Maximum Variance Correction with Application to A^* Search

Wenlin Chen WENLINCHEN@WUSTL.EDU
Kilian Q. Weinberger KILIAN@WUSTL.EDU
Yixin Chen CHEN@CSE.WUSTL.EDU
Washington University, One Brookings Dr., St. Louis, MO 63130 USA

Abstract

In this paper we introduce Maximum Variance Correction (MVC), which finds large-scale feasible solutions to Maximum Variance Unfolding (MVU) by post-processing embeddings from *any* manifold learning algorithm. It increases the scale of MVU embeddings by several orders of magnitude and is naturally parallel. This unprecedented scalability opens up new avenues of applications for manifold learning, in particular the use of MVU embeddings as effective heuristics to speed-up A^* search. We demonstrate unmatched reductions in search time across several non-trivial A^* benchmark search problems and bridge the gap between the manifold learning literature and one of its most promising high impact applications.

1. Introduction

Manifold learning has become a strong sub-field of machine learning with many mature algorithms (Saul et al., 2006; Lee & Verleysen, 2007), often accompanied by large scale extensions (Platt, 2004; Silva & Tenenbaum, 2002; Weinberger et al., 2007) and thorough theoretical analysis (Donoho & Grimes, 2002; Paprotny et al., 2012). Until recently, this success story was not matched by comparably strong applications (Blitzer et al., 2005). Rayner et al. (2011) propose to use the Euclidean embedding of a search space graph as a heuristic for A^* search (Russell & Norvig, 2003). The graph-distance between two states is approximated by the Euclidean distance between their respective embedded points.

Exact A^* search with informed heuristics is an application of great importance in many areas of real life. For example, GPS navigation systems need to find the shortest path between two locations *efficiently* and *repeatedly* (*e.g.* each time a new traffic update has been received, or when the driver makes a wrong turn). As the processor capabilities of these devices and the patience of the users are both limited, the quality of the search heuristic is of great importance. This importance only increases as increasingly *low powered* embedded devices (*e.g.* smart-phones) are equipped with similar capabilities. Other applications include massive online multiplayer games, where agents need to identify the shortest path along a map which can change dynamically through actions by other users.

For an embedding to be a A^* heuristic, it must satisfy two properties: 1. *admissible* (distances are never *overestimated*), 2. *consistent* (a triangular inequality like property is preserved). To be maximally effective, a heuristic should have a minimal gap between its estimate and the true distance—*i.e.* all pair-wise distances should be maximized under the admissibility and consistency constraints. In the applications highlighted by Rayner et al. (2011), a heuristic must require small space to be broadcasted to the end-users. The authors show that the constraints of Maximum Variance Unfolding (MVU) (Weinberger & Saul, 2006)[1] guarantee admissibility and consistency, while the objective maximizes distances and reduces space requirement of heuristics from $O(n^2)$ to $O(dn)$. In other words, the MVU manifold learning algorithm is a perfect fit to learn Euclidean heuristics for A^* search.

Unfortunately, it is fair to say that due to its semi-definite programming (SDP) formulation (Boyd & Vandenberghe, 2004), MVU is amongst the least scalable manifold learning algorithms and cannot embed state spaces beyond 4000 states—severally limiting the usefulness of the proposed heuristic in practice. Although there have been efforts to increase the scala-

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

[1] Throughout this paper we refer to MVU as the formulation with *inequality* constraints.

bility of MVU (Weinberger et al., 2005; 2007), these lead to approximate solutions which no longer guarantee admissibility or consistency of heuristics.

In this paper we propose a novel algorithm, Maximum Variance Correction (MVC), which improves the scalability of MVU by several orders of magnitude. In a nutshell, MVC post-processes embeddings from any manifold learning algorithm, to strictly satisfy the MVU constraints by rearranging embedded points within local patches. Hereby MVC combines the strict finite-size guarantees of MVU with the large-scale capabilities of alternative algorithms. Further, it bridges the gap between the rich literature on manifold learning and what we consider its most promising and high-impact application to date—the use of Euclidean state-space embeddings as A^* heuristics.

Our contributions are summarized as follows: 1) We introduce MVC, a fully *parallelizable* algorithm that scales up and speeds up MVU by several orders of magnitudes. 2) We provide a formal proof that any solution of our relaxed problem formulation still satisfies all MVU constraints. 3) We demonstrate on several A^* search benchmark problems that the resulting heuristics lead to impressive reductions in search-time—even beating the competitive differential heuristic (Ng & Zhang, 2002) by a large factor on all data sets.

2. Background and related work

There have been several recent publications that increase the scalability of manifold learning algorithms. Vasiloglou et al. (2008); Weinberger et al. (2007); Weinberger & Saul (2006) directly scale up MVU by relaxing its constraints and restricting the solution to the space spanned by landmark points or the eigenvectors of the graph laplacian matrix. Silva & Tenenbaum (2002); Talwalkar et al. (2008) scale up Isomap (Tenenbaum et al., 2000b) with Nyström approximations. Our work is complementary as we refine these embeddings to meet the MVU constraints while maximizing the variance of the embedding.

Shaw & Jebara (2009) introduce structure preserving embedding, which learns embeddings that strictly preserve graph properties (such as nearest neighbors). Zhang et al. (2009) also focus on local patches of manifolds, however preserves discriminative ability rather than the finite-size guarantees of MVU.

From a technical stand-point, our paper is probably most similar to Biswas & Ye (2004) which uses a semi-definite program for sensor network embedding. Due to the nature of their application, they deal with different constraints and objectives.

2.1. Graph Embeddings

We briefly review MVU and Isomap as algorithms for proximity graph embedding. For a more detailed survey we recommend (Saul et al., 2006). Let $G = (V, E)$ denote the graph with undirected edges E and nodes V, with $|V| = n$. Edges $(i, j) \in E$ are weighted by some $d_{ij} \geq 0$. Let δ_{ij} denote the shortest path distance from node i to j. Manifold learning algorithms embed the nodes in V into a d-dimensional Euclidean space, $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{R}^d$, such that $\|\mathbf{x}_i - \mathbf{x}_j\|_2 \approx \delta_{ij}$.

