A Sparse Low-Rank Biclique Decomposition for Graphs

Antoine Vialle*
LTCI, Télécom Paris, IP Paris
Palaiseau, France
antoine.vialle@telecom-paris.fr

Sergei Gerasimov
LTCI, Télécom Paris, IP Paris
Palaiseau, France
sergei.gerasimov@telecom-paris.fr

Aref Einizade
SAMOVAR, Télécom SudParis, IP Paris
Palaiseau, France
aref.einizade@telecom-sudparis.eu

Antonio Ortega
University of Southern California
Los Angeles, USA
aortega@usc.edu

Fragkiskos D. Malliaros
Université Paris Saclay, CentraleSupélec, Inria
Gif-Sur-Yvette, France
fragkiskos.malliaros@centralesupelec.fr

Jhony H. Giraldo*
LTCI, Télécom Paris, IP Paris
Palaiseau, France
jhony.giraldo@telecom-paris.fr

Abstract

Low-rank plus sparse decompositions, such as robust PCA, typically model the low-rank component as dense and the sparse component as corruption. This viewpoint is poorly suited to graph adjacency matrices, where sparsity is structural and dense low-rank factors destroy the combinatorial and computational properties of the graph. We propose signed biclique decomposition (SBD), a sparse low-rank decomposition for integer-weighted graphs that represents the structured component as a sum of sparse signed binary rank-one factors, together with a sparse signed residual. The resulting representation remains discrete, interpretable, sparse, and compatible with standard linear algebra. We formulate fixed-rank and greedy SBD objectives and derive a gain-based greedy algorithm with theoretical guarantees. Empirically, SBD recovers block structure in noisy graphs, while its factorized representation enables efficient sparse matrix-matrix multiplication (SpMM). On real graphs with strong shared-neighborhood structure, SBD achieves compression factors above 10 and SpMM speedups up to $6\times$ over cuSPARSE, suggesting a practical alternative to dense low-rank decompositions for graphs. Code is available at github.com/IAntoineV/sbd.

1 Introduction

Low-rank decomposition is a fundamental tool for uncovering latent structure in high-dimensional data. In sparse plus low-rank (SLR) models [30] (see Table 4 for abbreviations), a data matrix $\mathbf{A}$ is decomposed as $\mathbf{A} = \mathbf{L}_0 + \mathbf{R}_0$, where $\mathbf{L}_0$ is a low-rank matrix that captures global or latent structure, and $\mathbf{R}_0$ is a sparse matrix capturing localized deviations, outliers, or corruptions. We refer to $\mathbf{R}_0$ as the *residual* (or remainder) of the decomposition. A prominent example of SLR is robust principal component analysis (RPCA) [8], which recovers these two components through a convex optimisation problem. RPCA and related SLR methods have been successful in many applications, including medical imaging [7, 9], asset pricing [3], background subtraction, and corruption removal [8].

*Corresponding authors: Antoine Vialle and Jhony H. Giraldo.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

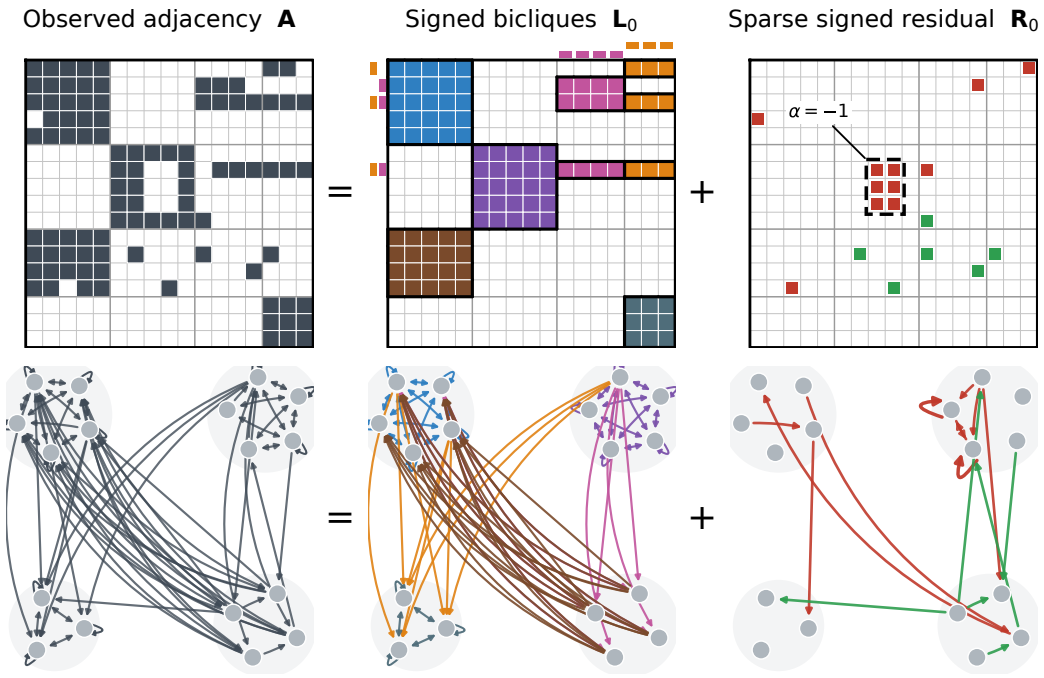


Figure 1: A sparse graph adjacency matrix $\mathbf{A}$ is decomposed into a sparse low-rank block component $\mathbf{L}_0$ and a sparse signed residual $\mathbf{R}_0$. Green entries denote added edges, while red entries denote missing edges relative to the recovered block structure.

In this paper, we ask what an SLR-like decomposition should look like for sparse graph adjacency matrices. Since sparsity is intrinsic to graph structure, both components should remain sparse: the structured term should capture shared connectivity patterns through graph-supported objects, such as clique-, biclique-, or shared-neighborhood blocks, while the residual should store sparse signed corrections. Classical SLR methods such as RPCA are poorly matched to this goal: their low-rank component is typically dense and incoherent, and sparsity is treated as corruption [8]. Applied to graphs, this can introduce dense fill-in, destroy the adjacency sparsity pattern, and produce representations that are inefficient to store (see Appendix C.4).

A relevant alternative is sparse PCA (SPCA) [35, 19, 29], which enforces sparsity on the low-rank factors rather than directly on the reconstructed matrix or residual. However, the resulting singular vectors are generally not binary or indicator-like, and thus the induced low-rank approximation is poorly aligned with the discrete structure of graphs. As a consequence, the residual component typically remains comparably dense, offering limited improvement in sparsity over the original adjacency matrix $\mathbf{A}$ (see Appendix C.4).

These limitations suggest a different sparse-low-rank principle for graphs: the structured component should be both low-rank and sparse, while the residual should remain sparse. This leads to a sparse plus low-rank-and-sparse (SLRS) formulation, in which $\mathbf{L}_0$ is represented by a small number of sparse graph-supported low-rank components and $\mathbf{R}_0$ stores the remaining signed corrections. Figure 1 illustrates this goal: a sparse graph adjacency matrix is decomposed into a sparse low-rank block component and a sparse signed residual. This is the formulation pursued in this work.

