Robust Spectral Clustering for Noisy Data

Modeling Sparse Corruptions Improves Latent Embeddings

Aleksandar Bojchevski
Technical University of Munich
a.bojchevski@in.tum.de

Yves Matkovic
Technical University of Munich
matkovic@in.tum.de

Stephan Günnemann
Technical University of Munich
guennemann@in.tum.de

ABSTRACT

Spectral clustering is one of the most prominent clustering approaches. However, it is highly sensitive to noisy input data. In this work, we propose a robust spectral clustering technique able to handle such scenarios. To achieve this goal, we propose a sparse and latent decomposition of the similarity graph used in spectral clustering. In our model, we jointly learn the spectral embedding as well as the corrupted data – thus, enhancing the clustering performance overall. We propose algorithmic solutions to all three established variants of spectral clustering, each showing linear complexity in the number of edges. Our experimental analysis confirms the significant potential of our approach for robust spectral clustering. Supplementary material is available at www.kdd.in.tum.de/RSC.

CCS CONCEPTS

•Computing methodologies →Machine learning approaches; Unsupervised learning; Spectral methods; •Information systems →Data mining; Clustering;

1 INTRODUCTION

Clustering is one of the fundamental data mining tasks. Among the variety of methods that have been introduced in the literature [1], spectral clustering [20] is one of the most prominent and successful approaches. It has been successfully applied in many domains ranging from computer vision to network analysis.

Since spectral clustering relies on a similarity graph only (e.g. connecting each instance with its m nearest neighbors), it is applicable to almost any data type, with vector data being the most frequent case. Spectral clustering embeds the data instances into a vector space that is spanned by the k eigenvectors corresponding to the k smallest eigenvalues of the graph's (normalized) Laplacian matrix. By clustering in this space, even complex structures can be detected – such as the half-moon data shown in Fig. 1 (left).

While spectral clustering is widely used in practice, one big issue is rarely addressed: it is highly sensitive to noisy input data. Fig. 1 illustrates this effect. While for the data on the left spectral clustering perfectly recovers the ground-truth clusters, the scenario on the right – with only slightly perturbed data – leads to a completely wrong clustering for any of the three established versions [20] of spectral clustering. Spectral clustering fails in such scenarios.

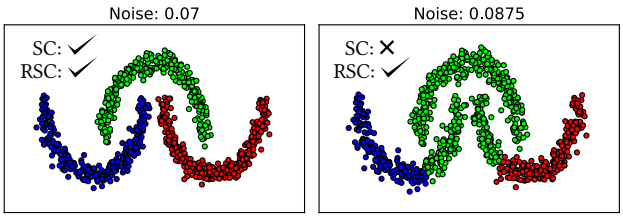

Figure 1: Spectral clustering (SC) is sensitive to noisy input. Left: SC detects the clustering. Right: SC fails. Our method (RSC) is successful in both scenarios.

In this work, we introduce a principle to robustify spectral clustering. The core idea is that the observed similarity graph is not perfect but corrupted by errors. Thus, instead of operating on the original graph – or performing some, often arbitrary, data cleaning that precedes the analysis – we assume the graph to be decomposed into two *latent* factors: the clean data and the corruptions. Following the idea that corruptions are sparse, we *jointly* learn the latent corruptions and the latent spectral embedding using the clean data.

For tasks such as regression [18], PCA [2], and autoregression [6, 8], such ideas have shown to significantly outperform non-robust techniques. And, indeed, also our method – called RSC – leads to clusterings that are more robust to corruptions. *In Fig. 1 (right) our approach is able to detect the correct clustering structure.* More precisely, our work is based on a sparse latent decomposition of the graph with the aim to optimize the *eigenspace of the graph's Laplacian*. This is in strong contrast to, e.g., robust PCA where the decomposition is guided by the eigenspace of the data itself. In particular, different Laplacians affect the eigenspace differently and require different solutions.

We note that the focus of this work is not on finding the number of clusters automatically. Principles using, e.g., the largest eigenvalue gap [14] might similarly be applied to our work. We left this aspect for future work. Overall, our contributions are:

- **Model:** We introduce a model for robust spectral clustering that handles noisy input data. Our principle is based on the idea of sparse latent decompositions. This is the first work exploiting this principle for spectral clustering, in particular tackling also the challenging case of normalized Laplacians.
- **Algorithms:** We provide algorithmic solutions for our model for all three established versions of spectral clustering using different Laplacian matrices. For our solutions we relate to principles such as Eigenvalue perturbation and the multidimensional Knapsack problem. In each case, the complexity of the overall method is linear in the number of edges.

KDD'17, August 13–17, 2017, Halifax, NS, Canada.
 ISBN 978-1-4503-4887-4/17/08…$15.00.
DOI: http://dx.doi.org/10.1145/3097983.3098156

- **Experiments:** We conduct extensive experiments showing the high potential of our method, with up to 15 percentage points improvement in accuracy on real-world data compared to standard spectral clustering. Moreover, we propose two novel measures – *local purity* and *global separation* – which enable us to evaluate the intrinsic quality of an embedding without relying on a specific clustering technique.

2 PRELIMINARIES

We start with some basic definitions required in our work. Let $\boldsymbol{A}$ be a matrix, we denote with $\boldsymbol{a}_i$ the i-th *row*-vector of $\boldsymbol{A}$ and with $a_{i,j}$ the value at position i, j. A similarity graph is represented by a symmetric adjacency matrix $\boldsymbol{A} \in (\mathbb{R}_{\geq 0})^{n \times n}$, with n being the number of instances. We denote the set of *undirected edges* as $\mathcal{E} = \{(i,j) \mid a_{i,j} > 0 \wedge i > j\}$. The set of edges incident to node i is given by $\mathcal{E}_i = \{(x,y) \in \mathcal{E} \mid x = i \vee y = i\}$. The vector representing the edges of $\boldsymbol{A}$ is written as $[a_{i,j}]_{(i,j)\in\mathcal{E}} = [a_e]_{e\in\mathcal{E}}$.

We denote with $d_i = \sum_j a_{i,j}$ the degree of node i, and with $\boldsymbol{D}(\boldsymbol{A}) = diag(d_1, \ldots, d_n)$ the diagonal matrix representing all degrees. We denote with $\boldsymbol{I}$ the identity matrix, whose dimensionality becomes clear from the context. Furthermore, as required for spectral clustering, we introduce different notions of Laplacian matrices:

- unnormalized Laplacian: $\boldsymbol{L}(\boldsymbol{A}) = \boldsymbol{D}(\boldsymbol{A}) - \boldsymbol{A}$
- normalized Laplacians: $\boldsymbol{L}_{rw}(\boldsymbol{A}) = \boldsymbol{D}(\boldsymbol{A})^{-1}\boldsymbol{L}(\boldsymbol{A})$
  and $\boldsymbol{L}_{sym}(\boldsymbol{A}) = \boldsymbol{D}(\boldsymbol{A})^{-1/2}\boldsymbol{L}(\boldsymbol{A})\boldsymbol{D}(\boldsymbol{A})^{-1/2}$

2.1 Spectral Clustering

Spectral clustering can be briefly summarized in three steps (see [20] for details). Step 1: Construct the similarity graph $\boldsymbol{A}$. Different principles for the similarity graph construction exist. We focus on the symmetric x-nearest-neighbor graph, as it is recommended by [20] – any other construction can be used as well. Thus, the graph $\boldsymbol{A}$ is given by $a_{i,j} = 1$ if i is a x nearest neighbor of j or vice versa, and $a_{i,j} = 0$ else.

Step 2: Depending on the considered Laplacian, the next step is to compute the following eigenvectors[1]:

- $\boldsymbol{L}(\boldsymbol{A})$: k first eigenvectors of $\boldsymbol{L}(\boldsymbol{A})$
- $\boldsymbol{L}_{rw}(\boldsymbol{A})$: k first generalized eigenv. of $\boldsymbol{L}(\boldsymbol{A})\boldsymbol{u} = \lambda\boldsymbol{D}(\boldsymbol{A})\boldsymbol{u}$
- $\boldsymbol{L}_{sym}(\boldsymbol{A})$: k first eigenvectors of $\boldsymbol{L}_{sym}(\boldsymbol{A})$

This step stems from the fact that spectral clustering tries to obtain solutions that minimize the ratio-cut/normalized-cut in the similarity graph. As shown in [20], an approximation to, e.g., the ratio-cut is obtained by the following trace minimization problem

$$\min_{\boldsymbol{H}\in\mathbb{R}^{n\times k}} Tr(\boldsymbol{H}^T\boldsymbol{L}(\boldsymbol{A})\boldsymbol{H}) \text{ subject to } \boldsymbol{H}^T\boldsymbol{H} = \boldsymbol{I} \tag{1}$$

The solution being the k first eigenvectors of the Laplacian $\boldsymbol{L}$ as stated above. Similar trace minimization problems can be formulated for the other Laplacians. We denote with $\boldsymbol{H} \in \mathbb{R}^{n\times k}$ the matrix storing the eigenvectors as columns.

Step 3: Clustering on $\boldsymbol{H}$. The spectral embedding of each instance i is given by the i-th *row* of $\boldsymbol{H}$. To find the final clustering, the vectors $\boldsymbol{h}_i$ are (in case of $\boldsymbol{L}_{sym}$ first normalized and then) clustered using, e.g., k-means.

[1]We denote with 'k first' eigenvectors, those k eigenvectors refering to the k smallest eigenvalues.

3 RELATED WORK

Multiple principles to improve spectral clustering have been introduced – focusing on different kinds of robustness. Surprisingly, many of the techniques [9, 11, 14, 23] are based on fully connected similarity graphs – even though nearest neighbor graphs are recommended [20]. First, using fully connected graphs highly increases the runtime – the considered matrices are no longer sparse – and, second, one has to select an appropriate scaling factor σ, required, e.g., for the Gaussian Kernel when constructing the graph (see [20]). Thus, many techniques [9, 11, 14, 23] focus on robustness regarding the parameter σ.