Maximum Variance Unfolding formulates this task as an optimization problem that maximizes the variance of the embedding, while enforcing strict constraints on the local edge distances:

$$\begin{aligned} \underset{\mathbf{x}_1,\ldots,\mathbf{x}_n \in \mathcal{R}^d}{\text{maximize}} \quad & \sum_{i=1}^{n} \mathbf{x}_i^2 \\ \text{subject to} \quad & ||\mathbf{x}_i - \mathbf{x}_j||_2 \leq d_{ij} \quad \forall (i,j) \in E \\ & \sum_{i=1}^{n} \mathbf{x}_i = 0 \end{aligned} \tag{1}$$

The last constraint centers the embedding at the origin, to remove translation as a degree of freedom in the optimization. Because the data is centered, the objective is identical to maximizing the variance, as $\sum_i \mathbf{x}_i^2 = 0.5 \sum_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|^2$. Although (1) is non-convex, Weinberger & Saul (2006) show that with a rank relaxation, $\mathbf{x} \in \mathcal{R}^n$, this problem can be rephrased as a convex semi-definite program by optimizing over the inner-product matrix $\mathbf{K}$, with $k_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$:

$$\begin{aligned} \underset{\mathbf{K}}{\text{maximize}} \quad & \text{trace}(\mathbf{K}) \\ \text{subject to} \quad & k_{ii} - 2k_{ij} + k_{jj} \leq d_{ij}^2 \quad \forall (i,j) \in E \\ & \sum_{i,j} k_{ij} = 0 \\ & \mathbf{K} \succeq 0. \end{aligned} \tag{2}$$

The final constraint $\mathbf{K} \succeq 0$ ensures positive semi-definiteness and guarantees that $\mathbf{K}$ can be decomposed into vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with a straightforward eigenvector decomposition. To ensure strictly $r-$dimensional output, the final embedding is projected into $\mathcal{R}^d$ with principal component analysis (PCA). (This is identical to composing the vectors $\mathbf{x}_i$ out of the r leading eigenvectors of $\mathbf{K}$.) The time-complexity of MVU is $O(n^3 + c^3)$ (where c is the number of constraints in the optimization problem), which makes it prohibitive for larger data sets.

Graph Laplacian MVU (gl-MVU), Weinberger & Saul (2006); Wu et al. (2009), is an extension of MVU that reduces the size of $\mathbf{K}$ by matrix factorization,

$\mathbf{K} = \mathbf{Q}^\top \mathbf{L}\mathbf{Q}$. Here, $\mathbf{Q}$ are the bottom eigenvectors of the Graph Laplacian, also referred to as **Laplacian Eigenmaps** (Belkin & Niyogi, 2002). All local distance constraints are removed and instead added as a penalty term into the objective. The resulting algorithm scales to larger data sets but makes no exact guarantees about the distance preservations.

Isomap, Tenenbaum et al. (2000a), preserves the global structure of the graph by directly preserving the graph distances between *all* pair-wise nodes:

$$\min_{\mathbf{x}_1,\ldots,\mathbf{x}_n \in \mathcal{R}^d} \sum_{i,j} ((\mathbf{x}_i - \mathbf{x}_j)^2 - \delta_{ij}^2)^2. \tag{3}$$

Tenenbaum et al. (2000a) show that (3) can be approximated as an eigenvector decomposition by applying multi-dimensional scaling (MDS) (Kruskal, 1964) on the shortest path distances $\delta(i,j)$. The landmark extension (Silva & Tenenbaum, 2002) leads to significant speed-ups with Nyström approximations of the graph-distance matrix. For simplicity, we refer to it also as "Isomap" throughout this paper.

2.2. Euclidean Heuristic

The A^* search algorithm finds the shortest path between two nodes in a graph. In the worst case, the complexity of the algorithm is exponential in the length of the shortest path, but the search time can be drastically reduced with a good heuristic, which estimates the graph distance between two nodes. Rayner et al. (2011) suggest to use the distance $h(i,j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ of the MVU graph embedding as such a heuristic, which they refer to as *Euclidean Heuristic*. A^* with this heuristic provably converges to the exact solution, as the heuristic is admissible and consistent. More precisely, for all nodes i,j,k the following holds:

$$\text{Admissibility:} \quad \|\mathbf{x}_i - \mathbf{x}_k\|_2 \leq \delta_{ik} \tag{4}$$

$$\text{Consistency:} \quad \|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq \delta_{ik} + \|\mathbf{x}_k - \mathbf{x}_j\|_2 \tag{5}$$

The proof is straight-forward. As the shortest-path between nodes i and j in the embedding consists of edges which are all underestimated, it must be underestimated itself and so is $\|\mathbf{x}_i - \mathbf{x}_j\|_2$ (which implies admissibility). Consistency follows from the triangular inequality in combination with(4).

The closer the gap in the admissibility inequality (4), the better is the search heuristic. The perfect heuristic would be the actual shortest path, $h(i,j) = \delta_{ij}$ (with which A^* could find the exact solution in linear time with respect to the length of the shortest path). The MVU objective maximizes all pairwise distances, and therefore minimizes exactly the gap in (4). Consequently, MVU is the perfect optimization problem to find a Euclidean Heuristic—however in its original formulation it can only scale to $n \approx 4000$. In the following we will scale up MVU to much larger data sets.

3. Maximum Variance Correction

In this section, we introduce our MVC algorithm. Intuitively, MVC combines the scalability of gl-MVU and Isomap with the strong guarantees of MVU: It uses the former to obtain an initial embedding of the data and then post-processes it into a local optimum of the MVU optimization. The post-processing only involves re-optimizations of local patches, which is fast and can be decomposed into independent sub-problems.