In this paper, we propose sparse low-rank decompositions for integer-weighted graphs from a combinatorial perspective. We establish a connection between RPCA-style decompositions and sparse decompositions of graphs by introducing a signed biclique decomposition (SBD) framework. SBD represents a graph using a residual and a small number of sparse, signed, binary components, which can be interpreted as discrete shared-neighborhood or biclique-like patterns. This result provides a theoretical foundation for fast and scalable SLRS models on graphs and shows that low-rank graph structure can be approximated by discrete combinatorial objects rather than dense

real-valued factors. Building on this connection, we propose a simple greedy algorithm that extracts such components efficiently and scales to large graphs.

SBD is motivated by the need for graph representations that are simultaneously sparse, interpretable, and computationally efficient. Sparser representations are desirable in graph settings as they preserve locality and reduce redundancy in the underlying structure. Empirically, we show that this formulation matches the reconstruction behavior of RPCA-style methods on small graphs while remaining scalable to large-scale settings. Moreover, the resulting decomposition enables significant computational gains, in particular, by accelerating sparse matrix-matrix multiplication (SpMM).

Our main contributions are:

- We formulate graph sparse low-rank decomposition as a signed biclique decomposition of an integer-weighted adjacency matrix. Each component is a sparse signed binary rank-one matrix, so the resulting low-rank term remains combinatorial and sparse rather than dense.
- We derive a greedy objective for signed bicliques. We prove that positive-gain steps decrease the residual norm and give conditions for alternating optimisation to find them. We bound the remainder using the stopping rule.
- We introduce scalable SBD implementations for medium and large graphs. Because the decomposition is expressed in standard linear algebra, it can be used directly to accelerate SpMM.
- We evaluate the method on graph denoising and SpMM benchmarks, where the sparse signed-biclique representation yields speedups over standard sparse kernels on graphs with strong shared-neighborhood structure.

2 Related Work

Several lines of work are closely related to our sparse low-rank decomposition framework.

Sparse low-rank decompositions for graphs and matrices. The most closely related works study matrices that are simultaneously sparse and low-rank. Savalle et al. [31] consider the estimation of matrices that satisfy both constraints, motivating sparse low-rank structure beyond the classical RPCA setting. Closer to our setting, Kanada et al. [20] propose a low-rank sparse decomposition of graph adjacency matrices for extracting clean clusters. These works share our motivation that graph structure should not be represented by a dense low-rank matrix. Like RPCA, these methods use singular-value thresholding for regularized nuclear-norm minimization, which is difficult to scale to large graphs and does not provide the discrete signed-biclique representation considered here.

Binary and Boolean matrix decompositions. A related line of work decomposes binary matrices under algebraic structures, such as the Boolean semiring or arithmetic over $\mathbb{Z}/2\mathbb{Z}$ [26, 10, 5, 17]. These models suit compression and pattern discovery, but they use notions of rank that differ from standard linear algebra: Boolean addition satisfies $1 + 1 = 1$, while arithmetic over $\mathbb{Z}/2\mathbb{Z}$ satisfies $1 + 1 = 0$. Consequently, they do not directly support linear operations such as multiplying an adjacency matrix by a dense feature matrix. Binary matrix factorization methods use ordinary arithmetic [22, 34, 33, 4], this typically restricts the model to unsigned bicliques, making it closer to edge partitioning than to a signed low-rank-plus-residual decomposition. Other methods, like Miller et al. [27], learn binary latent features on nodes to predict edge probabilities of the graph.

Cut, biclique, and spectral graph decompositions. Cut and biclique decompositions have also been studied in graph approximation and regularity theory. The Frieze–Kannan weak regularity lemma [11, 12] approximates a weighted adjacency matrix by a sum of weighted cut matrices, equivalently biclique-like blocks, with a residual controlled in cut norm. This provides an important connection between matrix approximation and graph structure. However, regularity-based decompositions are not designed to preserve sparsity.

There is also a classical connection between biclique decompositions and spectral graph structure. For example, the minimum number of bicliques required to partition the edges of a graph is lower bounded by $\max(n_+(\mathbf{A}), n_-(\mathbf{A}))$, where $n_+(\mathbf{A})$ and $n_-(\mathbf{A})$ denote the numbers of positive and negative eigenvalues of the adjacency matrix [16, 15]. This shows that the biclique structure is closely tied to the spectrum of the graph. Related work by Ames and Vavasis [2] studies recovery of planted cliques and bicliques via nuclear norm minimization, further connecting combinatorial graph patterns with low-rank structure. These results motivate the use of biclique-like components.

Graph compression and summarization. Graph compression and summarization methods aim to represent graphs using compact structures, often through lossless or near-lossless encodings [24, 6]. While these methods are effective for reducing storage or identifying dense patterns, they are not designed to provide low-rank plus sparse decompositions compatible with linear algebraic operations. Those are much more aligned with binary matrix decomposition.

Accelerating SpMM on graphs. Several optimized GPU kernels have been proposed to accelerate SpMM on graphs, including Sputnik [13], dgSparse, and GE-SpMM [18]. More recently, compressed binary matrices (CBM) [1] proposed compressing graph adjacency operators using ordered neighborhood differences and cumulative sums to accelerate SpMM.

Overall, prior work either provides sparse low-rank formulations without a scalable discrete signed representation, binary decompositions under nonstandard algebraic operations, or graph compression methods that are not designed as linear low-rank-plus-residual models. Our contribution is to combine these desiderata: we decompose an integer-valued adjacency matrix in standard linear space into sparse signed biclique components and a sparse signed residual. The signed components allow cancellations between overlapping structures, while the binary supports preserve a combinatorial graph interpretation and enable efficient storage and computation.

3 Preliminaries

Notation. We write vectors and matrices in boldface. For an integer $n \geq 1$, let $[n] := \{1, \ldots, n\}$. For a matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$, $\|\mathbf{M}\|_F$ denotes the Frobenius norm, $\|\mathbf{M}\|_*$ the nuclear norm, and $\|\mathbf{M}\|_1$the entrywise ℓ_1 norm. We denote by $\mathrm{rank}(\mathbf{M})$ the rank of $\mathbf{M}$ and by $\|\mathbf{M}\|_0$ its number of nonzero entries. Given a set $S \subseteq [n]$, we write $\mathbf{1}_S \in \{0,1\}^n$ for its indicator vector and $|S|$ its cardinality. For an optimisation problem over a vector $\mathbf{x}$, we denote an optimal solution by $\mathbf{x}^\star$. Throughout the paper, for a given integer-weighted directed graph, n denotes the number of nodes, E denotes set of edges and $\mathbf{A} \in \mathbb{Z}^{n \times n}$ denotes an integer-weighted adjacency matrix. We write decompositions of the form

$$\mathbf{A} = \mathbf{L}_0 + \mathbf{R}_0, \tag{1}$$

where $\mathbf{L}_0$ is the structured component and $\mathbf{R}_0$ is the residual. We use the notation $\mathbf{L}_0$ for the low-rank component in order to avoid confusion with the graph Laplacian.