Local similarity scaling: [23] introduces a principle where the similarity is locally scaled per instance, i.e. the parameter σ changes per instance. By doing so, an improved similarity graph is obtained that better separates dense and sparse areas in the dataspace. The work [11] has extended this principle by using a weighted local scaling. The methods work well on noise-free data; however, they are still sensitive to noisy inputs.

Laplacian smoothing: [9] considers the problem of noisy data similar to our work, and they propose a principle of eigenvector smoothing. The initial Laplacian matrix is replaced by a smoothed version $M = \sum_{i=2}^{n} \frac{1}{\gamma+\lambda_i}\boldsymbol{x}_i \cdot \boldsymbol{x}_i^T$ where $\boldsymbol{x}_i$ and λ_i are the eigenvectors/values of the original Laplacian matrix. Clustering is then performed on the eigenvectors of the matrix M. A significant drawback is that a full eigenvalue decomposition is required.

Data warping: [14] focuses on data where uniform noise has been added; not noisy data itself. They propose the principle of data warping. Intuitively, the data is transformed to a new space where noise points form its own cluster. Since they focus on fully connected graphs, noise can easily be detected by inspecting points with the lowest overall similarity. ***Since [9] and [14] are the most closely related works to our principle, we compare against them in our experiments.***

Feature weighting: Focusing on a different scenario, multiple works have considered noisy/irrelevant features. In [10] a global feature weighting is learned in a semi-supervised fashion, thus, leading to an improved similarity matrix. [24] learns an affinity matrix based on random subspaces focusing on discriminative features. In [7], inspired by the idea of subspace clustering, feature weights are learned locally per cluster.

All the above techniques (except [7]) follow a *two-step, sequential* approach: They first construct an improved similarity graph/ Laplacian and then apply standard spectral clustering. In contrast, our method *jointly* learns the similarity graph and the spectral embedding. Both steps repeatedly benefit from each other.

Besides the above works focusing on general spectral clustering, different extended formulations have been introduced: [13] considers hypergraphs to improve robustness, [3] uses path-based characteristics. None of the techniques jointly learns a similarity matrix and the spectral embedding. Not focusing on robustness w.r.t. noise, [21] computes a doubly stochastic matrix by imposing low-rank constraints on the graph's Laplacian. It is restricted to the unnormalized Laplacian and leads to dense graphs, making it impractical for large data. Moreover, works such as [15] consider the problem of finding anomalous subgraphs using spectral principles, again not focusing on the case of noise.

We further note that the spectral analysis is not restrictred to a graph's Laplacian (as used in standard spectral clustering). The classical works of Davis-Kahan [5], for example, study the perturbation of a matrix X and the change of X's eigenspace. Following this line, [22] studies clustering based on the eigenspace of the adjacency matrix itself. In contrast, in this paper, we focus on the change of the eigenspace of $L(X)$. In particular, we also consider the case of normalized Laplacians, which often lead to better results [20].[2]

4 ROBUST SPECTRAL CLUSTERING

In the following, we introduce the major principle of our technique – called RSC. For illustration purposes, we will start with spectral clustering based on the unnormalized Laplacian. The (more complex) principles for normalized Laplacians are described in Sec. 5

Let $\boldsymbol{A} \in (\mathbb{R}_{\geq 0})^{n \times n}$ be the symmetric similarity graph extracted for the given data, with n being the number of instances in our data (see Sec. 2). *Our major idea is that the similarity graph $\boldsymbol{A}$ is not perfect but might be corrupted (e.g. due to noisy input data). Any analysis performed on $\boldsymbol{A}$ might lead to misleading results.*

Therefore, we assume that the observed graph $\boldsymbol{A}$ is obtained by two *latent* factors: $\boldsymbol{A}^c$ representing the corruptions and $\boldsymbol{A}^g$ representing the 'good' (clean) graph. More formally, we assume an additive decomposition[3], i.e.

$$\boldsymbol{A} = \boldsymbol{A}^g + \boldsymbol{A}^c \text{ with } \boldsymbol{A}^g, \boldsymbol{A}^c \in (\mathbb{R}_{\geq 0})^{n \times n} \text{ , both symmetric.}$$

Instead of performing the spectral clustering on the corrupted $\boldsymbol{A}$, our goal is to perform it on $\boldsymbol{A}^g$. *The core question is, how to find the matrices $\boldsymbol{A}^g$ and $\boldsymbol{A}^c$?* In particular since clustering is an unsupervised learning task we don't know which entries in $\boldsymbol{A}$ might be wrong. For solving this challenge, we exploit two core ideas:

1) Corruptions are relatively rare – if they were not rare, i.e. the majority of the data is corrupted, a reasonable clustering structure can not be expected. Technically, we assume the matrix $\boldsymbol{A}^c$ to be sparse.

Let θ denote the maximal number of corruptions a user expects in the data. We require $\|\boldsymbol{A}^c\|_0 \leq 2 \cdot \theta$ where

$$\|\boldsymbol{A}^c\|_0 := |\{(i,j) \mid a^c_{i,j} \neq 0\}|$$

denotes the element-wise $\boldsymbol{L}_0$ pseudo-norm ($2 \cdot \theta$ due to symmetry of the graph).

While θ constrains the number of corruptions *globally*, it is likewise beneficial to enforce sparsity *locally* per node. This can be realized by the constraint $\left\|\boldsymbol{a}^g_i\right\|_0 \geq m$ for each node i (or equivalently: $\left\|\boldsymbol{a}^c_i\right\|_0 \leq |\mathcal{E}_i| - m$; we chose the first version due to easier interpretability: each node in $\boldsymbol{A}^g$ will be connected to at least m other nodes). Note that θ and m control different effects. To ignore either global or local sparsity, one can simply set the parameter to its extreme value ($\theta = \frac{1}{2}\|\boldsymbol{A}\|_0$ or $m = 1$).

2) The detection of $\boldsymbol{A}^g/\boldsymbol{A}^c$ is steered by the clustering process, i.e., we *jointly* perform the spectral clustering and the decomposition of $\boldsymbol{A}$. This is in contrast to a sequential process where first the matrix is constructed and then the clustering is performed.

[2]Surprisingly, many advanced spectral works still consider only the easier case of unnormalized Laplacians. Our competitors [9, 14] handle normalized Laplacians.
[3]This general decomposition not only leads to good performance, as we will see later, but also facilitates easy interpretation.

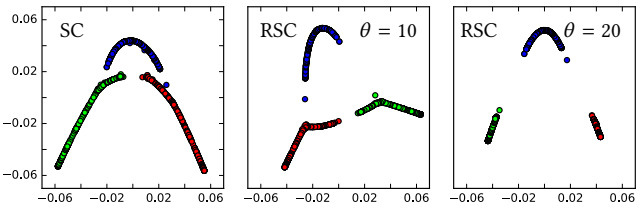

Figure 2: Spectral embeddings for data of Fig. 1 (right). Left: Spectral clustering; middle: RSC with $\theta = 10$, right: $\theta = 20$. RSC enhances the discrimination of points.

The strong advantage of a simultaneous detection is that we don't need to specify a separate – often arbitrary – objective for finding $\boldsymbol{A}^g$, but the process is complete determined by the underlying spectral clustering. More precise, we exploit the equivalence of spectral clustering to trace minimization problems (see Sec. 2.1, Eq. (1)). Intuitively, the value of the trace in Eq. (1) corresponds to an approximation of the ratio-cut in the graph $\boldsymbol{A}$. The smaller the value, the better the clustering. Thus, we aim to find the matrix $\boldsymbol{A}^g$ by minimizing the trace based on the Laplacian's eigenspace – subject to the sparsity constraints. Overall, our problem becomes:

Problem 1. *Given the matrix $\boldsymbol{A}$, the number of clusters k, the sparsity threshold θ, and the minimal number of nearest neighbors m per node. Find $\boldsymbol{H}^* \in \mathbb{R}^{n \times k}$ and $\boldsymbol{A}^{g*} \in (\mathbb{R}_{\geq 0})^{n \times n}$ such that*

$$(\boldsymbol{H}^*, \boldsymbol{A}^{g*}) = \underset{\boldsymbol{H}, \boldsymbol{A}^g}{argmin}\ Tr(\boldsymbol{H}^T \cdot \boldsymbol{L}(\boldsymbol{A}^g) \cdot \boldsymbol{H}) \tag{2}$$

subject to $\boldsymbol{H}^T \cdot \boldsymbol{H} = \boldsymbol{I}$ and $\boldsymbol{A}^g = \boldsymbol{A}^{g\,T}$ and
$\|\boldsymbol{A} - \boldsymbol{A}^g\|_0 \leq 2 \cdot \theta$ and $\left\|\boldsymbol{a}^g_i\right\|_0 \geq m \quad \forall i \in \{1, \ldots, n\}$

The crucial difference between Eq. (1) and Problem 1 is that we now *jointly* optimize the spectral embedding $\boldsymbol{H}$ and the similarity graph $\boldsymbol{A}^g$. The Laplacian matrix $\boldsymbol{L}(\boldsymbol{A}^g)$ is *no longer constant* but adaptive.

Figure 2 shows the strong advantage of this joint learning. Here, different spectral embeddings $\boldsymbol{H}$ (2nd and 3rd eigenvector since the 1st is constant) for the data in Fig. 1 (right) are shown. The left plot shows the embedding using *usual* spectral clustering. Due to the noisy input, the three groups are very close to each other and each spread out. Clustering on this embedding merges multiple groups and, thus, leads to low quality (for real-world data these embeddings look even harder as we will see in the experimental section). In contrast, the middle and right images show the spectral embedding learned by our technique when removing just 10 or 20 corrupted edges, respectively. Evidently, the learned embeddings highlight the clustering structure more clearly. Thus, by simultaneously learning the embedding and the corruptions, we improve the clustering quality.