Initialization. We obtain an initial embedding $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n$ of the graph with any (large-scale) manifold learning algorithm (*e.g.* Isomap, gl-MVU or Eigenmaps). The resulting embedding is typically not a feasible solution to the exact MVU problem, because it violates many distance inequality constraints in (1). To make it feasible, we first center it and then rescale the entire embedding such that all inequalities hold with at least one equality,

$$\mathbf{x}_i = \alpha(\hat{\mathbf{x}}_i - \frac{1}{n}\sum_{i=1}^{n} \hat{\mathbf{x}}_i), \text{ with } \alpha = \min_{(i,j)\in E} \frac{d_{ij}}{\|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|}. \tag{6}$$

After the translation and rescaling in (6) we obtain a solution in the feasible set of MVU embeddings, and therefore also an admissible and consistent Euclidean Heuristic. In practice, this heuristic is of very limited use because it has a very large admissibility gap (4). In the following sections we explain how to transform the embedding to maximize the MVU objective, while remaining inside the MVU feasible region.

3.1. Local patching

The (convex) MVU optimization is an SDP, which in their general formulation scale cubic in the input size n. To scale-up the optimization we therefore utilize a *specific* property of the MVU constraints: All constraints are strictly *local* as they only involve directly connected nodes. This allows us to divide up the graph embedding into local patches and re-optimize the MVU optimization on each patch individually. This approach has two clear advantages: the local patches can be made small enough to be re-optimized very quickly and the individual patch optimizations are inherently parallelizable—leading to even further speed-ups on modern multi-core computers. A challenge is to ensure that the local optimizations do not interfere with each other and remain globally feasible.

Graph partitioning. There are several ways to di-

vide the graph $G = (V, E)$ into r mutually exclusive connected components. We use repeated breadth first search (BFS) (Russell & Norvig, 2003) because of its simplicity, fast speed and guarantee that all partitions are connected components. Specifically, we pick a node i uniformly at random and apply BFS to identify the m closest nodes according to graph distance, that are not already assigned to patches. These nodes form a new patch $G_p = (V_p, E_p)$. The partitioning is continued until all nodes in V are assigned to exactly one partition, resulting in approximately $r = \lceil n/m \rceil$ patches.[2] The final partitioning satisfies $V = V_1 \cup \cdots \cup V_r$ and $V_p \cap V_q = \{\}$ for all p, q.

We distinguish between two types of nodes within a partition V_p (illustrated in figure 1). A node $i \in V_p$ is an *inner point* (blue circle) of V_p if all edges $(i, j) \in E$ connect it to other nodes $j \in V_p$; i is an *anchor point* (red circle) of V_p if there exists an edge $(i, j) \in E$ to some $j \notin V_p$. Let V_p^x denote the set of all inner nodes and V_p^a the set of all anchor points in V_p. By definition, these sets are mutually exclusive and together contain all points, *i.e.* $V_p^x \cap V_p^a = \{\}$ and $V_p = V_p^x \cup V_p^a$.

Similarly, all edges in E can be divided into three mutual exclusive subsets (see figure 1): edges between inner points (E^{xx}, blue); between anchor points (E^{aa}, red); between anchor and inner points (E^{ax}, purple).

Optimization. We first re-state the non-convex MVU optimization (1), slightly re-formulated to incorporate the graph partitioning. As each input is either an anchor point or an inner point of its respective patch, we can denote the set of all inner points as $V^x = \bigcup_p V_p^x$ and the set of all anchor points as $V^a = \bigcup_p V_p^a$. If we re-order the summations and constraints by these sets, we can re-phrase the non-convex MVU optimization (1) as

$$\begin{aligned} \underset{\mathbf{x}_i, \mathbf{a}_k}{\text{maximize}} \quad & \sum_{i \in V^x} \mathbf{x}_i^2 + \sum_{k \in V^a} \mathbf{a}_k^2 \\ \text{subject to} \quad & ||\mathbf{x}_i - \mathbf{x}_j||_2 \leq d_{ij} \quad \forall (i,j) \in E^{xx} \\ & ||\mathbf{x}_i - \mathbf{a}_k||_2 \leq d_{ik} \quad \forall (i,k) \in E^{ax} \\ & ||\mathbf{a}_i - \mathbf{a}_j||_2 \leq d_{ij} \quad \forall (i,j) \in E^{aa} \\ & \sum_{\mathbf{a}_i \in V^a} \mathbf{a}_i + \sum_{\mathbf{x}_i \in V^x} \mathbf{x}_i = \mathbf{0}. \end{aligned} \tag{7}$$

For clarity, we denote all anchor points as $\mathbf{a}_i$'s and inner points as $\mathbf{x}_j$'s and with a slight abuse of notation write $\mathbf{a}_i \in V^a$.

Optimization by patches. The optimization (7) is identical to the non-convex MVU formulation (1) and just as hard to solve. To reduce the computational complexity we make two changes: we remove the centering constraint and fix the anchor points in place. The removal of the centering constraint is a harmless relaxation because the fixed anchor points already remove translation as a degree of freedom and fixate the solution very close to zero-mean. (The objective changes slightly, but in practice this has minimal impact on the solution.) The fixing of the anchor points allows us to break down the optimization into r independent sub-problems. This can be seen from the fact that by definition all constraints in E^{xx} never cross patch boundaries, and constraints in E^{ax} only connect points within a patch with fixed points. Constraints over edges in E^{aa} can be dropped entirely, as edges between anchor points are necessarily fixed also. We obtain r independent optimization problems of the following type:

$$\begin{aligned} \underset{\mathbf{x}_i \in V_p^x}{\text{maximize}} \quad & \sum_{i \in V_p} \mathbf{x}_i^2 \\ \text{subject to} \quad & ||\mathbf{x}_i - \mathbf{x}_j||_2 \leq d_{ij} \quad \forall (i,j) \in E_p^{xx} \\ & ||\mathbf{x}_i - \mathbf{a}_k||_2 \leq d_{ik} \quad \forall (i,k) \in E_p^{ax}. \end{aligned} \tag{8}$$

The solutions of the r sub-problems (8) can be combined and centered, to form a feasible solution to (7).

