paths to j that end with the edge (i, j) , and paths to another vertex k that pass through the edge (i, j) . The latter two categories contribute both a $+1$ and a -1 to the constraint, while the first category contributes a $+1$, resulting in a net sum of 1.

For vertex pairs i and j in which the shortest path from i to j begins with the edge (i, k) , this assignment is also consistent with the constraint. There are two types of shortest paths ending at node j and using edge (i, k) : the path that starts at i , and paths that start at a node l and pass through (i, k) before finishing at j . The latter type of path contributes both a $+1$ and a -1 to the constraint, while the first type contributes a $+1$ for a total of 1.

Optimality of the assignment. Under the proposed variable assignment procedure, the objective function for (4.4) is the sum of all path distances. Because the paths were taken from an APSP solution, this objective is minimized.

APSP from DOMN. Given an optimal solution to (4.4), we construct an APSP solution using the following procedure:

- If π_{ij} is positive, then the edge (i, j) is a shortest path from i to j .
- If γ_{ikj} is positive, then there is a shortest path from i to j that passes through k ; we may recursively find the shortest paths from i to k and from k to j .