Sparse plus low-rank decompositions. Sparse plus low-rank decompositions model an observed matrix in the form (1), where the two components satisfy $\mathrm{rank}(\mathbf{L}_0) \ll n$ and $\|\mathbf{R}_0\|_0 \ll n^2$.

The low-rank component is intended to capture global or latent structure, whereas the sparse component captures localized deviations, corruptions, or outliers. Classical convex relaxations replace the rank by the nuclear norm and the number of nonzeros by the entrywise ℓ_1 norm.

Robust principal component analysis. RPCA is a canonical sparse plus low-rank model. Given an observed matrix $\mathbf{A}$, RPCA seeks a decomposition (1) by solving the convex optimisation problem

$$\min_{\mathbf{L}_0, \mathbf{R}_0} \|\mathbf{L}_0\|_* + \lambda \|\mathbf{R}_0\|_1 \quad \text{subject to} \quad \mathbf{A} = \mathbf{L}_0 + \mathbf{R}_0, \tag{2}$$

where $\lambda > 0$ controls the low-rank–sparsity tradeoff [8]. Under suitable incoherence and sparsity assumptions, RPCA can exactly recover low-rank matrices with sparse corruptions. However, these assumptions are poorly matched to sparse graph adjacency matrices (see Appendix C.4). This motivates decompositions in which the structured component is not only low-rank but also sparse.

Sparse principal component analysis. SPCA modifies PCA by imposing sparsity on the principal directions. A typical rank-K formulation estimates sparse vectors $\mathbf{V}_1, \ldots, \mathbf{V}_K$ through

$$\min_{\mathbf{U}, \mathbf{V}} \|\mathbf{A} - \mathbf{U}\mathbf{V}^\top\|_F^2 + \lambda \sum_{k=1}^{K} \|\mathbf{V}_k\|_1, \qquad \text{subject to suitable normalization constraints.} \tag{3}$$

Different variants use different normalizations, deflation schemes, and sparsity penalties [19, 35]. Although SPCA promotes sparse low-rank factors, it is not directly adapted to adjacency matrices since the factors are real-valued rather than discrete vertex indicators, and factor sparsity does not ensure that the residual $\mathbf{R}_0$ remains sparse. Thus, SPCA does not, by itself, provide the sparse, combinatorial, and storage-efficient graph decomposition required here.

4 Signed Biclique Decomposition

We introduce SBD, a discrete low-rank model that represents an integer-weighted directed adjacency matrix as a sum of signed biclique blocks plus a sparse residual.

4.1 Model

Let $\mathbf{A} \in \mathbb{Z}^{n\times n}$ be a directed integer-weighted adjacency matrix. For each component t, let $S_t, T_t \subseteq [n]$ denote source and target node sets, and let $\alpha_t \in \{-1, +1\}$ denote its sign. We write $\mathbf{w}_t = \mathbf{1}_{S_t}$, $\mathbf{q}_t = \mathbf{1}_{T_t}$, where $\mathbf{w}_t, \mathbf{q}_t \in \{0,1\}^n$ are binary indicator vectors. A signed biclique is the rank-one integer matrix $\alpha_t \mathbf{w}_t \mathbf{q}_t^\top$. Given K components, SBD writes

$$\mathbf{A} = \mathbf{L}_0^{(K)} + \mathbf{R}_0^{(K)} = \sum_{t=0}^{K-1} \alpha_t \mathbf{w}_t \mathbf{q}_t^\top + \mathbf{R}_0^{(K)}. \tag{4}$$

Since each biclique is rank one, $\mathrm{rank}(\mathbf{L}_0^{(K)}) \le K$. The decomposition has two goals. First, the biclique component $\mathbf{L}_0^{(K)}$ should capture the dominant structure of $\mathbf{A}$, leaving a small residual $\mathbf{R}_0^{(K)}$, *i.e.*, $\|\mathbf{R}_0^{(K)}\|_F^2 \ll \|\mathbf{A}\|_F^2$. Second, the representation should be compact. We encode each component by its supports and its sign, giving a storage cost of $|S_t| + |T_t| + 1$. The residual is stored sparsely. This motivates the cost $\sum_{t=0}^{K-1}(|S_t| + |T_t| + 1) + \|\mathbf{R}_0^{(K)}\|_0$, and the corresponding compression factor

$$\mathrm{CF} = \frac{\|\mathbf{A}\|_0}{\sum_{t=0}^{K-1}(|S_t| + |T_t| + 1) + \|\mathbf{R}_0^{(K)}\|_0}. \tag{5}$$

Thus, larger values of CF correspond to more compact representations of the input graph and directly reduce the number of arithmetic operations required for sparse matrix computations.

4.2 Fixed-rank SBD objective

For a fixed number of components K, we search for an SBD as the following discrete regularized optimisation problem.

Fixed-rank signed biclique approximation. Given $\lambda \ge 0$ and $K \in \mathbb{N}$, solve

$$\min_{\substack{\mathbf{w}_t,\mathbf{q}_t\in\{0,1\}^n \\ \alpha_t\in\{-1,+1\}}} \left\| \mathbf{A} - \sum_{t=0}^{K-1} \alpha_t \mathbf{w}_t \mathbf{q}_t^\top \right\|_F^2 + \lambda \sum_{t=0}^{K-1} \left(\|\mathbf{w}_t\|_2^2 + \|\mathbf{q}_t\|_2^2 \right). \tag{6}$$

The first term controls reconstruction error:

$$\left\| \mathbf{A} - \mathbf{L}_0^{(K)} \right\|_F^2 = \|\mathbf{R}_0^{(K)}\|_F^2.$$

The second term controls the length of the biclique component. Indeed, since $\mathbf{w}_t$ and $\mathbf{q}_t$ are binary, $\|\mathbf{w}_t\|_2^2 = |S_t|$ and $\|\mathbf{q}_t\|_2^2 = |T_t|$. Thus, λ trades off reconstruction accuracy against biclique size. Small values of λ favor aggressive reconstruction, while larger values favor smaller and denser components (see Appendix C.2). The Frobenius loss in (6) also promotes sparse residuals. Since $\mathbf{A}$ and all signed bicliques are integer-valued, the residual $\mathbf{R}_0^{(K)}$ is integer-valued. Therefore, $\|\mathbf{R}_0^{(K)}\|_0 \le \|\mathbf{R}_0^{(K)}\|_F^2$ and minimizing the squared residual hence upper-bounds the number of residual entries that must be stored explicitly.