Convex patch re-optimization. Similar to the non-convex MVU formulation (1), optimization (8) is also non-convex and non-trivial to solve. However, with a change of variables and a slight relaxation we can transform it into a semi-definite program. Let $n_p = |V_p|$. Given a patch G_p, we define a matrix $X = [\mathbf{x}_1, \ldots, \mathbf{x}_{n_p}] \in \mathcal{R}^{d \times n_p}$, where each column corresponds to one embedded input of V_p^x—the variables we want to optimize. Further, let us define the matrix $K \in \mathcal{R}^{(d+n_p)\times(d+n_p)}$ as:

$$\mathbf{K} = \begin{pmatrix} \mathbf{I} & \mathbf{X} \\ \mathbf{X}^\top & \mathbf{H} \end{pmatrix} \quad \text{where } \mathbf{H} = \mathbf{X}^\top \mathbf{X}. \tag{9}$$

The vector $\mathbf{e}_{i,j} \in \mathcal{R}^{n_p}$ is all-zero except the i^{th} element is 1 and the j^{th} element is -1. The vector $\mathbf{e}_i$ is all-zero except the i^{th} element is -1. With this notation, we

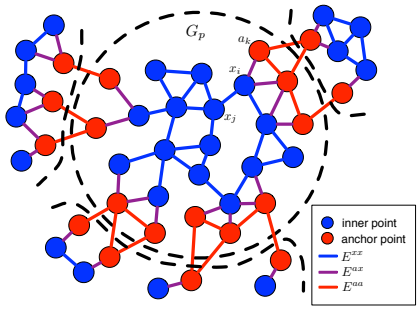

Figure 1. Drawing of a patch with inner and anchor points.

[2] The exact number of patches and number of nodes per patch vary slightly, depending on the connectivity of the graph, but all $|V_p| \leq m$.

obtain

$$\begin{aligned}(\mathbf{0};\mathbf{e}_{ij})^\top \mathbf{K}(\mathbf{0};\mathbf{e}_{ij}) &= \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \\ (\mathbf{a}_k;\mathbf{e}_i)^\top \mathbf{K}(\mathbf{a}_k;\mathbf{e}_i) &= \|\mathbf{x}_i - \mathbf{a}_k\|_2^2,\end{aligned} \tag{10}$$

where $(\mathbf{0};\mathbf{e}_{ij}) \in \mathcal{R}^{(d+n_p)}$ denotes the vector $\mathbf{e}_{ij}$ padded with zeros on top and $(\mathbf{a}_k;\mathbf{e}_i) \in \mathcal{R}^{(d+n_p)}$ the concatenation of $\mathbf{a}_k$ and $\mathbf{e}_i$.

Through (10), all constraints in (8) can be reformulated as a linear form of $\mathbf{K}$ (after squaring). The objective reduces to $\text{trace}(\mathbf{H}) = \sum_{i=1}^{n_p} \mathbf{x}_i^2$. The resulting optimization problem becomes:

$$\begin{aligned}\max_{\mathbf{X},\mathbf{H}} \quad & \text{trace}(\mathbf{H}) \\ \text{s.t.} \quad & (\mathbf{0};\mathbf{e}_{ij})^\top \mathbf{K}(\mathbf{0};\mathbf{e}_{ij}) \le d_{ij}^2 \qquad \forall (i,j) \in E_p^{xx} \\ & (\mathbf{a}_k;\mathbf{e}_i)^\top \mathbf{K}(\mathbf{a}_k;\mathbf{e}_i) \le d_{ik}^2 \qquad \forall (i,k) \in E_p^{ax} \\ & \mathbf{H} = \mathbf{X}^\top \mathbf{X} \\ & \mathbf{K} = \begin{pmatrix} \mathbf{I} & \mathbf{X} \\ \mathbf{X}^\top & \mathbf{H} \end{pmatrix}.\end{aligned} \tag{11}$$

The constraint $\mathbf{H} = \mathbf{X}^\top \mathbf{X}$ fixes the rank of $\mathbf{H}$ and is not convex. To mitigate, we relax it into $\mathbf{H} \succeq \mathbf{X}^\top \mathbf{X}$. In the following section we prove that this weaker constraint is sufficient to obtain MVU-feasible solutions. The Schur Complement Lemma (Boyd & Vandenberghe, 2004) states that $\mathbf{H} \succeq \mathbf{X}^\top \mathbf{X}$ if and only if $\mathbf{K} \succeq 0$, which we enforce as an additional constraint:

$$\begin{aligned}\max_{\mathbf{X},\mathbf{H}} \quad & \text{trace}(\mathbf{H}) \\ \text{s.t.} \quad & (\mathbf{0};\mathbf{e}_{ij})^\top \mathbf{K}(\mathbf{0};\mathbf{e}_{ij}) \le d_{ij}^2 \qquad \forall (i,j) \in E_p^{xx} \\ & (\mathbf{a}_k;\mathbf{e}_i)^\top \mathbf{K}(\mathbf{a}_k;\mathbf{e}_i) \le d_{ik}^2 \qquad \forall (i,k) \in E_p^{ax} \\ & \mathbf{K} = \begin{pmatrix} \mathbf{I} & \mathbf{X} \\ \mathbf{X}^\top & \mathbf{H} \end{pmatrix} \succeq 0.\end{aligned} \tag{12}$$

The optimization (12) is convex and scales $O((n_p + d)^3)$. It monotonically increases the objective in (7) and converges to a fixed point. For a maximum patch-size m, *i.e.* $n_p \le m$ for all p, each iteration of MVC scales *linearly* with respect to n, with complexity $O(\lceil \frac{n}{m} \rceil (m+d)^3)$. As the choice of m is independent of n, it can be fixed to a medium-sized value *e.g.* $m \approx 500$ for maximum efficiency. The $r \approx \lceil \frac{n}{m} \rceil$ sub-problems are completely independent and can be solved *in parallel*, leading to almost perfect parallel speed-up on computing clusters. Algorithm 1 states MVC in pseudo-code.