4.1 Algorithmic Solution

While our general objective is hard to optimize (in particular due to the $\|.\|_0$ constraints the problem becomes NP-hard in general), we propose a highly efficient block coordinate-descent (alternating) optimization scheme to approximate it. That is, given $\boldsymbol{H}$, we optimize for $\boldsymbol{A}^g/\boldsymbol{A}^c$; and given $\boldsymbol{A}^g/\boldsymbol{A}^c$ we optimize for $\boldsymbol{H}$ (cf. Algorithm 1). Of course, since $\boldsymbol{A}^c$ determines $\boldsymbol{A}^g$ and vice versa, it is sufficient to focus on the update of one, e.g., $\boldsymbol{A}^c$. It is worth pointing out

that in many works, the $\|.\|_0$ norm is simply handled by relaxation to the $\|.\|_1$ norm. In our work, in contrast, we aim to *preserve the interpretability* of the $\|.\|_0$ norm; for this, we derive a connection to the multidimensional Knapsack problem.

Update of H: Given A^c, the update of H is straightfarward. Since $A^g = A - A^c$ and therefore $L(A^g)$ are now constant, we can simply refer to Eq. (2): finding H is a standard trace minimization problem. The solution of H are the k first eigenvectors of $L(A^g)$.

Update of A^c: Clearly, since A^c needs to be non-negative, for all elements (i,j) with $a_{i,j} = 0$, it also holds $a^c_{i,j} = 0$. Thus, in the following, we only have to focus on the elements $a^c_{i,j}$ with $(i,j) \in \mathcal{E}$, i.e. the vector $[a^c_e]_{e\in\mathcal{E}}$. We base our update on the following lemma:

Lemma 4.1. *Given H, the solution for A^c minimizing Eq.* (2) *can be obtained by* maximizing

$$f_1([a^c_e]_{e\in\mathcal{E}}) := \sum_{(i,j)\in\mathcal{E}} a^c_{i,j} \cdot \left\|\boldsymbol{h}_i - \boldsymbol{h}_j\right\|_2^2 \tag{3}$$

subject to the $\|.\|_0$ constraints and for each e: $a^c_e \in \{0, a_e\}$.

Proof. See appendix. □

Exploiting Lemma 4.1, our problem can equivalently be treated as a set selection problem. For this, let $\mathcal{X} \subseteq \mathcal{E}$ and $[v^{\mathcal{X}}_e]_{e\in\mathcal{E}} = \boldsymbol{v}^{\mathcal{X}} \in \mathbb{R}^{|\mathcal{E}|}$ be the vector with $v^{\mathcal{X}}_e = \begin{cases} a_{i,j} & \text{if } (i,j) = e \in \mathcal{X} \\ 0 & \text{else} \end{cases}$, our goal is to find a set $\mathcal{X}^* \subseteq \mathcal{E}$ maximizing $f_1(\boldsymbol{v}^{\mathcal{X}^*})$ subject to the constraints. Accordingly, Problem 1 can be represented as (a special case of) a multidimensional Knapsack problem [16] operating on the set of edges $\mathcal{E}$:

Corollary 4.2. *Given H. Let $\mathcal{X} = \{e \in \mathcal{E} \mid x_e = 1\}$ be the solution of the following multidimensional Knapsack problem: Find $x_e \in \{0,1\}$, $e \in \mathcal{E}$ such that $\sum_{e\in\mathcal{E}} x_e \cdot p_e$ is maximized subject to $\sum_{e\in\mathcal{E}} x_e \le \theta$ and $\forall i = 1,\dots,n : \sum_{e\in\mathcal{E}_i} x_e \le |\mathcal{E}_i| - m$ where*

$$p_e = p_{(i,j)} = a_{i,j} \cdot \left\|\boldsymbol{h}_i - \boldsymbol{h}_j\right\|_2^2 \tag{4}$$

The solution for A^c w.r.t. Eq. (2) *corresponds to $\boldsymbol{v}^{\mathcal{X}}$.*

This result matches the intuition of corrupted edges: The term p_e is high for instances whose embeddings are very dissimilar (i.e. they should not belong to the same cluster) but which are still connected by an edge.

While finding the optimal solution of a multidim. Knapsack problem is intractable, multiple efficient and effective approximate solutions exist [12, 16]. We exploit these approaches for our final algorithm. Following the principle of [12], we first sort the edges $e \in \mathcal{E}$ based on their ratio $p_e/\sqrt{s_e}$. Here, s_e is the number of constraints the variable x_e participates in. Since in our special case, *each* x_e participates in exactly three constraints, $s_e = 3$, it is sufficient to sort the edges based on the value p_e. We then construct a solution by adding one edge after another to A^c as long as the constraints are not violated. This approach leads to the best possible worst-case bound of $1/\sqrt{n+1}$ [12].

Algorithm 1 (lines 5-15) shows the update of A^c/A^g. Note that we do not need to sort the full edge set. It is sufficient to iteratively obtain the best edges. Thus, a priority queue PQ (e.g. a heap) is used (line 7, 10). The local $\|.\|_0$ constraints can simply be ensured by recording how many edges per node can still be removed (line 6, 13). Thus, an edge can only be included in the result (line 12) if the incident nodes allow to do so (line 11).

```
input : Similarity graph A, parameters k, θ, m
output: Clustering C_1, ..., C_k
1  A^g ← A;
2  while true do
       /* Update of H                                          */
3      Compute Laplacian, matrix H, and trace;
4      if Trace could not be lowered then break;
       /* Update of A^c/A^g                                    */
5      X = ∅ ;
6      for each node i set count_i ← |E_i| − m;
7      priority queue PQ on tuples (score, edge) ;
8      for each edge e ∈ E add tuple (p_e, e) to PQ if p_e > 0
                                   [Eq. (4) or Eq. (6)];
9      while PQ not empty do
10         get first element from PQ → (., e_best = (i, j)) ;
11         if count_i > 0 ∧ count_j > 0 then
12             X ← X ∪ {e_best};
13             count_i − −; count_j − −;
14             if |X| = θ then break;
15     construct A^c according to v^X; A^g = A − A^c ;
16 apply k-means on (normalized) vectors (h_i)_{i=1,...,n}
```

Algorithm 1: Robust spectral clustering

The overall method for robust spectral clustering using unnormalized Laplacians iterates between the two update steps (lines 3-15). Note that in each iteration, line 8 considers all edges of the original graph. Thus, an edge marked as corrupted in a previous iteration might be evaluated as non-corrupted later. The algorithm terminates when the trace can not been improved further. In the last step (line 16), the k-means clustering on the improved H matrix is performed as usual.

Complexity: Using a heap, the update of A^c can be computed in time $O(|\mathcal{E}| + \theta' \cdot \log|\mathcal{E}|)$, where $\theta' \le |\mathcal{E}|$ is the number of iterations of the inner while loop. Using power iteration, the eigenvectors H can be computed in time linear in the number of edges. Thus, overall, linear runtime can be achieved, as also verified empirically.

Sparse operations: It is worth mentioning that all operations performed in the algorithm operate on sparse data. This includes the computation of the Laplacian, its eigenvectors, and the constructions of A^c and A^g. Thus, even large datasets can easily be handled.

5 RSC: NORMALIZED LAPLACIANS

We now tackle the more complex cases of the two normalized Laplacians, which often lead to better clustering. For this, different algorithmic solutions are required.

5.1 Random Walk Laplacian

Spectral clustering based on L_{rw} corresponds to a *generalized* eigenvector problem using L [20]. Our problem definition becomes:

Problem 2. *Identical to Problem 1 but replacing the constraint $H^T \cdot H = I$ with $H^T \cdot D(A^g) \cdot H = I$.*

Again, our goal is to solve this problem via block-coordinate descent. While the update of H is clear (corresponding to the first k generalized eigenvectors w.r.t. $L(A^g)$ and $D(A^g)$), using the

same approach for $\boldsymbol{A}^c/\boldsymbol{A}^g$ as introduced in Sec. 4.1 turns out to be impractical: Since the constraint $\boldsymbol{H}^T \cdot \boldsymbol{D}(\boldsymbol{A}^g) \cdot \boldsymbol{H} = \boldsymbol{I}$ now also depends on $\boldsymbol{A}^g$, we get a highly restrictive constrained problem. As a solution, we propose a principle exploiting the idea of eigenvalue perturbation [19].

Using eigenvalue perturbation, we derive a matrix $\boldsymbol{A}^g$ aiming to minimize the sum of the k smallest *generalized* eigenvalues. Minimizing this sum is equivalent to minimizing the trace based on the normalized Laplacian's eigenspace. We obtain:

Lemma 5.1. *Given the eigenvector matrix $\boldsymbol{H}$ and the corresponding eigenvalues $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_k)$. An approximation of $\boldsymbol{A}^c$ minimizing the objective of Problem 2 can be obtained by* maximizing

$$f_2([a^c_e]_{e\in\mathcal{E}}) = \sum_{(i,j)\in\mathcal{E}} a^c_{i,j} \left(\left\| \boldsymbol{h}_i - \boldsymbol{h}_j \right\|_2^2 - \left\| \sqrt{\boldsymbol{\lambda}} \circ \boldsymbol{h}_i \right\|_2^2 - \left\| \sqrt{\boldsymbol{\lambda}} \circ \boldsymbol{h}_j \right\|_2^2 \right) \quad (5)$$

subject to the $\|.\|_0$ constraints and for each e: $a^c_e \in \{0, a_e\}$.

Here, $\sqrt{.}$ denotes the element-wise square-root of the vector elements, and $\circ$ the Hadamard product.

Proof. See appendix. □

Clearly, the solution of the unnormalized case (Eq. (3)) and the normalized case (Eq. (5)) are structural very similar – and for solving it we can use the same principle as before (Algorithm 1), simply using as edge scores now the values

$$p_e = p_{(i,j)} = a_{i,j} \left(\left\| \boldsymbol{h}_i - \boldsymbol{h}_j \right\|_2^2 - \left\| \sqrt{\boldsymbol{\lambda}} \circ \boldsymbol{h}_i \right\|_2^2 - \left\| \sqrt{\boldsymbol{\lambda}} \circ \boldsymbol{h}_j \right\|_2^2 \right) \quad (6)$$

Accordingly, also the complexity for finding $\boldsymbol{A}^c$ remains unchanged. Note that only edges with positive score need to be added to the queue (line 8).