4.3 Greedy SBD objective

The problem in (6) is a combinatorial optimisation problem over K coupled bicliques. We therefore use a greedy approximation that iteratively extracts one signed biclique at a time, in the spirit of matching pursuit [25], but with data-dependent combinatorial atoms. Let $\mathbf{R}_0^{(0)} = \mathbf{A}$. At iteration t, the next component is chosen by solving the rank-one problem

$$(\alpha_t^\star, \mathbf{w}_t^\star, \mathbf{q}_t^\star) = \operatorname*{arg\,min}_{\substack{\mathbf{w},\mathbf{q}\in\{0,1\}^n \\ \alpha\in\{-1,+1\}}} \left\| \mathbf{R}_0^{(t)} - \alpha \mathbf{w}\mathbf{q}^\top \right\|_F^2 + \lambda \left(\|\mathbf{w}\|_2^2 + \|\mathbf{q}\|_2^2 \right). \tag{7}$$

The residual is then updated as $\mathbf{R}_0^{(t+1)} = \mathbf{R}_0^{(t)} - \alpha_t^\star \mathbf{w}_t^\star (\mathbf{q}_t^\star)^\top$. An exact solver stops when no nonempty biclique improves the objective. Our finite seed search (Section 4.5) stops when its search budget is exhausted or no returned candidate has positive gain. This provides guarantees on the properties of the remainder obtained using either the exact solver or our heuristic methodology (Theorem 1).

4.4 Guarantees

The following result establishes the theoretical guarantees of greedy minimizing in (7). Full proof is deferred to Appendix A.1.

Definition 1. *We call gain the following function:*

$$G_\lambda(\alpha, S, T, \mathbf{R}_0) = 2\alpha \sum_{(i,j)\in S\times T} \mathbf{R}_0(i,j) - |S||T| - \lambda(|S| + |T|). \tag{8}$$

Note that minimizing (7) is equivalent to maximizing G_λ.

Theorem 1. *Let $\lambda \geq 0$. Start with $\mathbf{R}_0^{(0)} = \mathbf{A}$. At each accepted step update $\mathbf{R}_0^{(t+1)} = \mathbf{R}_0^{(t)} - \alpha_t \mathbf{1}_{S_t} \mathbf{1}_{T_t}^\top$, where $G_t = G_\lambda(\alpha_t, S_t, T_t, \mathbf{R}_0^{(t)}) > 0$. Then:*

1. ***Residual decrease.*** *Each step satisfies*

$$\|\mathbf{R}_0^{(t+1)}\|_F^2 = \|\mathbf{R}_0^{(t)}\|_F^2 - G_t - \lambda(|S_t| + |T_t|). \tag{9}$$

   There are at most $K \leq \|\mathbf{A}\|_F^2$ steps. If $\lambda > 0$, also $K \leq \lfloor \|\mathbf{A}\|_F^2/(2\lambda) \rfloor$.

2. ***Constraints at stopping.*** *Suppose that stopping certifies $G_\lambda \leq 0$ for every candidate in a set $\mathcal{C}_K$, evaluated on the final residual. For each $(\alpha, S, T) \in \mathcal{C}_K$,*

$$\alpha d_{\mathbf{R}_0^{(K)}}(S,T) \leq \frac{1}{2} + \frac{\lambda}{2}\left(\frac{1}{|S|} + \frac{1}{|T|}\right), \tag{10}$$

   where $d_R(S,T) = \frac{\sum_{i\in S, j\in T} R(i,j)}{|S||T|}$. If both signs were checked for the same supports, this also bounds $|d_{\mathbf{R}_0^{(K)}}(S,T)|$.

3. ***Compression bound.*** *Define $J_t = \|\mathbf{R}_0^{(t)}\|_F^2 + \lambda \sum_{\ell<t}(|S_\ell| + |T_\ell|)$. $J_{t+1} = J_t - G_t$ and J_t is non increasing. For $\lambda \geq 1$,*

$$\mathrm{CF} \geq \frac{\|\mathbf{A}\|_0}{J_K + K}. \tag{11}$$

   Also, $J_t + t$ is non increasing for $\lambda \in \mathbb{N}^*$. It shows that our update increases a surrogate lower bound on CF at each step.

Corollary 1. *If the stopping certificate includes every singleton support with both signs, then $|\mathbf{R}_0^{(K)}(i,j)| \leq \lfloor (1+2\lambda)/2 \rfloor$ for all i, j. Thus $\mathbf{R}_0^{(K)} = 0$ for $\lambda < 1/2$, and its entries have magnitude at most one for $\lambda \in [1/2, 3/2)$.*

A sequential search that checks positive gain on the exact current residual before acceptance satisfies the descent assumptions of Theorem 1. This includes both SBDGreedy and SBDFast. Neither method needs an exact rank-one solver for this descent guarantee.

The stopping constraints apply only to candidates checked on the final residual. A global rank-one oracle checks all candidates and implies Corollary 1. A finite seed search does not provide that global certificate. Appendix B gives additional theoretical guarantees. Appendix C.3 compares our search with a rank-one solver on small graphs.

4.5 Algorithms

SBDGreedy uses alternating optimisation (AO) to maximise (8). For a fixed sign and row support, columns contribute independently. With a density floor $d_{\min}$, the update is

$$T \leftarrow \{j \in V : 2\alpha \sum_{i\in S} \mathbf{R}_0(i,j) > \max(|S| + \lambda, 2d_{\min}|S|)\}.$$

The row update is symmetric. The density floor limits holes in the selected block. Appendix B.2 derives the updates and local candidate sets. SBDGreedy checks the full gain before each sequential extraction, so Theorem 1 applies. Its stopping certificate covers only the candidates checked.

SBDFast searches seed batches in parallel and applies proposals sequentially. Before each acceptance, it recomputes the gain on the current exact residual, including updates from earlier proposals in the batch. The search first proposes positive blocks, then searches both signs. See Appendix D.1 for more details.

4.6 Accelerating SpMM with SBD

Standard truncated SVD (k-SVD) accelerates dense matrix multiplication by replacing a full matrix with dense low-rank factors, but this is poorly suited to graph SpMM: the dense singular vectors destroy the sparsity structure that makes SpMM efficient. In contrast, SBD accelerates SpMM while preserving sparsity and remaining lossless. After k decomposition steps, defining $\mathbf{w} = (\mathbf{w}_0, \ldots, \mathbf{w}_{k-1}), \mathbf{\Lambda} = \mathrm{diag}(\alpha_0, \ldots, \alpha_{k-1}), \mathbf{q} = (\mathbf{q}_0, \ldots, \mathbf{q}_{k-1})$, we obtain

$$\begin{aligned}\mathbf{A} &= \sum_{t=0}^{k-1} \alpha_t \mathbf{w}_t \mathbf{q}_t^\top + \mathbf{R}_0^{(k)} \\ &= \mathbf{w}\mathbf{\Lambda}\mathbf{q}^\top + \mathbf{R}_0^{(k)}.\end{aligned}$$

This factorization yields a sparser representation of the propagation operator, thereby reducing the number of multiplications in SpMM. Computationally, it replaces a single SpMM by two sequential sparse multiplications involving thinner factors. As a result, the method reduces arithmetic and communication costs at the expense of decreased parallelism. In practice, the decomposition can be implemented efficiently in CUDA, where the reduced message volume helps amortize kernel-launch overhead.