3.2. MVU feasibility

We prove that the MVC algorithm returns a feasible MVU solution and consequently gives rise to a well defined Euclidean Heuristic. First we need to show that the relaxation from $\mathbf{H} = \mathbf{X}^\top \mathbf{X}$ to $\mathbf{H} \succeq \mathbf{X}^\top \mathbf{X}$ does not cause any constraint violations.

Algorithm 1 MVC (V,E)

```
compute initial solution X with gl-MVU or Isomap
center and rescale X according to (6)
repeat
    identify r random sub-graphs (V_1, E_1), ..., (V_r, E_r)
    parfor p=1 to r do
        solve (12) for (V_p, E_p) to obtain X_p
    end parfor
    concatenate all X_p into X and center.
until variance of embedding X has converged.
return X
```

Lemma 1. The solution $\mathbf{X}$ of (12) satisfies all constraints in (8).

Proof. We first focus on constraints on $(i,j) \in E_p^{xx}$. The first constraint in (12) guarantees

$$\mathbf{H}_{ii} - 2\mathbf{H}_{ij} + \mathbf{H}_{jj} \le d_{ij}^2. \tag{13}$$

The last constraint of (12) and the Schur Complement Lemma enforce that $\mathbf{H} - \mathbf{X}^\top \mathbf{X} \succeq 0$. Thus,

$$\begin{aligned} & \mathbf{e}_{ij}^\top (\mathbf{H} - \mathbf{X}^\top \mathbf{X}) \mathbf{e}_{ij} \ge 0 \\ \Leftrightarrow \quad & \mathbf{e}_{ij}^\top (\mathbf{X}^\top \mathbf{X}) \mathbf{e}_{ij} \le \mathbf{e}_{ij}^\top \mathbf{H} \mathbf{e}_{ij} \qquad (14) \\ \Leftrightarrow \quad & \mathbf{x}_i^2 - 2\mathbf{x}_i^\top \mathbf{x}_j + \mathbf{x}_j^2 \le \mathbf{H}_{ii} - 2\mathbf{H}_{ij} + \mathbf{H}_{jj} \\ \Leftrightarrow \quad & \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \le \mathbf{H}_{ii} - 2\mathbf{H}_{ij} + \mathbf{H}_{jj}. \qquad (15)\end{aligned}$$

The first result follows from the combination of (13) and (15). Concerning constraints $(i,j) \in E_p^{ax}$, the second constraint in (12) guarantees that

$$\mathbf{a}_k^2 - 2\mathbf{a}_k^\top \mathbf{x}_i + \mathbf{H}_{ii} \le d_{ik}^2. \tag{16}$$

With a similar reasoning as for (14) we obtain $\mathbf{e}_i^\top (\mathbf{X}^\top \mathbf{X}) \mathbf{e}_i \le \mathbf{e}_i^\top \mathbf{H} \mathbf{e}_i$ and therefore $\mathbf{x}_i^2 \le \mathbf{H}_{ii}$. Combining this inequality with (16) leads to the result:

$$\|\mathbf{a}_k - \mathbf{x}_i\|_2^2 \le \mathbf{a}_k^2 - 2\mathbf{a}_k \mathbf{x}_i + \mathbf{H}_{ii} \le d_{ik}^2. \blacksquare$$

Theorem 1. The embedding obtained with the MVC Algorithm 1 is in the feasible set of (1).

Proof. We apply an inductive argument. The initial solution after centering and re-scaling according to (6) is MVU feasible by construction. By **Lemma 1**, the solution of (12) for each patch satisfies all constraints in E_p^{xx} and E_p^{ax} in (8). As each distance constraint in (7) is associated with exactly one patch, all its constraints in E^{xx} and E^{ax} are satisfied. Constraints in E^{aa} are fixed and satisfied by the induction hypothesis. Centering $\mathbf{X}$ satisfies the last constraint in (7) and leaves all distance constraints unaffected. As (7) is equivalent to (1), we obtain an MVU feasible solution at the end of each iteration in Algorithm 1, which concludes the proof. $\blacksquare$

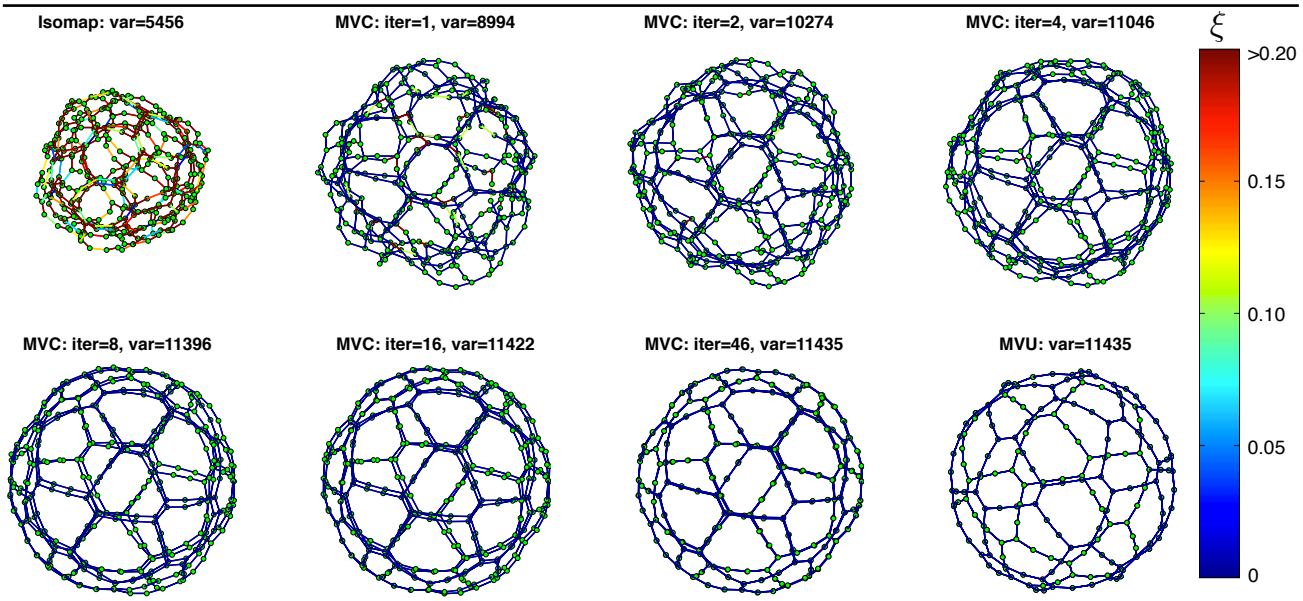

Figure 2. Visualization of several MVC iterations on the 5-puzzle data set ($m = 30$). The edges are colored proportional to their relative admissibility gap ξ, as defined in (17). The top left image shows the (rescaled) Isomap initialization. The successive graphs show that MVC decreases the edge admissibility gaps and increases the variance with each iteration (indicated in the title of each subplot) until it converges to the same variance as the MVU solution (bottom right).