Advantages: Comparing Eq. (6) with Eq. (4) one sees an additional 'penalty' term which takes the norm/length of the vectors $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ into account. Thereby, instances whose embeddings are far away from the origin get a lower (or even negative) score. *This aspect is highly beneficial for spectral clustering*: e.g., in the case of two clusters, the final clustering can be obtained by inspecting the sign of the 1d-embedding [20] – in general, intuitively speaking, clusters are separated by the origin (see Fig. 2 where the origin is in the center of the plots). *Instances that are far away from the origin can be clearly assigned to their cluster*; thus, marking their edges as corrupt might improve the clustering only slightly. In contrast, edges that are at the border between different clusters are the challenging ones – and exactly these are the ones preferred by Eq. (6).

5.2 Symmetric Laplacian

We now turn to the last case, spectral clustering using $\boldsymbol{L}_{sym}$.

Problem 3. *Identical to Problem 1 but replacing Eq.* (2) *with*

$$(\boldsymbol{H}^*, \boldsymbol{A}^{g*}) = \underset{\boldsymbol{H}, \boldsymbol{A}^g}{argmin}\ Tr(\boldsymbol{H}^T \cdot \boldsymbol{L}_{sym}(\boldsymbol{A}^g) \cdot \boldsymbol{H}) \quad (7)$$

Using alternating optimization, the matrix $\boldsymbol{H}$ can easily be updated when $\boldsymbol{A}^g$ is given. For updating the matrix $\boldsymbol{A}^g$ (or equivalently $\boldsymbol{A}^c$) we use the following result:

Lemma 5.2. *Given the eigenvector matrix $\boldsymbol{H}$. The matrix $\boldsymbol{A}^c$ minimizing Eq.* (7) *can be obtained by* maximizing

$$f_3([a^c_e]_{e\in\mathcal{E}}) := \sum_{(i,j)\in\mathcal{E}} \frac{a_{i,j} - a^c_{i,j}}{\sqrt{d_i - d^c_i} \cdot \sqrt{d_j - d^c_j}} \cdot \boldsymbol{h}_i \cdot \boldsymbol{h}_j^T$$

subject to the $\|.\|_0$ constraints and $0 \le a^c_e \le a_e$, where $d^c_i = \sum_{e\in\mathcal{E}_i} a^c_e$.

Proof. Similar to proof of Lemma 4.1; see appendix □

What is the crucial difference between Lemma 5.2 and Lemma 4.1/5.1? For the previous solutions, the objective function has decomposed in independent terms. That is, when adding an edge to $\boldsymbol{A}^c$, i.e. changing $a^c_{i,j}$ from 0 to $a_{i,j}$, the scores of the other edges are not affected. In Lemma 5.2, the sum in f_3 does *not* decompose into independent terms. In particular, the terms d^c_i in the denominator lead to a coupling of multiple edges.

While, in principle, f_3 can be optimized via projected gradient ascent, each gradient step would require to iterate through all edges. Therefore, as an alternative, we propose a more efficient greedy approximation: Similar to before, we focus on the solutions $\boldsymbol{v}^{\mathcal{X}}$. Starting with $\mathcal{X} = \emptyset$, we iteratively let this set grow following a steepest ascent strategy. That is, we add the edge e_{best} to $\mathcal{X}$ fulfilling

$$e_{best} = \arg\max_{e\in\mathcal{E}'} f_3(\boldsymbol{v}^{\mathcal{X}\cup\{e\}}) \quad (8)$$

where $\mathcal{E}'$ indicates the edges that could be added to $\mathcal{X}$ without violating the constraints. Naively computing Equation (8) requires $|\mathcal{E}'| \cdot |\mathcal{E}|$ many steps – and since we perform multiple iterations to let $\mathcal{X}$ grow, it results in a runtime complexity of $O(\theta \cdot |\mathcal{E}|^2)$; obviously not practical. In the following, we show how to compute this result more efficiently.

Definition 5.3. Let $\mathcal{X} \subseteq \mathcal{E}$, $d^{\mathcal{X}}_i := d_i - \sum_{e\in\mathcal{E}_i\cap\mathcal{X}} a_e$, and $p_{i,j} := a_{i,j} \cdot \boldsymbol{h}_i \cdot \boldsymbol{h}_j^T$. We define

$$s(i, w, \mathcal{X}) := \sum_{\substack{j \\ (i,j)\in\mathcal{E}_i\setminus\mathcal{X} \\ \vee(j,i)\in\mathcal{E}_i\setminus\mathcal{X}}} \left(\frac{1}{\sqrt{d^{\mathcal{X}}_i - w}\sqrt{d^{\mathcal{X}}_j}} - \frac{1}{\sqrt{d^{\mathcal{X}}_i}\sqrt{d^{\mathcal{X}}_j}} \right) p_{i,j}$$

for each node i, and

$$\delta(e, \mathcal{X}) := \left(\frac{1}{\sqrt{d^{\mathcal{X}}_i}\sqrt{d^{\mathcal{X}}_j}} - \frac{1}{\sqrt{d^{\mathcal{X}}_i - a_e}\sqrt{d^{\mathcal{X}}_j}} - \frac{1}{\sqrt{d^{\mathcal{X}}_i}\sqrt{d^{\mathcal{X}}_j - a_e}} \right) p_{i,j}$$

for each edge $e = (i, j)$, and

$$\Delta(e, \mathcal{X}) := s(i, a_e, \mathcal{X}) + s(j, a_e, \mathcal{X}) + \delta(e, \mathcal{X})$$

Corollary 5.4. *Given $\mathcal{X}$ and $\mathcal{E}' \subseteq \mathcal{E}\setminus\mathcal{X}$. It holds*

$$\arg\max_{e\in\mathcal{E}'} f_3(\boldsymbol{v}^{\mathcal{X}\cup\{e\}}) = \arg\max_{e\in\mathcal{E}'} \Delta(e, \mathcal{X})$$

Proof. See appendix. □

By exploiting Corollary 5.4, we can find the best edge according to Eq. (8), by only considering the terms $\Delta(e, \mathcal{X})$. This term can be interpreted as the gain in f_3 when adding the edge e to the set $\mathcal{X}$. After computing the scores $s(i, w, \mathcal{X})$ for each *node*, $\Delta(e, \mathcal{X})$ can be evaluated in constant time per *edge*.

Moreover, let $e = (i, j)$, for each non-incident edge $(i', j') = e' \in \mathcal{E}\setminus(\mathcal{E}_i \cup \mathcal{E}_j)$ it obviously holds $s(i', w, \mathcal{X}) = s(i', w, \mathcal{X} \cup \{e\})$ and $\delta(e', \mathcal{X}) = \delta(e', \mathcal{X} \cup \{e\})$. Thus, assume the edge $e_{best} = (i, j)$ has been identified and added to $\mathcal{X}$. For finding the *next* best edge, only the scores $s(i, ., .)$ and $s(j, ., .)$ need to be updated; followed by an evaluation of δ for all edges incident to the nodes i and j. *The remaining nodes and edges are not affected; their s, δ, and Δ values are unchanged.*

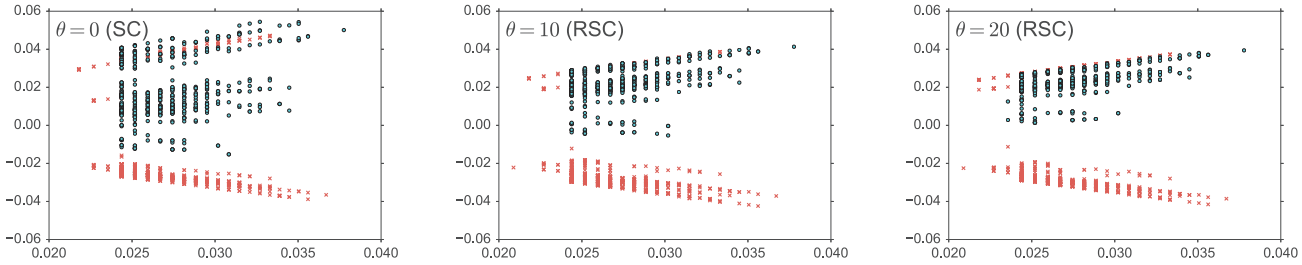

Figure 3: Spectral embedding of banknote data based on L_{sym}. Note that the dataset contains *two* clusters. Left: Standard spectral clustering; middle & right: Our method (θ = 10 and 20). The learned embeddings increase the discrimination between the points. The two clusters stand out more clearly.

Exploiting these results, we compute the set $\mathcal{X}$ similar to Algorithm 1 (lines 5 - 15): Initially, compute for each node i and unique edge weight $a_{i,j}$ the term $s(i, a_{i,j}, \mathcal{X})$. Then compute for each edge e the term $(\Delta(e, \mathcal{X}), e)$ and add it to the priority queue PQ. These steps can be done in time $O(\gamma \cdot |\mathcal{E}|)$, where γ is the number of unique edge weights per node. Within the while loop: Every time the best element $e_{best} = (i, j)$ from the PQ is retrieved, we recompute $s(i, ., \mathcal{X})$ and $s(j, ., \mathcal{X})$, followed by a recomputation of $\delta(e, \mathcal{X})$ for all incident edges. Noticing that there are at most $2 \cdot x$ many incident edges (x nearest-neighbor graph) these steps can be done in time $O(\gamma \cdot x + x \cdot log(|\mathcal{E}|))$.