Algorithm 1: SBDGreedy: signed biclique extraction

Require: $\mathbf{A}$, λ, $d_{\min}$, seed budget M per sign, AO cap J, extraction cap $K_{\max}$

$\mathbf{R}_0 \leftarrow \mathbf{A}$
repeat
  For each sign α, take up to M rows ranked by $\sum_j \max(\alpha \mathbf{R}_0(i,j), 0)$
  for all signed seeds (u, α) **in parallel do**
    Form local candidate sets (U, V) around u; set $S \leftarrow \{u\}, T \leftarrow V$
    Alternate for at most J iterations
    Stop AO if both supports are unchanged or either is empty
  end for
  Select the nonempty candidate $(\alpha^\star, S^\star, T^\star)$ with largest full gain
  if no candidate has positive gain **then**
    stop
  end if
  $\mathbf{R}_0 \leftarrow \mathbf{R}_0 - \alpha^\star \mathbf{1}_{S^\star} \mathbf{1}_{T^\star}^\top$
until at most $K_{\max}$ blocks have been extracted

5 Experiments

We evaluate SBD along three axes. First, we test whether the signed biclique model can recover sparse low-rank graph structure in a controlled denoising setting. Second, we compare the scalability of SBD against existing sparse low-rank matrix decomposition methods on real graphs (see Appendix D.6). Third, we evaluate whether the learned representation yields practical speedups for SpMM, the core linear-algebraic primitive in many graph representation learning methods like graph neural networks [21].

5.1 Experimental setup

Graph denoising. We follow the benchmark of Kanada et al. [20]. The clean graph has ten cliques of 20 nodes ($p_{\text{in}} = 1$). We add inter-community edges with probability $q \in \{0.05, 0.15, 0.25, 0.35\}$. We threshold $\mathbf{L}_0$ at 0.5 and measure F_1 against the clean graph, excluding the diagonal. We compare SBDGreedy with RPCA [8], sparse SVD, Kanada et al. [20], and Savalle et al. [31], solved by ADMM. Each method uses 30 Optuna trials on five tuning graphs per noise level. Evaluation uses ten separate graphs. Appendix D.2 gives all ranges and fixed settings.

Decomposition scalability. We measure decomposition time on real graphs with fixed settings. SBDGreedy and SBDFast run on CPU using eight threads, while the continuous baselines run on an RTX 3090 GPU. These settings do not match reconstruction quality. For compression, we select the SBDFast decomposition with the highest CF from 30 Optuna trials per graph.

SpMM acceleration. We reuse the selected decomposition across feature widths and GPUs. We compare with cuSPARSE, Sputnik [13], dgSPARSE, GE-SpMM [18], and CBM [1]. SBD_base uses our kernel without bicliques ($\mathbf{L}_0 = 0$ and $\mathbf{R}_0 = \mathbf{A}$). All kernels are measured on RTX 3090, A40, and L40S, with ten warm-up and 50 timed launches; we report median latency. We use float32 features with $F \in \{32, 128, 256\}$ columns. Multiplication uses $\mathbf{L}_0 + \mathbf{R}_0 = \mathbf{A}$, retaining the exact remainder. Appendix D.3 describes the kernel and Appendix D.2 gives the timing protocol.

5.2 Denoising sparse low-rank graph structure

Table 1 reports denoising performance across corruption levels. SBD exactly recovers the clean graph on the evaluation seeds in the low-noise regimes $q \leq 0.15$ and remains competitive as the inter-community noise increases. At $q = 0.25$, SBD substantially improves over the noisy baseline and Savalle et al., and remains close to the best-performing methods. At the highest noise level, SBD and Kanada et al. both score 0.992, while SBD outperforms RPCA and sparse SVD. These results indicate that SBD captures the intended sparse block structure while using discrete biclique factors rather than dense or continuous low-rank components.

Table 1: SBM denoising F_1 (↑), $K = 10$, $N = 200$, $p_{\text{in}} = 1$. Means over ten evaluation graphs.

Method	0.05	0.15	0.25	0.35
Noisy input	0.808	0.585	0.457	0.375
Savalle	1.000	0.989	0.869	0.744
RPCA	1.000	1.000	0.999	0.707
Sparse SVD	1.000	0.999	0.970	0.749
Kanada	1.000	1.000	1.000	0.992
SBDGreedy (ours)	1.000	1.000	0.999	0.992

5.3 Scalability of the decomposition

Table 2 compares decomposition runtimes under a 60 second budget. Kanada et al. and Savalle reach the time limit on Cora, CiteSeer and ca-HepPh, and run out of memory on ca-AstroPh. Savalle uses the same symmetry-based ADMM solver as in denoising. RPCA succeeds on Cora and CiteSeer, while its dense intermediate matrices scale as $\mathcal{O}(n^2)$ in memory. Sparse SVD is computationally efficient, but its continuous factors are not binary indicators and therefore do not provide the signed biclique representation needed for our SpMM kernel; see Appendix C.4. In contrast, SBDFast completes all five graphs on CPU, from 0.022 seconds on Cora to 12.3 seconds on COLLAB, and SBDGreedy remains practical on the smaller instances. Continuous baselines are not evaluated on COLLAB because its dense input exceeds the memory limit.

5.4 Representation quality

Table 3 reports the size of the SBDFast representation on larger real graphs. We distinguish between the compression factor CF, which counts support memberships, signs and residual entries, and the memory compression ratio CR, which compares representation bytes with CSR. CBM uses a graph compression format based on ordered neighborhood differences. SBD optimizes a different objective: it factorizes shared neighborhoods into signed biclique components while preserving compatibility with ordinary linear algebra. Thus, the relevant question is not only whether SBD compresses the

Table 2: Runtime in seconds under a 60-second compute limit. OOT: time limit; OOM: allocation failure or dense input exceeding the 24 GiB budget. SBD uses eight CPU threads; continuous baselines use RTX 3090.

Method	Cora	CiteSeer	ca-HepPh	ca-AstroPh	COLLAB
SBDGreedy	5.99	5.59	OOT	OOT	OOT
SBDFast	0.022	0.026	0.35	0.60	12.3
Kanada	OOT	OOT	OOT	OOM	OOM
RPCA	4.36	7.10	OOT	OOT	OOM
Sparse SVD	6.30	2.08	55.0	OOT	OOM
Savalle	OOT	OOT	OOT	OOM	OOM

Table 3: SBDFast decomposition quality using the highest-CF trial per graph. CF counts stored entries; CR counts bytes relative to CSR. SBD uses its kernel arrays; CBM uses the storage model defined by its authors.