4. Experimental Result

We evaluate our algorithm on a real world shortest path application data set and on two well-known benchmark AI problems.

Game Maps is a real world map dataset with 3,155 states from the international success multi-player game *Biowares Dragon Age: Origins*^{TM}.[3] A game map is a maze that consists of empty spaces (states) and obstacles. Cardinal moves take unit costs while diagonal moves cost 1.5. The search problem is to find an optimal path between a given start and goal state, while avoiding all obstacles. Although not large-scale, this data set is a great example for an application where the search heuristic is of extraordinary importance. Speedy solvers are essential to reduce upkeep costs and to ensure a positive user experience. In the game, many player and non-player characters interact and search problems have to be solved frequently as agents move. The shortest path solutions cannot be cached as the map changes dynamically with player actions.

M-Puzzle Problem (Jones, 2008) is a NP-hard sliding puzzle, often used as a benchmark problem for search algorithms/heuristics. It consists of a frame of M square tiles and one tile missing. All tiles are numbered and a state constitutes any order from which a path to the unique state with sorted (increasing) tiles exists. An action is to move a cardinal neighbor tile of the empty space into the empty space. The task is to find a shortest action sequence from a pre-defined start to a goal state. We evaluate our algorithm on the 5- (for visualization), 7- and 8-puzzle problem (3×2, 4×2 and 3×3 frames), which contain 360, 20160 and 181440 states respectively.

Blocks World (Gupta & Nau, 1992) is a NP-hard problem with the goal to build several pre-defined stacks out of a set of numbered blocks. Blocks can be placed on the top of others or on the ground. Any block that is currently under another block cannot be moved. The goal is to find a minimum action sequence from a start state to a goal state. We evaluate our algorithm on block world problems with 6 blocks (4,051 states) and 7 blocks (37,633 states), respectively.

Problem characteristics. The three types of problems not only feature different sized state spaces but also have different state space characteristics. Game maps has random obstacles that prevents movement for some state pairs, and thus has an irregular state space. The puzzle problems have a more regular search space (which lie on the surface of a sphere, see figure 2) with stable out-degree for each state. The state space of the blocksworld problems is also regular (it lies inside a sphere); however, the out-degree varies largely across states. For example, in 7-blocks, the state in which every block is placed on the ground has 42 edges, while the state in which all blocks are stacked in a single column has only 1 edge. We set $d_{ij} = 1$ for all edges in blocksworld and M-puzzle problems.

[3] http://en.wikipedia.org/wiki/Dragon_Age:_Origins

Table 1. Relative A^* search speedup over the differential heuristic (in expanded nodes) and embedding variance ($\times 10^5$).

	game map		6-blocksworld		7-puzzle		7-blocksworld		8-puzzle	
Method	speedup	var	speedup	var	speedup	var	speedup	var	speedup	var
Diff. Heuristic	1	N/A	1	N/A	1	N/A	1	N/A	1	N/A
Eigenmap	0.32	0.88	0.66	0.058	0.81	3.52	0.61	0.50	0.76	13.47
Isomap	0.50	12.13	0.61	0.046	0.84	3.73	0.65	0.46	0.67	10.62
MVU	**1.12**	**37.27**	1.23	0.154	N/A	N/A	N/A	N/A	N/A	N/A
gl-MVU	0.41	7.54	1.18	0.138	1.14	6.66	1.05	1.20	0.88	17.79
MVC-10 (eigenmap)	0.88	31.31	1.49	0.22	1.41	9.59	1.33	1.88	1.47	43.48
MVC-10 (isomap)	1.09	36.96	1.56	0.22	1.43	9.62	1.25	1.71	1.45	43.08
MVC-10 (gl-mvu)	0.90	32.98	1.96	0.27	1.45	9.82	1.67	2.27	1.52	45.75
MVC (eigenmap)	1.06	35.92	2.08	0.29	1.45	**9.86**	2.17	2.93	1.54	46.52
MVC (isomap)	**1.12**	37.22	2.22	0.30	**1.47**	9.85	**2.22**	**2.95**	1.54	46.58
MVC (gl-mvu)	1.11	36.47	**2.27**	**0.30**	1.45	**9.86**	**2.22**	**2.95**	**1.61**	**49.06**

Experimental setting. Besides MVC, we evaluate four graph embedding algorithms: MVU (Weinberger & Saul, 2006), Isomap (Tenenbaum et al., 2000b), (Laplacian) Eigenmap (Belkin & Niyogi, 2002) and gl-MVU (Wu et al., 2009). The last three are used as initializations for MVC. Following Rayner et al. 2011, the embedding dimension is $d = 3$ for all experiments. For gl-MVU, we set the dimension of graph Laplacian to be 40. For datasets of size greater than 10K, we set 10K landmarks for Isomap. For MVC we use a patch-size of $m = 500$ throughout (for which problem (12) can be solved in less than $20s$ on our lab desktops).