Overall, this leads to a time complexity of $O(\gamma \cdot |\mathcal{E}| + \theta \cdot (x \cdot log(|\mathcal{E}|) + \gamma \cdot x))$. Note that the worst case (each edge has a unique weight) corresponds to $\gamma = x$. In this case we obtain $O(x \cdot |\mathcal{E}| + \theta \cdot (x \cdot log(|\mathcal{E}|) + x^2))$. For our case of spectral clustering using nearest-neighbor graphs, however, it holds $\gamma = 1$. In this case, we obtain an algorithm with complexity

$$O(|\mathcal{E}| + \theta \cdot x \cdot log(|\mathcal{E}|))$$

Thus, being linear in the number of edges.

In summary, the principle for solving Eq. (7) is almost identical to Algorithm 1 with the additional overhead of re-evaluating the term $\Delta(e, \mathcal{X})$ for the edges incident to e_{best}. The full pseudocode of this algorithm and the detailed complexity analysis are provided in the supplementary material for convenience.

6 EXPERIMENTS

Setup. We compare our method, called RSC, against spectral clustering (SC), and the two related works AHK [9] and NRSC [14]. We denote with RSC-L_{xy} the different variants of our method using the corresponding Laplacian. For all techniques, we set the number of clusters k equal to the number of clusters in the data. As default values we construct nearest neighbor graphs with 15 neighbors, allowing half of the edges to be removed per node ($m = 0.5 \cdot x$). While [14] uses a principle for automatically setting their parameters, the obtained results were often extremely low. Thus, we manually optimized their parameters to obtain better solutions. All experiments are averaged over several k-means runs to ensure stability. All used datasets are publicly available/on our website. Real world data: We use handwritten digits (pendigits; 7494 instances; 16 attributes; 10 clusters)[4], banknote authentication data (1372 inst.; 5 att.; 2 clus.)[4], iris (150 inst.; 4 att.; 3 clus.)[4], and USPS data (9298 inst.; 256 att.; 10 clus.)[5]. Further, we use two random subsamples of the MNIST data (10k/20k inst., 784 att., 10 clus.) because our competitors can not handle larger samples due to their cubic complexity. Synthetic data: Besides the well known moon data as shown in Fig. 1, where the vectors' positions are perturbed based on Gaussian noise using different variance, we also generate synthetic similarity graphs based on the planted partitions model [4]: Given the clusters, we randomly connect each node to x percent of the other nodes in its cluster. Additionally, we add a certain fraction of noise edges to the graph. By default we generate data with 1000 instances, x = 0.3 and 20 clusters. ***We evaluate the clustering quality of the different approaches using NMI (1=best).*** We start with an in-depth analysis of our technique followed by a comparison with competing techniques.

Spectral embedding. RSC optimizes the spectral embedding $\boldsymbol{H}$ by learning the matrix $\boldsymbol{A}^g$. Thus, we start by analyzing the spectral embeddings obtained by RSC. In Fig. 2 we illustrated the spectral embeddings for the data of Fig. 1 (right). Standard spectral clustering fails on this data, since the embedding (left plot in Fig. 2) leads to unclear groupings. In contrast, applying our technique, we obtain the embeddings as shown in Fig. 2 (right): the three clusters stand out; thus, perfect clustering structure can be obtained.

A similar behavior can be observed for real world data. Fig. 3 shows the spectral embedding of the banknote data (*two* clusters) regarding $\boldsymbol{L}_{sym}$ (the other Laplacians show similar results). On the left we see the original embedding: The points do not show a clear separation in two groups. In the middle and right plot, we applied RSC with θ=10 and θ=20, respectively. As shown, the separation between the points clearly increases. The embedding gets optimized leading to higher clustering accuracy. As we will see later, for the banknote data, the NMI score increases from 0.46 to 0.61.

Sparsity threshold. As indicated in Fig. 3, increasing the sparsity threshold might lead to a clearer separation. We now analyze this aspect in more detail. Figure 4 (left) analyzes a two-moons datasets with noise of 0.1. We vary θ for all three techniques. $\theta = 0$ corresponds to original spectral clustering using the corresponding Laplacian; clearly, its quality is low. As shown, for all techniques we observe an increase in the clustering quality until a stable point is reached. Fig. 4 (right) shows the same behavior for the banknote data. The removal of corrupted edges improves the clustering results. All three variants are able to reach the highest NMI of 0.61.

[4]https://archive.ics.uci.edu/ml/

[5]http://www.cs.nyu.edu/~roweis/data.html

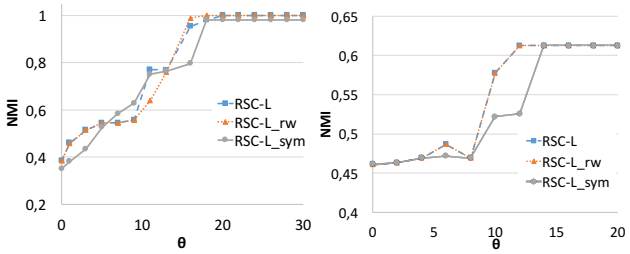

Figure 4: Increasing the sparsity threshold θ improves the clustering quality. Left: two moons data; right: banknote.

Remark: Using the variant based on $\boldsymbol{L}$, at some point, the quality will surely drop again. When all corrupted edges have been removed, one will start to remove 'good' edges. The reason is that the terms in Eq. (3) are always non-negative. In contrast, using $\boldsymbol{L}_{rw}/\boldsymbol{L}_{sym}$ (Eq. (3); Cor. 5.4, Δ), edges connecting points within the same cluster will often obtain negative scores. Those edges will *never* be included in the matrix $\boldsymbol{A}^c$ – independent of θ. Thus, in general, the later two versions are more robust regarding θ.

According to our definitions we aim to minimize the trace. In Fig. 5 we illustrate the value of the trace for the setting of Fig. 4 (right). Since the trace between the different Laplacian can not be meaningfully compared in absolute values, we plot it relative to the trace obtained by standard spectral clustering. For all of our approaches the trace can successfully be lowered by a significant amount; thus, confirming the effectiveness of our learning approach. Note also that our algorithms often need only around 10 iterations to converge to these good results.

Detection of corrupted edges. Next, we analyze how well our principles are able to spot corrupted edges. For this, we artificially added corrupted edges to the similarity graph based on the planted partition model. We used two different settings: in one case 10% of all edges in the graph are corrupted; in the other even 20% of all edges. Knowing the corrupted edges, we measure the precision $p = |B \cap A|/|B|$ and recall $r = |B \cap A|/|A|$, where A denotes the corrupted edges, and B the edges removed by our technique.

Fig. 6 shows the results when increasing the number of removed edges (i.e. θ). For the 10% noise case (left plot), we observe a very high precision which stays at the optimal value until 1200 – only the corrupted edges are removed. Note that the absolute number of corrupted edges in the data is 1261. Likewise, the recall is continuously increasing until around 0.96. Thus only a few corrupted edges could not be detected. The scenario with 20% noise (3605 corrupted edges) is more challenging. While $\boldsymbol{L}_{sym}$ obtains a result very close to optimal, $\boldsymbol{L}_{rw}$ and $\boldsymbol{L}$ perform slightly worse. Thus, for these techniques also some 'good' edges get removed. Note that the curves do not need to be monotonic. Due to the joint optimization, different edges can be removed for each parameter setting.

Overall, for realistic scenarios of noise, all techniques perform well – with $\boldsymbol{L}_{sym}$ often being the best one.

Robustness. In the next experiment, we analyze the robustness of our method regarding perturbed data. That is, we study how an increasing degree of noisy data effects the clustering quality. We refer to the established moon data and perturb it randomly according to Gaussian noise with variance increased from 0 to 0.115. To highlight the variation in the clustering quality we average the results over 10 datasets for each noise parameter.

Fig. 7 shows the results of our principles and standard spectral clustering. The lines represents the mean NMI, while the error bars represent the variance. Note that for standard SC we report the best result among all three Laplacian (for each dataset individually). *Thus, standard spectral clustering gets an additional strong benefit.* Clearly, spectral clustering is not robust and rapidly decreases in quality. Interestingly, for the moon data, $\boldsymbol{L}$ performs best. In any case, all of our approaches clearly outperform the baseline.

Comparison of Runtime. We now turn our attention to the comparison between RSC and related techniques. First, we briefly evaluate the runtime behaviour. The experiments were conducted on 2.9 GHz Intel Core i5 with 8GB of RAM running Matlab R2015a. Fig. 8 shows the overall runtime for each method on pendigits. To obtain larger data, we performed supersampling; adding small noise (variance of 0.1) to avoid duplicates. Confirming our complexity analysis, RSC scales linear in the number of edges – and it easily handles graphs with around 1 mio edges. Not surprisingly, standard spectral clustering is the fastest. The competing techniques are much slower due to their cubic complexity in the number of nodes; they can only handle small graphs. For the larger datasets, they did not finish within 24 hours.

Comparison of clustering quality. Next, we provide an overview of the clustering quality. For all techniques we used the symmetric normalized Laplacian since it performed best. Even though our main aim is to improve spectral clustering approaches, we additionally report the results of two famous clustering principles: (A) k-means and (B) density-based clustering (here: mean shift). For the later, we tuned the bandwidth parameter to obtain highest scores. As already mentioned in the set-up, the competing techniques' parameters were tuned as well. For RSC, we simply used

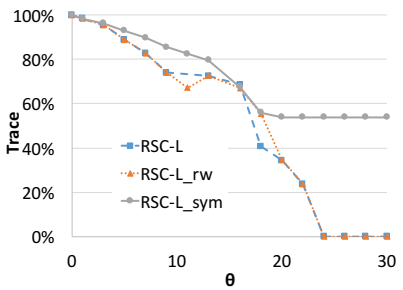

Figure 5: Our method obtains better (lower) trace values (trace of SC=100%)

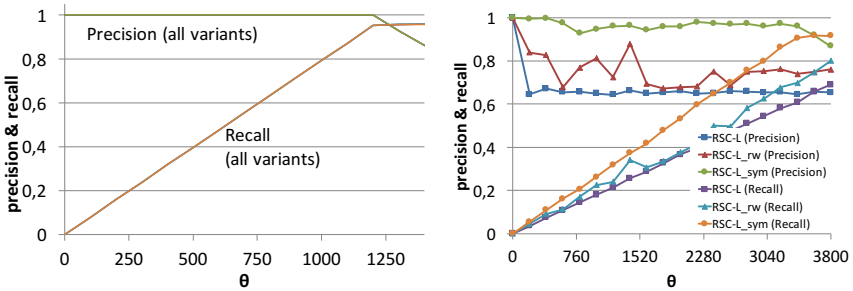

Figure 6: Our method achieves high precision and recall; the corrupted edges are successfully detected. Left: 10% noise; right: 20% noise.