Dataset	N	$\|\mathbf{A}\|_0$	K	CF	$\mathrm{CR}_{\mathrm{SBD}}$	$\mathrm{CR}_{\mathrm{CBM}}$
Cora	2,708	10,556	181	1.05	0.99	1.06
PubMed	19,717	88,648	1,651	1.08	1.04	1.05
ca-HepPh	12,008	237,010	2,437	3.23	3.43	2.75
ca-AstroPh	18,772	396,160	10,519	2.15	2.70	1.73
coPapersDBLP	540,486	30,491,458	86,855	8.57	10.3	5.98
coPapersCiteseer	434,102	32,073,440	84,862	11.3	14.4	9.91
COLLAB	372,474	24,572,158	53,646	10.9	12.2	11.0

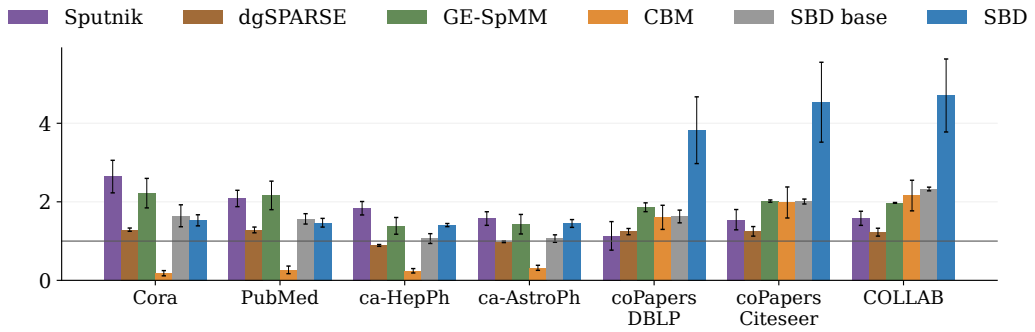


Figure 2: SpMM speedup over cuSPARSE, averaged over $F \in \{32, 128, 256\}$ within each GPU and then over RTX 3090, A40 and L40S. Error bars show variation across GPU averages. SBD_base uses no bicliques. Per-width results are in Appendix C.5.

graph, but whether the resulting operator representation accelerates SpMM. Appendix D.2.4 gives both storage models.

The results show two regimes. On graphs with limited shared-neighborhood structure, such as Cora and PubMed, SBD is close to break-even in CF and does not provide substantial memory compression. Their byte ratios are 0.99 and 1.04. On graphs with stronger shared-neighborhood structure, such as coPapersDBLP, coPapersCiteseer and COLLAB, SBD obtains compression factors of 8.57, 11.3 and 10.9. Under the stated byte models, CBM compresses slightly better on Cora and PubMed, while SBD compresses better on the other five graphs.

5.5 SpMM acceleration

Figure 2 reports SpMM speedups relative to cuSPARSE. Averaged across the three GPUs, SBD is faster than cuSPARSE on every tested graph at each feature width, with the largest gains on high-CF datasets. On COLLAB, SBD reaches $6.37\times$ speedup at $F = 32$ and remains above $3.8\times$ at $F = 128$ and $F = 256$. On coPapersCiteseer, SBD obtains $5.80\times$ speedup at $F = 32$ and above $3.8\times$ at larger feature dimensions. These results show that the signed biclique representation can reduce repeated computation while preserving the exact SpMM operator.

Figure 2 includes CBM alongside the GPU SpMM kernels. Compared with CBM, SBD is faster on every dataset and feature dimension in the GPU averages of Table 8. CBM benefits from compression on the largest community graphs, but is slower than cuSPARSE on the smaller graphs. SBD is also faster than Sputnik, dgSPARSE and GE-SpMM on the three largest graphs. However, Sputnik is faster on Cora, PubMed and ca-HepPh at every tested width.

SBD_base uses the same kernel without bicliques. Comparing it with SBD isolates the effect of the decomposition. On COLLAB at $F = 32$, it gives $1.75\times$ speedup compared with SBD's $6.37\times$, whereas it is faster than SBD on PubMed. The two-pass biclique computation amortizes column aggregation across rows sharing the same target set, but its overhead outweighs these benefits on weakly compressible graphs. Additional kernel-level details are provided in Appendix D.3.

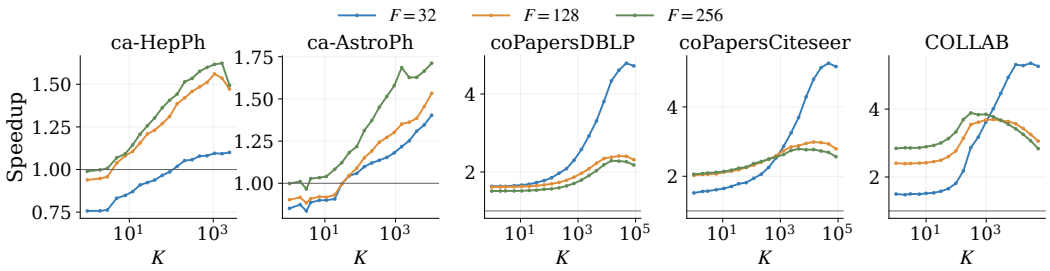


Figure 3: SpMM speedup over PyTorch sparse multiplication vs. number of bicliques K used on L40S, for $F \in \{32, 128, 256\}$. Bicliques are added in decreasing area order and the exact residual is rebuilt for each prefix.

5.6 When does SBD help?

The largest speedups in Figure 2 occur on the graphs with the highest compression factors in Table 3. The observed speedup is smaller than CF on these graphs because SBD replaces a single CSR SpMM by a two-pass computation involving the biclique factors and the residual. CF counts entries rather than GPU costs, so it does not predict latency directly. Graphs such as Cora and PubMed provide little compression, and the additional two-pass overhead can dominate the benefit of the blocks.

Figure 3 studies the effect of using only the top K bicliques, ordered by decreasing area. Increasing K moves more of the graph from the sparse residual into the structured two-pass path, but the benefit depends on the feature dimension. At $F = 32$, speedup continues to improve over a large range of K: COLLAB reaches its best sampled speed using about 56% of the available bicliques, and coPapersCiteseer behaves similarly. At $F = 128$ and $F = 256$, these graphs reach their best sampled speed earlier. On COLLAB, the corresponding fractions are about 3.2% and 0.6%.

This is not a universal cutoff: ca-AstroPh reaches its best sampled speed at the full decomposition for all three widths. Thus, the decomposition is most beneficial when the graph contains many reusable shared-neighborhood blocks, and using all blocks need not give the fastest multiplication. Appendix C.6 discusses the feature-width dependence. Both experiments use PyTorch sparse CSR multiplication with CUDA cuSPARSE backend as the reference.

5.7 Limitations

Our approach relies on structural redundancy in the graph for effective representation. Graphs with few shared-neighborhood structures require more biclique components and reduce compression efficiency. Signed binary factors may be less effective for weighted graphs (varying edge weights can require more components). Finally, our decomposition does not always lead to faster SpMM, as the overhead may outweigh its benefits on graphs with limited redundancy.