Visualization of MVC iterations ($m = 30$). Figure 2 visualizes successive iterations of the $d = 3$ dimensional MVC embedding of the 5−puzzle problem. All edges are colored proportionally to their relative admissibility gap,

$$\xi_{ij} = \frac{d_{ij} - ||\mathbf{x}_i - \mathbf{x}_j||}{||\mathbf{x}_i - \mathbf{x}_j||}. \tag{17}$$

The plot in the top left shows the original Isomap initialization after re-scaling, as defined in (6). The plot in the bottom right shows the actual MVU embedding from (2)—which can be computed precisely because of the small problem size. Intermediate plots show the embeddings after several iterations of MVC. Two trends can be observed: 1. the admissibility gap decreases with MVC (all edges are blue in the final embedding) and 2. the variance $\sum_i \mathbf{x}_i^2$ of the embedding, *i.e.* the MVU objective, increases monotonically. The final embedding has the same variance as the actual MVU embedding. The figure also shows that the 5−puzzle state space lies on a sphere, which is a beautiful example that visualization of states spaces in itself can be valuable. For example, the discovery of specific topological properties might lead to a better understanding of the state space structure and aid the development of problem specific heuristics.

Setup. As a baseline heuristic, we compare all results with a differential heuristic (Ng & Zhang, 2002). The differential heuristic pre-computes the exact distance from all states to a few pivot points in a (randomly chosen) set $S \subseteq V$. The graph distance between two states a, b is then approximated with $\max_{s \in S} |\delta(a, s) - \delta(b, s)| \leq \delta(a, b)$. In our experiments we set the number of pivots to 3 so that differential heuristics and embedding heuristics share the same memory limit. Figure 3 (*right*) shows the total expanded nodes as a function of the solution length, averaged over 100 start/goal pairs for each solution length. The figure compares MVC with various initializations, the differential heuristic and the MVU algorithm on the 6-blocksworld puzzle. Speedups (reported in Table 1) measure the reduction in expanded states during search, relative to the differential heuristic, averaged over 100 random (start, goal) pairs across all solution lengths.

Comprehensive evaluation. Table 1 shows the A^* search performances of Euclidean heuristics obtained by the MVC initializations, MVC after only 10 iterations (MVC-10) and after convergence (bottom section). Table 1 also shows the MVU objective/variance, $\sum_{\mathbf{x} \in V} \mathbf{x}_i^2$, of each embedding. Several trends can be observed: 1. MVC performs best when initialized with gl-MVU—this is not surprising as gl-MVU has a similar objective and is likely to lead to better initializations; 2. all MVC embeddings lead to drastic speedups over the differential heuristics; 3. the variance is highly correlated with speedup—supporting Rayner et al. (2011) that the MVU objective is well suited to learn Euclidean heuristics; 4. even MVC after only 10 iterations already outperforms the differential heuristic on almost all data sets. The consistency of the speedups and their unusually high factors (up to 2.22) show great promise for MVC as an embedding algorithm for Euclidean heuristics.

Exceeding MVU. On the 6-blocksworld data set in table 1, the variance of MVC actually exceeds that of MVU. In other words, the MVC algorithm finds a better solution for (1). This phenomenon can be explained by the fact that the convex MVU problem (2) is rank-relaxed and the final embedding is ob-

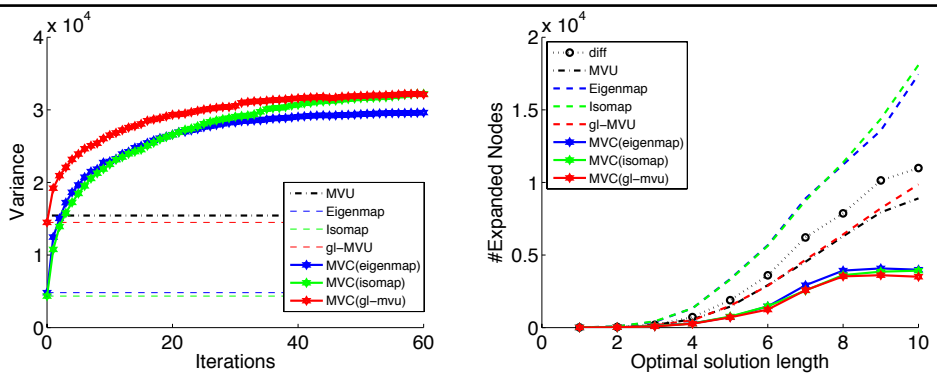

Figure 3. (*Left*) the embedding variance of 6-blocksworld plotted over 30 MVC iterations. The variance increases monotonically and even outperforms the actual MVU embedding (Weinberger & Saul, 2006) after only a few iterations. (*Right*) the number of expanded nodes in A^* search as a function of the optimal solution length. All MVC solutions strictly outperform the Differential Heuristic (*diff*) and even expand fewer nodes than MVU.

Table 2. Training time for MVU (Weinberger et al., 2005) and MVC, reported after initialization, the first 10 iterations (MVC-10), and after convergence.

Method	game	6-block	7-puzz	7-block	8-puzz
$\|V\|$	3,155	4,051	20,160	37,633	181,440
MVU	3h	10h	N/A	N/A	N/A
Eigenmap	1s	1s	4s	2m	7m
Isomap	14s	57s	3m	4m	32m
gl-MVU	2m	1m	1m	8m	15m
MVC-10 (eig)	20m	2m	14m	9m	2h
MVC-10 (iso)	15m	3m	20m	11m	3h
MVC-10 (glm)	21m	3m	18m	14m	3h
MVC (eig)	36m	9m	72m	53m	6h
MVC (iso)	17m	8m	56m	51m	7h
MVC (glm)	34m	5m	26m	33m	7h

tained after projection into a $d = 3$ dimensional sub-space. As MVC performs its optimization directly in this d-dimensional sub-space, it can find a better *rank-constrained* solution. This effect is further illustrated in Figure 3 (*left*), which shows the monotonic increase of the embedding variance as a function of the MVC iterations (on 6-blocksworld). After only a few iterations, all three MVC runs exceeds the MVU solution. A similar effect is utilized by Shaw & Jebara (2007), who optimize MVU in lower dimensional spaces directly (but cannot scale to large data sets). Figure 3 (*left*) also illustrates the importance of initialization: MVC initialized with Isomap and gl-mvu converge to the same (possibly globally optimal) solution, whereas the run with Laplacian Eigenmaps initialization is stuck in a sub-optimal solution. Our findings are highly encouraging and show that we might not only approximate MVU effectively on very large data sets, but actually *outperform it* if the intrinsic dimensionality of the data is higher than the desired embedding dimensionality d.