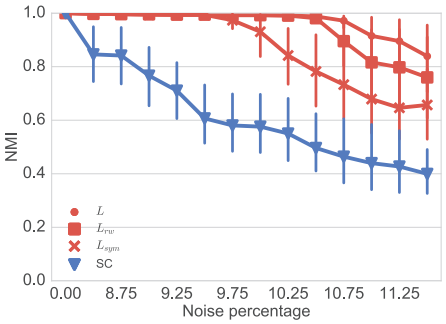

Figure 7: Robustness to noise. Our RSC clearly outperforms spectral clustering.

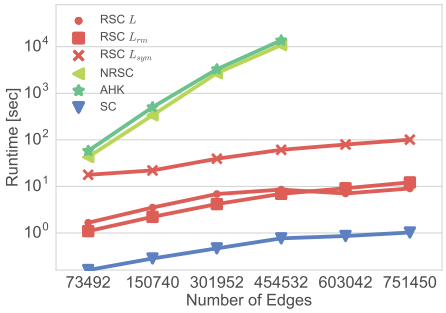

Figure 8: Runtime analysis. RSC scales linear in the number of edges

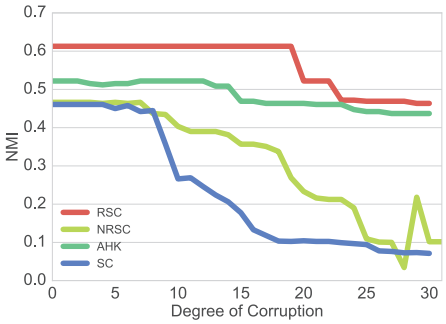

Figure 9: Robustness of all techniques on banknote. RSC is very stable.

a very large θ and let the method automatically decide how many edges to mark as corrupted – one advantage of $\boldsymbol{L}_{sym}$ (see remark in experiment on sparsity threshold). Table 1 summarizes the results for the different datasets.

Besides using the full datasets, we use the principle of [9, 14] and additionally select sets of the data. More precisely, from the pendigits data we select specific digits indicated with pendigits-xyz.

data	SC	NRSC	AHK	(A)	(B)	RSC
moons	0.47	0.99	0.53	0.19	0.34	**1.00**
banknote	0.46	0.47	0.52	0.03	0.03	**0.61**
USPS	0.78	0.83	0.77	0.61	0.15	**0.85**
MNIST-10 K	0.71	0.70	0.70	0.48	0.48	**0.73**
MNIST-20 K	0.70	0.76	0.71	0.48	d.n.f.	**0.78**
iris	0.78	0.79	0.53	0.76	0.72	**0.80**
pendigits	0.82	**0.83**	0.82	0.69	0.66	0.82
pendigits-16	0.86	0.87	0.88	0.88	0.01	**0.91**
pendigits-146	0.93	0.94	0.94	0.88	0.47	**0.96**

Table 1: Clustering quality

As shown, in many cases our technique outperforms the competing techniques. In some scenarios by even 15 percentage points w.r.t. spectral clustering. Though, it is also fair to mention that not for all datasets an improvement can be achieved. Our method clearly outperforms k-means and density based clustering and it finishes for all these datasets in a few seconds to minutes. In contrast, NRSC and AHK required already around one and three hours respectively on the larger MNIST data.

Comparison of robustness. Next, we analyze the robustness of the methods by artificially adding noise to the real data. To ensure that the cluster detection is indeed getting more difficult, we specifically add corruptions to the similarity graph connecting different clusters. The results for the banknote data are presented in Fig. 9. As shown, at the beginning all techniques remain at their quality level obtained on the original data, with RSC obtaining the highest quality. Adding more corruptions, however, standard spectral clustering drops very quickly and sharply to low quality. In contrast, RSC stays at its highest level for the longest time. AHK is quite stable as well, while NRSC is much more sensitive.

Comparison of the embeddings' quality. One of our main hypothesis is that jointly learning the embedding and the corruptions leads to improved embeddings. Thus, lastly, we study the quality of the embeddings learned by all techniques. While we have already seen in Table 1, that the NMI scores of RSC are good, such measures give only limited insights how the underlying embedding space looks like. *How can we measure the quality of the embeddings?* In particular, we aim to derive statistics that do *not* depend on applying a clustering technique on the data – instead we want to evaluate the embedding based on the ground truth classes[6] only. We argue that two properties should be fulfilled: (a) *Local purity*. In a good embedding, the instances within a local neighborhood should belong to the same class. (b) *Global separation*: In a good embedding, it should be possible to distinguish between instances of different classes by inspecting the intra-class and inter-class distances only. That is, the classes should be easily separable.

Evaluation of Local Purity. Let $\boldsymbol{h}_i$ denote the embedding of instance i and $c_i \in C$ its class according to the ground truth. We define the purity $pur_x(i)$ of the neighborhood around instance i (x-nearest neighbors) as the largest fraction of instances belonging to the same class. Formally: Let $NN_x(i)$ denote the set of x nearest neighbors of i in the embedding space and $occ_x(c,i)$ the number of times the class c occurs in the neighborhood of node i (including the node itself), i.e. $occ_x(c,i) = |\{j \in NN_x(i) \cup \{i\} \mid c_j = c\}|$. Then the purity is given by

$$pur_x(i) = \frac{1}{x+1} \max_{c \in C} occ_x(c,i)$$

The overall local purity (at scale x) is defined as the average over all instances:

$$PUR(x) = \frac{1}{N} \sum_{i=1}^{N} pur_x(i)$$

In the best case $PUR(x) = 1$, each neighborhood contains instances of a single class only; in the worst case $1/k$ where $k = |C|$ is the number of classes. As an example, imagine an embedding that looks like Figure 1: We would observe good purity ($PUR(x)$=1) for small x. Slightly increasing x, the purity decreases since different classes get merged, demonstrating the low quality of such an embedding.

Results: Fig. 10 shows the result for banknote and USPS for all competing techniques. x is scaled from 1 to the maximal cluster size to ensure that all scales are captured. As a baseline we also evaluate the purity of the original input data (i.e. the embedding space is the raw data). As seen, in both plots, the original data only has good purity for small x, but drops quickly. This indicates complex shapes like in Fig. 1. For banknote, RSC has consistently the highest local purity. The embedding well reflects the ground truth locally. SC and AHK perform slightly worse. NRSC, in contrast, drops very quickly similar to the baseline. For USPS, all techniques clearly outperform the baseline. While for small x RSC is slightly below

[6]We specifically use the term 'class' to indicate the groups given by the ground truth, not the groups detected by an arbitrary clustering method applied on the embedding.

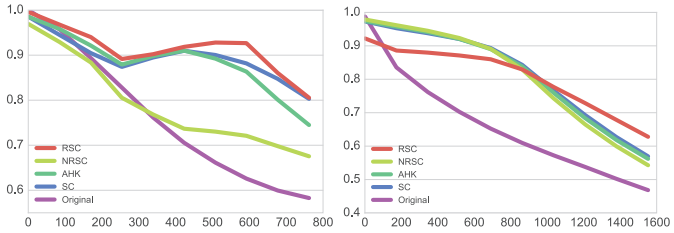

Figure 10: Evaluation of local purity (left: banknote, right: USPS). RSC's embedding represents the classes well locally.

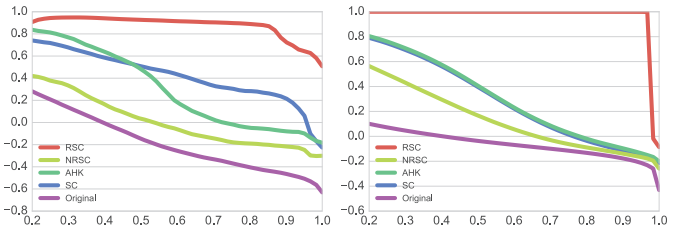

Figure 11: Evaluation of global separation (left: banknote, right: USPS). RSC's embedding separates the classes well.

the competitors, it outperforms them for larger x, which is also reflected by a higher NMI.

Evaluation of Global Separation. We formalize global separation by extending the idea of the Silhouette coefficient [17]. For each class c we compute the list of pairwise distances $P_{c,c}$ of all instances within the class, as well as the list of pairwise distances $P_{c,c'}$ between instances from class c to c', i.e.

$$P_{c,c'} = [dist(\boldsymbol{h}_i, \boldsymbol{h}_j)]_{i \in C_c, j \in C_{c'}}$$

where $C_c = \{i \mid c_i = c\}$ is the set of all instances from class c.

For each list we compute the average over the $(x \cdot 100)\%$ smallest elements, denoted as $P_{c,c'}(x)$. Following the Silhouette coefficient, we then compute the difference between the *within class* distances and the distance to the *closest other class*, i.e.

$$GS_c(x) = \frac{P_{c,c'}(x) - P_{c,c}(x)}{\max\{P_{c,c'}(x), P_{c,c}(x)\}}$$

where $c' = \arg\min_{c' \neq c} P_{c,c'}(x)$. In the best case $GS_c(x) = 1$, in the worst case -1. $GS_c(x)$ can intuitively be regarded as a robust extension of the Silhouette coefficient (w.r.t. the ground truth classes). For $x = 1$, it resembles the Silhouette coefficient w.r.t. class c. For $x < 1$ only parts of the distances are considered, thus, capturing that the embedding might not completely represent the ground truth. Imagine an embedding that resembles Fig. 1, $GS_c(x)$ will be relatively low due to similar inter-class and intra-class distances.