6 Conclusion

We introduced signed biclique decomposition (SBD), a sparse plus low-rank-and-sparse decomposition for integer-weighted graphs. Unlike classical low-rank methods such as RPCA, SBD represents the structured component as a sum of sparse signed binary rank-one factors, preserving the discrete, combinatorial, and computational properties of the graph. We formulated fixed-rank and greedy SBD objectives with theoretical guarantees. Empirically, SBD recovers latent block structure in noisy graphs competitively with RPCA-based methods, while remaining scalable to large graphs. On graphs with strong shared-neighborhood structure, SBD achieves compression factors above 10 and SpMM speedups up to $6\times$ over cuSPARSE, demonstrating that sparse combinatorial decompositions can serve as a practical and interpretable alternative to dense low-rank representations for graph data.

Acknowledgment

The authors declare no competing interests. The authors acknowledge the French National Research Agency (ANR) for the JCJC project DeSNAP ANR-24-CE23-1895-01. This work is also supported by Hi! PARIS, ANR/France 2030 program ANR-23-IACL-0005, ANR-23-CMAS-0033, and ANR-23-RHUS-0002 (Innov4-ePiK).

References

[1] Joao Alves, Samir Moustafa, Siegfried Benkner, Alexandre P. Francisco, Wilfried Gansterer, and Luís M. S. Russo. Accelerating graph neural networks using a novel computation-friendly matrix compression format. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2025.

[2] Brendan P. W. Ames and Stephen A. Vavasis. Nuclear norm minimization for the planted clique and biclique problems. *Mathematical Programming (MP)*, 2011.

[3] Sumanta Basu, Xianqi Li, and George Michailidis. Low rank and structured modeling of high-dimensional vector autoregressions. *IEEE Transactions on Signal Processing (TSP)*, 2019.

[4] Melanie Beckerleg and Andrew Thompson. A divide-and-conquer algorithm for binary matrix completion. *Linear Algebra and its Applications (LAA)*, 2020.

[5] Radim Belohlavek and Vilem Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *Journal of Computer and System Sciences (JCSS)*, 2010.

[6] Maciej Besta and Torsten Hoefler. Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations. *arXiv*, 2018.

[7] Adrienne E. Campbell-Washburn, David Atkinson, Zoltan Nagy, Rachel W. Chan, Oliver Josephs, Mark F. Lythgoe, Roger J. Ordidge, and David L. Thomas. Using the robust principal component analysis algorithm to remove rf spike artifacts from mr images. *Magnetic Resonance in Medicine (MRM)*, 2016.

[8] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 2011.

[9] Junbo Chen, Shouyin Liu, and Min Huang. Low-rank and sparse decomposition model for accelerating dynamic mri reconstruction. *Journal of Healthcare Engineering (JHE)*, 2017.

[10] Chen Dan, Kristoffer Arnsfelt Hansen, He Jiang, Liwei Wang, and Yuchen Zhou. Low Rank Approximation of Binary Matrices: Column Subset Selection and Generalizations. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Leibniz International Proceedings in Informatics (LIPIcs), 2018.

[11] Alan Frieze and Ravindran Kannan. A simple algorithm for constructing szemerédi's regularity partition. *The Electronic Journal of Combinatorics (E-JC)*, 1999.

[12] Alan Frieze and Ravindran Kannan. Quick approximation to matrices and applications. *Combinatorica*, 1999.

[13] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse GPU kernels for deep learning. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

[14] Robert Geisberger, Peter Sanders, and Dominik Schultes. Better approximation of betweenness centrality. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2008.

[15] R. L. Graham and H. O. Pollak. On the addressing problem for loop switching. *The Bell System Technical Journal (BSTJ)*, 1971.

[16] David A. Gregory, Valerie L. Watts, and Bryan L. Shader. Biclique decompositions and hermitian rank. *Linear Algebra and its Applications (LAA)*, 1999.

[17] Oktay Günlük, Raphael Andreas Hauser, and Réka Ágnes Kovács. Binary matrix factorization and completion via integer programming. *Mathematics of Operations Research (MOR)*, 2024.

[18] Guyue Huang, Guohao Dai, Yu Wang, and Huazhong Yang. Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

[19] Ian Jolliffe, Nickolay Trendafilov, and Mudassir Uddin. A modified principal component technique based on the lasso. *Journal of Computational and Graphical Statistics (JCGS)*, 2003.

[20] Taiju Kanada, Masaki Onuki, and Yuichi Tanaka. Low-rank sparse decomposition of graph adjacency matrices for extracting clean clusters. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2018.

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[22] Mehmet Koyutürk and Ananth Grama. Proximus: a framework for analyzing very high dimensional discrete-attributed datasets. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.

[23] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2007.

[24] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *ACM Computing Surveys (CSUR)*, 2018.

[25] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing (TSP)*, 1993.

[26] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2008.

[27] Kurt Miller, Michael Jordan, and Thomas Griffiths. Nonparametric latent feature models for link prediction. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[28] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML Workshop Graph Representation Learning and Beyond*, 2020.

[29] Ankit Parekh and Ivan W. Selesnick. Improved sparse low-rank matrix estimation. *Signal Processing*, 2017.

[30] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 2010.

[31] Pierre-André Savalle, Emile Richard, and Nicolas Vayatis. Estimation of simultaneously sparse and low rank matrices. In *International Conference on Machine Learning (ICML)*, 2012.

[32] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 2008.

[33] Bao-Hong Shen, Shuiwang Ji, and Jieping Ye. Mining discrete patterns via binary matrix factorization. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

[34] Zhongyuan Zhang, Tao Li, Chris Ding, and Xiangsun Zhang. Binary matrix factorization with applications. In *IEEE International Conference on Data Mining (ICDM)*, 2007.

[35] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics (JCGS)*, 2006.

Appendix

Table of Contents

Table of Abbreviations **14**

A Proofs **15**
A.1 Proof of Theorem 1 15

B Theoretical analysis **16**
B.1 Additional Analysis on SBD Objective 16
B.2 Derivation of the AO closed-form update 17
B.3 Positive-gain certificates 18
B.4 Exact-recovery guarantees 19
B.5 Bounds from empty updates 19

C Additional analysis **20**
C.1 Local positive-gain structure 20
C.2 Impact of the Regularization Parameter λ 20
C.3 Comparison with a rank-one solver 22
C.4 Why SVD-based Methods are Mismatched to our Objective 22
C.5 Full kernel comparison 24
C.6 Feature width and retained blocks 25

D Implementation details **26**
D.1 SBDFast implementation (binary matrices) 26
D.2 Experimental protocol and hyperparameters 26
D.3 CUDA Kernel Design 28
D.4 SpMM baselines 29
D.5 Computational Resources 29
D.6 Datasets 29

Table of Abbreviations

To assist the reader, Table 4 provides a summary of the main abbreviations and acronyms used throughout this paper.

Table 4: Summary of abbreviations and acronyms used in this paper.