Embedding time. Table 2 shows the training time required for the convex MVU algorithm (2), three MVC initializations (Eigenmap, Isomap, gl-MVU), the time for 10 MVC iterations and the time until MVC converges, across all five data sets. Note that for real deployment, such as GPS systems, MVC only needs to be run once to obtain the embedding. The online calculations of Euclidean heuristics are very fast. All embeddings were computed on an off-the-shelve desktop with two 8-core Intel(R) Xeon(R) processors of 2.67 GHz and 128GB of RAM. Our MVC implementation is in MATLAB^{TM} and uses CSDP (Borchers, 1999) as the SDP solver. We parallelize each run of MVC on eight cores. All three initializations require roughly similar time (although Laplacian Eigenmaps is the fastest on all data sets), which is only a small part of the overall optimization. Whereas MVU requires 10 hours for graphs with $|V| = 4051$ (and cannot be executed on larger problems), we can find a superior solution to the same problem in 5 minutes and are able to run MVC on 45× larger problems in only 7 hours.

5. Conclusion

We have presented MVC, an iterative algorithm to transform graph embeddings into MVU feasible solution. On several small-sized problems where MVU can finish, we show that MVC gives comparable or even better solutions than MVU. We apply MVU on data sets of unprecedented sizes ($n = 180,000$) and, because of *linear* scalability, expect future (parallel) implementations to scale to graphs with millions of nodes. By satisfying all MVU constraints, MVC embeddings are provably well-defined Euclidean heuristics for A^* search and unleash an exciting new area of applications to all of manifold learning. We hope it will fuel new research directions in both fields.

Acknowledgements. WC and YC are supported in part by NSF grants CNS-1017701 and CCF-1215302. KQW is supported by NIH grant U01 1U01NS073457-01 and NSF grants 1149882 and 1137211. The authors thank Lawrence K. Saul for many helpful discussions.

References

Belkin, M. and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2002.

Biswas, P. and Ye, Y. Semidefinite programming for ad hoc wireless sensor network localization. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, IPSN '04, pp. 46–54, New York, NY, USA, 2004. ACM.

Blitzer, J., Weinberger, K.Q., Saul, L. K., and Pereira, F. C. N. Hierarchical distributed representations for statistical language modeling. In *Advances in Neural and Information Processing Systems*, volume 17, Cambridge, MA, 2005. MIT Press.

Borchers, B. Csdp, ac library for semidefinite programming. *Optimization Methods and Software*, 11(1-4):613–623, 1999.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

Donoho, D.L. and Grimes, C. *When does isomap recover the natural parameterization of families of articulated images?* Department of Statistics, Stanford University, 2002.

Gupta, N. and Nau, D.S. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2):223–254, 1992.

Jones, M.T. *Artificial Intelligence: A Systems Approach: A Systems Approach*. Jones & Bartlett Learning, 2008.

Kruskal, J.B. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

Lee, J.A. and Verleysen, M. *Nonlinear dimensionality reduction*. Springer, 2007.

Ng, T.S.E. and Zhang, H. Predicting internet network distance with coordinates-based approaches. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pp. 170–179. IEEE, 2002.

Paprotny, A., Garcke, J., and Fraunhofer, S. On a connection between maximum variance unfolding, shortest path problems and isomap. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*. MIT Press, 2012.

Platt, J.C. Fast embedding of sparse music similarity graphs. *Advances in Neural Information Processing Systems*, 16:571578, 2004.

Rayner, C., Bowling, M., and Sturtevant, N. Euclidean Heuristic Optimization. In *Proceedings of the Twenty-Fifth National Conference on Artificial Intelligence (AAAI)*, pp. 81–86, 2011.

Russell, S. J. and Norvig, Peter. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

Saul, L.K., Weinberger, K.Q., Ham, J. H., Sha, F., and Lee, D. D. *Spectral methods for dimensionality reduction*, chapter 16, pp. 293–30. MIT Press, 2006.

Shaw, B. and Jebara, T. Minimum volume embedding. In *Proceedings of the 2007 Conference on Artificial Intelligence and Statistics*. MIT press, 2007.

Shaw, B. and Jebara, T. Structure preserving embedding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 937–944, New York, NY, USA, 2009. ACM.

Silva, V. and Tenenbaum, J.B. Global versus local methods in nonlinear dimensionality reduction. *Advances in neural information processing systems*, 15:705–712, 2002.

Talwalkar, A., Kumar, S., and Rowley, H. Large-scale manifold learning. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008.

Tenenbaum, J. B., Silva, V., and Langford, J. C. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000a.

Tenenbaum, J.B., De Silva, V., and Langford, J.C. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000b.

Vasiloglou, N., Gray, A.G., and Anderson, D.V. Scalable semidefinite manifold learning. In *Machine Learning for Signal Processing, 2008. MLSP 2008. IEEE Workshop on*, pp. 368–373. IEEE, 2008.

Weinberger, K.Q. and Saul, L.K. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70:77–90, 2006. ISSN 0920-5691.

Weinberger, K.Q., Packer, B. D., and Saul, L. K. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, Barbados, West Indies, 2005.

Weinberger, K.Q., Sha, F., Zhu, Q., and Saul, L. Graph laplacian regularization for large-scale semidefinite programming. In *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

Wu, X., So, A. Man-Cho, Li, Z., and Li, S.R. Fast graph laplacian regularized kernel learning via semidefinite quadratic linear programming. In *Advances in Neural Information Processing Systems 22*, pp. 1964–1972. 2009.

Zhang, T., Tao, D., Li, X., and Yang, J. Patch alignment for dimensionality reduction. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1299–1313, 2009.