Results: Fig. 11 shows the result for two exemplary classes. An overview of all classes is available in the supp. material. Again, the raw data shows the worst result, with scores consistently below 0.25 indicating no good separation/clusteredness of the class labels in the space. In contrast, RSC obtains extremely high scores in both datasets up to a very high x: for banknote until 0.85, for USPS 0.97. That is, a very large fraction of the ground truth class is well separated and clustered in the learned embedding. Clearly, not every class from the data shows such perfect result since the NMI scores are 0.61 and 0.85. The (sharp) drops at the end indicate that some of the instances of the ground truth class are wrongly assigned to a different region in the embedding space. When trying to include these ($x = 1$), the score highly drops. The competing approaches consistently perform worse, showing no good match between the ground truth and the clusteredness of the embedding.

The results on the USPS data also indicate that *local purity* and *global separation* of an embedding are indeed two different properties: While SC and AHK have good results on the local purity, they perform poor regarding global separation. The learned embeddings of RSC capture well both properties confirming the benefit of our joint learning principle.

7 CONCLUSION

We proposed a spectral clustering technique for noisy data. Our core idea was to decompose the similarity graph into two latent factors: sparse corruptions and clean data. We jointly learned the spectral embedding as well as the corrupted data. We proposed three different algorithmic solutions using different Laplacians. Our experiments have shown that the learned embeddings clearly emphasize the clustering structure and that our method outperforms spectral clustering and state-of-the-art competitors.

Acknowledgments. This research was supported by the German Research Foundation, Emmy Noether grant GU 1409/2-1, and by the Technical University of Munich - Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement no 291763, co-funded by the European Union.

REFERENCES

[1] C. C. Aggarwal and C. K. Reddy. *Data clustering: algorithms and applications*. CRC Press, 2013.
[2] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *JACM*, 58(3):11, 2011.
[3] H. Chang and D.-Y. Yeung. Robust path-based spectral clustering. *Pattern Recognition*, 41(1):191–203, 2008.
[4] A. Condon and R. M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 2001.
[5] C. Davis and W. M. Kahan. The rotation of eigenvectors by a perturbation. iii. *SIAM SINUM*, 7(1):1–46, 1970.
[6] N. Günnemann, S. Günnemann, and C. Faloutsos. Robust multivariate autoregression for anomaly detection in dynamic product ratings. In *WWW*, pages 361–372, 2014.
[7] S. Günnemann, I. Farber, S. Raubach, and T. Seidl. Spectral subspace clustering for graphs with feature vectors. In *ICDM*, pages 231–240, 2013.
[8] S. Günnemann, N. Günnemann, and C. Faloutsos. Detecting anomalies in dynamic rating data. In *KDD*, pages 841–850, 2014.
[9] H. Huang, S. Yoo, H. Qin, and D. Yu. A robust clustering algorithm based on aggregated heat kernel mapping. In *ICDM*, pages 270–279, 2011.
[10] F. Jordan and F. Bach. Learning spectral clustering. *Adv. Neural Inf. Process. Syst*, 16:305–312, 2004.
[11] W. Kong, S. Hu, J. Zhang, and G. Dai. Robust and smart spectral clustering from normalized cut. *Neural Computing and Applications*, 23(5):1503–1512, 2013.
[12] D. Lehmann, L. I. Oćallaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):577–602, 2002.
[13] X. Li, W. Hu, C. Shen, A. Dick, and Z. Zhang. Context-aware hypergraph construction for robust spectral clustering. *TKDE*, 26(10):2588–2597, 2014.
[14] Z. Li, J. Liu, S. Chen, and X. Tang. Noise robust spectral clustering. In *ICCV*, pages 1–8, 2007.
[15] B. A. Miller, M. S. Beard, P. J. Wolfe, and N. T. Bliss. A spectral framework for anomalous subgraph detection. *Trans. on Signal Proc.*, 63(16):4191–4206, 2015.
[16] J. Pfeiffer and F. Rothlauf. Analysis of greedy heuristics and weight-coded eas for multidimensional knapsack problems and multi-unit combinatorial auctions. In *GECCO*, pages 1529–1529, 2007.
[17] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Computational and applied mathematics*, 20:53–65, 1987.
[18] P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*, volume 589. John Wiley, 2005.

[19] G. Stewart and J.-G. Sun. *Matrix Perturbation Theory*. Academic Press Boston, 1990.

[20] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[21] X. Wang, F. Nie, and H. Huang. Structured doubly stochastic matrix for graph based clustering. In *KDD*, pages 1245–1254, 2016.

[22] L. Wu, X. Ying, X. Wu, and Z. Zhou. Line orthogonality in adjacency eigenspace with application to community partition. In *IJCAI*, pages 2349–2354, 2011.

[23] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *NIPS*, pages 1601–1608, 2004.

[24] X. Zhu, C. Loy, and S. Gong. Constructing robust affinity graphs for spectral clustering. In *CVPR*, pages 1450–1457, 2014.

APPENDIX

Proof of Lemma 4.1. Note that $\boldsymbol{L}(\boldsymbol{A}^g) = \boldsymbol{D}(\boldsymbol{A}^g) - \boldsymbol{A}^g = (\boldsymbol{D}(\boldsymbol{A}) - \boldsymbol{D}(\boldsymbol{A}^c)) - (\boldsymbol{A} - \boldsymbol{A}^c) = \boldsymbol{L}(\boldsymbol{A}) - \boldsymbol{L}(\boldsymbol{A}^c)$. Thus, Eq. (2) can equivalently be written as $\text{Tr}(\boldsymbol{H}^T \cdot \boldsymbol{L}(\boldsymbol{A}^g) \cdot \boldsymbol{H}) = \text{Tr}(\boldsymbol{H}^T \cdot (\boldsymbol{L}(\boldsymbol{A}) - \boldsymbol{L}(\boldsymbol{A}^c)) \cdot \boldsymbol{H}) = \text{Tr}(\boldsymbol{H}^T \cdot \boldsymbol{L}(\boldsymbol{A}) \cdot \boldsymbol{H}) - \text{Tr}(\boldsymbol{H}^T \cdot \boldsymbol{L}(\boldsymbol{A}^c) \cdot \boldsymbol{H})$.

Given $\boldsymbol{H}$, the term $\text{Tr}(\boldsymbol{H}^T \cdot \boldsymbol{L}(\boldsymbol{A}) \cdot \boldsymbol{H})$ is constant. Thus, minimizing the previous term is equivalent to maximizing $\text{Tr}(\boldsymbol{H}^T \cdot \boldsymbol{L}(\boldsymbol{A}^c) \cdot \boldsymbol{H})$.

Let $\boldsymbol{y}_k$ be a column vector of $\boldsymbol{H}$. Noticing that (see [20]) $\boldsymbol{y}_k^T \boldsymbol{L}(\boldsymbol{A}^c)\boldsymbol{y}_k = \sum_{i,j} \frac{1}{2} \cdot a^c_{i,j} \cdot (y_{k,i} - y_{k,j})^2$, and exploiting the orthogonality of $\boldsymbol{H}$ it follows: $\text{Tr}(\boldsymbol{H}^T \cdot \boldsymbol{L}(\boldsymbol{A}^c) \cdot \boldsymbol{H}) = \sum_k \sum_{i,j} \frac{1}{2} \cdot a^c_{i,j} \cdot (y_{k,i} - y_{k,j})^2 = \sum_{i,j} \frac{1}{2} \cdot a^c_{i,j} \cdot \left\|\boldsymbol{h}_i - \boldsymbol{h}_j\right\|_2^2$, where the last step used $y_{k,i} = h_{i,k}$.

To ensure that $\boldsymbol{A}^c$ as well as $\boldsymbol{A}^g$ are non-negative, it holds $0 \leq a^c_{i,j} \leq a_{i,j}$. Thus, if $a_{i,j} = 0$ then $a^c_{i,j} = 0$. Exploiting this fact and the symmetry of the graph leads to $\sum_{i,j} \frac{1}{2} \cdot a^c_{i,j} \cdot \left\|\boldsymbol{h}_i - \boldsymbol{h}_j\right\|_2^2 = \sum_{(i,j)\in\mathcal{E}} a^c_{i,j} \cdot \left\|\boldsymbol{h}_i - \boldsymbol{h}_j\right\|_2^2$.

Next, we show that there exists a solution where each $a^c_{i,j} \in \{0, a_{i,j}\}$. As known, $0 \leq a^c_{i,j} \leq a_{i,j}$. Let $M = [a^c_e]_{e\in\mathcal{E}}$ be a maximum of Eq. (3) where some $a^c_{i,j} > 0$ but $< a_{i,j}$. Let M' be the solution where this entry is replaced by $a^c_{i,j} = a_{i,j}$. Since only $\|.\|_0$ constraints are used, M and M' fulfill the same constraints. Since $\left\|\boldsymbol{h}_i - \boldsymbol{h}_j\right\|_2^2$ is non-negative, $f_1(M') \geq f_1(M)$. It follows, that a solution minimizing Eq. (2) can be found by investigating $a^c_{i,j} = 0$ or $a^c_{i,j} = a_{i,j}$ only. □

Proof of Lemma 5.1. The goal is to find a matrix $\boldsymbol{A}^g$ whose sum of the first k eigenvalues is minimal (and fulfills the given constraints). Since, however, $\boldsymbol{A}^g$ is not known, we refer to the principle of eigenvalue perturbation.

Let $\boldsymbol{A}^t$ be the matrix obtained in the previous iteration of the alternating optimization and let $\boldsymbol{y}_i$ be the i-th generalized eigenvector of $\boldsymbol{L}(\boldsymbol{A}^t)$ (these are the *columns* of the matrix $\boldsymbol{H}$ from above, i.e. $y_{i,j} = h_{j,i}$). Furthermore, denote the corresponding eigenvalues with λ_i. We define $\boldsymbol{L}(\boldsymbol{A}^g) - \boldsymbol{L}(\boldsymbol{A}^t) =: \Delta\boldsymbol{L}$ and $\boldsymbol{D}(\boldsymbol{A}^g) - \boldsymbol{D}(\boldsymbol{A}^t) = \Delta\boldsymbol{D}$.

Based on the theory of eigenvalue perturbation [19], the eigenvalue λ^g_i of $\boldsymbol{L}(\boldsymbol{A}^g)$ can be approximated by

$$\lambda^g_i \approx \lambda_i + \boldsymbol{y}_i^T \cdot (\Delta\boldsymbol{L} - \lambda_i \cdot \Delta\boldsymbol{D}) \cdot \boldsymbol{y}_i$$

$$= \lambda_i + \boldsymbol{y}_i^T \cdot ((\boldsymbol{L}(\boldsymbol{A}^g) - \boldsymbol{L}(\boldsymbol{A}^t)) - \lambda_i \cdot (\boldsymbol{D}(\boldsymbol{A}^g) - \boldsymbol{D}(\boldsymbol{A}^t))) \cdot \boldsymbol{y}_i$$

Using the fact that $\boldsymbol{L}(\boldsymbol{A}^g) = \boldsymbol{L}(\boldsymbol{A}) - \boldsymbol{L}(\boldsymbol{A}^c)$ and $\boldsymbol{D}(\boldsymbol{A}^g) = \boldsymbol{D}(\boldsymbol{A}) - \boldsymbol{D}(\boldsymbol{A}^c)$, and after rearranging the terms, we obtain

$$\lambda^g_i \approx \overbrace{\lambda_i + \boldsymbol{y}_i^T \cdot ((\boldsymbol{L}(\boldsymbol{A}) - \boldsymbol{L}(\boldsymbol{A}^t)) - \lambda_i \cdot (\boldsymbol{D}(\boldsymbol{A}) - \boldsymbol{D}(\boldsymbol{A}^t))) \cdot \boldsymbol{y}_i}^{=:c_i} - \underbrace{\boldsymbol{y}_i^T \cdot ((\boldsymbol{L}(\boldsymbol{A}^c)) - \lambda_i \cdot (\boldsymbol{D}(\boldsymbol{A}^c)) \cdot \boldsymbol{y}_i}_{=:g_i}$$

Since c_i is constant, minimizing λ^g_i is equivalent to maximizing g_i. Simplifying yields:

$$g_i = \boldsymbol{y}_i^T \cdot \boldsymbol{L}(\boldsymbol{A}^c) \cdot \boldsymbol{y}_i - \lambda_i \cdot \boldsymbol{y}_i^T \cdot \boldsymbol{D}(\boldsymbol{A}^c) \cdot \boldsymbol{y}_i$$

$$= \sum_{j,j'} \frac{1}{2} a^c_{j,j'}(y_{i,j} - y_{i,j'})^2 - \lambda_i \sum_j y^2_{i,j} \cdot d^c_j$$

where $d^c_j = [\boldsymbol{D}(\boldsymbol{A}^c)]_{j,j} = \sum_{j'} a^c_{j,j'}$. Thus

$$g_i = \sum_{j,j'} \frac{1}{2} a^c_{j,j'}(y_{i,j} - y_{i,j'})^2 - \lambda_i y^2_{i,j} a^c_{j,j'}$$

$$= \sum_{j,j'} a^c_{j,j'} \left(\frac{1}{2}(y_{i,j} - y_{i,j'})^2 - \lambda_i y^2_{i,j}\right)$$

and exploiting the symmetry of the graph, we obtain

$$g_i = \sum_{(j,j')\in\mathcal{E}} a^c_{j,j'} \left((y_{i,j} - y_{i,j'})^2 - \lambda_i y^2_{i,j} - \lambda_i y^2_{i,j'}\right)$$

Since the overall goal is to minimize $\sum_{i=1}^k \lambda^g_i$, we aim at maximizing

$$\sum_{i=1}^k g_i = \sum_{i=1}^k \sum_{(j,j')\in\mathcal{E}} a^c_{j,j'} \left((y_{i,j} - y_{i,j'})^2 - \lambda_i y^2_{i,j} - \lambda_i y^2_{i,j'}\right)$$

$$= \sum_{(j,j')\in\mathcal{E}} a^c_{j,j'} \left(\sum_{i=1}^k (y_{i,j} - y_{i,j'})^2 - \sum_{i=1}^k \lambda_i y^2_{i,j} - \sum_{i=1}^k \lambda_i y^2_{i,j'}\right)$$

By noticing that $y_{i,j} = h_{j,i}$ we obtain

$$= \sum_{(j,j')\in\mathcal{E}} a^c_{j,j'} \left(\underbrace{\left\|\boldsymbol{h}_j - \boldsymbol{h}_{j'}\right\|_2^2 - \left\|\sqrt{\boldsymbol{\lambda}} \circ \boldsymbol{h}_j\right\|_2^2 - \left\|\sqrt{\boldsymbol{\lambda}} \circ \boldsymbol{h}_{j'}\right\|_2^2}_{x}\right)$$

Note that some of the terms x might be negative. Clearly, since we aim to maximize the equation – and since $a^c_{i,j} \geq 0$ – for these terms we have to choose $a^c_{i,j} = 0$. For the remaining (non-negative) terms, the same arguments apply as in the proof of Lemma 4.1: i.e. they are either 0 or $a_{i,j}$. Thus, overall, for each term we have $a^c_e \in \{0, a_e\}$. □

Proof of Lemma 5.2. Note that $a^g_{i,j} = a_{i,j} - a^c_{i,j}$ and $d^g_i = d_i - d^c_i$. Let $\boldsymbol{y}_k$ be a column vector of $\boldsymbol{H}$. It holds $\boldsymbol{y}_k^T \cdot \boldsymbol{L}_{sym}(\boldsymbol{A}^g)\boldsymbol{y}_k \overset{[20]}{=} \sum_{i,j} \frac{1}{2} a^g_{i,j}(\frac{y_{k,i}}{\sqrt{d^g_i}} - \frac{y_{k,j}}{\sqrt{d^g_j}})^2 = \sum_{i,j} \frac{1}{2} a^g_{i,j}(\frac{y^2_{k,i}}{d^g_i} + \frac{y^2_{k,j}}{d^g_j} - \frac{2 \cdot y_{k,i} y_{k,j}}{\sqrt{d^g_i}\sqrt{d^g_j}}) = \sum_i \frac{1}{2} y^2_{k,i} + \sum_j \frac{1}{2} y^2_{k,j} - \sum_{i,j} \frac{a^g_{i,j} y_{k,i} y_{k,j}}{\sqrt{d^g_i}\sqrt{d^g_j}}$. Since $\boldsymbol{y}_k$ is given, the first two terms are constant. Furthermore, due to orthogonality it holds $Tr(\boldsymbol{H}^T \boldsymbol{L}_{sym} \boldsymbol{H}) = \sum_k \boldsymbol{y}_k^T \cdot \boldsymbol{L}_{sym} \boldsymbol{y}_k$. Thus, minimizing the trace is equivalent to maximizing $\sum_k \sum_{i,j} \frac{a^g_{i,j} y_{k,i} y_{k,j}}{\sqrt{d^g_i}\sqrt{d^g_j}} = \sum_{i,j} \frac{a^g_{i,j}}{\sqrt{d^g_i}\sqrt{d^g_j}} \boldsymbol{h}_i \cdot \boldsymbol{h}_j^T$, noticing that $y_{k,i} = h_{i,j}$. Exploiting the graph's symmetry concludes the proof. □

Proof of Corollary 5.4. Adding $e = (i, j)$ to $\mathcal{X}$ has the following effects: the term a^c_e changes from 0 to a_e; the degree of the two incident nodes becomes $d^{\mathcal{X}\cup\{e\}}_i = d^{\mathcal{X}}_i - a_e$. Therefore,

$$f_3(\boldsymbol{\nu}^{\mathcal{X}\cup\{e\}}) = f_3(\boldsymbol{\nu}^{\mathcal{X}}) - \frac{p_e}{\sqrt{d^{\mathcal{X}}_i} \cdot \sqrt{d^{\mathcal{X}}_j}} - \sum_{\substack{(x,y)\in(\mathcal{E}_i\cup\mathcal{E}_j)\setminus\mathcal{X} \\ (x,y)\neq(i,j)}} \frac{p_{x,y}}{\sqrt{d^{\mathcal{X}}_x} \cdot \sqrt{d^{\mathcal{X}}_x}}$$

$$+ \sum_{\substack{x\neq j \\ (i,x)\in\mathcal{E}_i\setminus\mathcal{X} \\ \vee(x,i)\in\mathcal{E}_i\setminus\mathcal{X}}} \frac{p_{i,x}}{\sqrt{d^{\mathcal{X}}_i - a_e}\sqrt{d^{\mathcal{X}}_x}} + \sum_{\substack{x\neq i \\ (j,x)\in\mathcal{E}_j\setminus\mathcal{X} \\ \vee(x,j)\in\mathcal{E}_j\setminus\mathcal{X}}} \frac{p_{x,j}}{\sqrt{d^{\mathcal{X}}_j - a_e}\sqrt{d^{\mathcal{X}}_x}}$$

$$= f_3(\boldsymbol{\nu}^{\mathcal{X}}) + s(i, a_e, \mathcal{X}) + s(j, a_e, \mathcal{X}) + \delta(e, \mathcal{X}) = f_3(\boldsymbol{\nu}^{\mathcal{X}}) + \Delta(e, \mathcal{X})$$

Since $\mathcal{X}$ is given, $f_3(\boldsymbol{\nu}^{\mathcal{X}})$ is constant. Thus, the edge $e \in \mathcal{E}'$ maximizing $f_3(\boldsymbol{\nu}^{\mathcal{X}\cup\{e\}})$ is found by maximizing $\Delta(e, \mathcal{X})$. □