Abbreviation	**Description**
SLR	Sparse plus Low-Rank (matrix decomposition model)
SLRS	Sparse plus Low-Rank-and-Sparse (our graph-adapted model)
RPCA	Robust Principal Component Analysis
SPCA	Sparse Principal Component Analysis
SBD	Signed Biclique Decomposition (proposed framework)
SpMM	Sparse Matrix-Matrix Multiplication ($\mathbf{A} \times \mathbf{X}$)
CF	Compression Factor ($\|\mathbf{A}\|_0/\mathrm{repr_cost}$)
CR	Memory Compression Ratio ($\mathrm{MB(CSR)}/\mathrm{MB(format)}$)
AO	Alternating Optimization
ILP	Integer Linear Programming (exact combinatorial solver)
CSR	Compressed Sparse Row format
SBM	Stochastic Block Model

A Proofs

A.1 Proof of Theorem 1

Proof of Theorem 1. **(1) Residual decrease and finite termination.** Write $R = \mathbf{R}_0^{(t)}$ and $B = \alpha_t \mathbf{1}_{S_t} \mathbf{1}_{T_t}^\top$. Expanding the squared norm gives

$$\|R - B\|_F^2 = \|R\|_F^2 - 2\alpha_t \sum_{i \in S_t, j \in T_t} R(i,j) + |S_t||T_t|.$$

Substituting the gain gives (9). Each accepted step strictly decreases the squared norm. This norm is a nonnegative integer, so there are at most $\|\mathbf{A}\|_F^2$ steps. For $\lambda > 0$, nonempty supports give $|S_t| + |T_t| \geq 2$. Summing the decreases over $K > 0$ accepted steps yields

$$0 \leq \|\mathbf{R}_0^{(K)}\|_F^2 < \|\mathbf{A}\|_F^2 - 2\lambda K. \tag{12}$$

Hence $K \leq \lfloor \|\mathbf{A}\|_F^2/(2\lambda) \rfloor$. The bound also holds for $K = 0$.

(2) Constraints at stopping. Let $R = \mathbf{R}_0^{(K)}$. A nonempty candidate $(\alpha, S, T) \in \mathcal{C}_K$ satisfies

$$2\alpha \sum_{i \in S, j \in T} R(i,j) - |S||T| - \lambda(|S| + |T|) \leq 0.$$

Dividing by $2|S||T|$ gives (10). Checking both signs on the same supports bounds $|d_R(S,T)|$. The bound covers only candidates checked on this final residual.

(3) Compression factor. Every nonzero integer has squared magnitude at least one. Thus

$$\|\mathbf{R}_0^{(K)}\|_0 \leq \|\mathbf{R}_0^{(K)}\|_F^2. \tag{13}$$

For $\lambda \geq 1$, the representation cost satisfies

$$\begin{aligned} C_K &= \|\mathbf{R}_0^{(K)}\|_0 + \sum_{t<K} (|S_t| + |T_t|) + K \\ &\leq \|\mathbf{R}_0^{(K)}\|_F^2 + \lambda \sum_{t<K} (|S_t| + |T_t|) + K = J_K + K. \end{aligned}$$

Since $\mathbf{A} \neq 0$, dividing $\|\mathbf{A}\|_0$ by these costs gives (11). Equation (9) also gives $J_{t+1} = J_t - G_t$. Hence

$$(J_{t+1} + t + 1) - (J_t + t) = 1 - G_t.$$

For positive integer λ, an accepted gain is an integer with $G_t \geq 1$. The right-hand side is nonpositive, so the lower bound on CF does not decrease. □

Proof of Corollary 1. The certificate checks both signs on every singleton. Choose the sign of a nonzero $R(i,j)$, where $R = \mathbf{R}_0^{(K)}$. Its gain satisfies

$$2|R(i,j)| - 1 - 2\lambda \leq 0.$$

Since the residual is integer-valued, $|R(i,j)| \leq \lfloor (1 + 2\lambda)/2 \rfloor$. Zero entries also satisfy this bound. It is zero for $\lambda < 1/2$ and one for $\lambda \in [1/2, 3/2)$. □

B Theoretical analysis

B.1 Additional Analysis on SBD Objective

Expansion of the greedy SBD objective. Expanding the Frobenius norm in (7) and using the fact that $\|\mathbf{w}_t\mathbf{q}_t^\top\|_F^2 = \|\mathbf{w}_t\|_F^2\|\mathbf{q}_t\|_F^2 = |S_t||T_t|$ for binary vectors, the minimization reduces to the following (dropping the constant $\|\mathbf{R}_0^{(t)}\|_F^2$):

$$\min_{\substack{S_t,\,T_t \\ \alpha_t\in\{-1,+1\}}} \; |S_t||T_t| - 2\alpha_t \sum_{(i,j)\in S_t\times T_t} \mathbf{R}_0^{(t)}(i,j) + \lambda\left(|S_t| + |T_t|\right). \tag{14}$$

Recall that we define the block density of a matrix $\mathbf{M}$ on $S \times T$ as:

$$d_{\mathbf{M}}(S,T) = \frac{1}{|S||T|} \sum_{(i,j)\in S\times T} \mathbf{M}(i,j). \tag{15}$$

Substitution of the residual. Substituting $\mathbf{R}_0^{(t)} = \mathbf{A} - \sum_{i=0}^{t-1} \alpha_i^*\mathbf{w}_i^*(\mathbf{q}_i^*)^\top$ into (14) and using (15) yields:

$$\min_{\substack{S_t,\,T_t \\ \alpha_t\in\{-1,+1\}}} \; |S_t||T_t|\left(1 - 2\alpha_t\, d_{\mathbf{A}}(S_t,T_t)\right) + 2\sum_{i=0}^{t-1}(\alpha_t\alpha_i^*)\left(\mathbf{w}_t^\top\mathbf{w}_i^*\right)\left(\mathbf{q}_t^\top\mathbf{q}_i^*\right) + \lambda\left(|S_t| + |T_t|\right). \tag{16}$$

Since $\mathbf{w}_t^\top\mathbf{w}_i^* = |S_t \cap S_i^*|$ and $\mathbf{q}_t^\top\mathbf{q}_i^* = |T_t \cap T_i^*|$ for binary vectors, the cross term satisfies:

$$\sum_{i=0}^{t-1}(\alpha_t\alpha_i^*)\left(\mathbf{w}_t^\top\mathbf{w}_i^*\right)\left(\mathbf{q}_t^\top\mathbf{q}_i^*\right) = |S_t||T_t|\sum_{i=0}^{t-1}(\alpha_t\alpha_i^*)\frac{|S_t\cap S_i^*|}{|S_t|}\frac{|T_t\cap T_i^*|}{|T_t|}. \tag{17}$$

Factoring out $|S_t||T_t|$ from (16), the objective reduces to:

$$\min_{\substack{S_t,\,T_t \\ \alpha_t\in\{-1,+1\}}} \; |S_t||T_t| \cdot \mathcal{C}(S_t,T_t,\alpha_t), \tag{18}$$

where:

$$\mathcal{C}(S_t,T_t,\alpha_t) = 1 - 2\alpha_t\, d_{\mathbf{A}}(S_t,T_t) + 2\sum_{i=0}^{t-1}(\alpha_t\alpha_i^*)\frac{|S_t\cap S_i^*|}{|S_t|}\frac{|T_t\cap T_i^*|}{|T_t|} + \frac{\lambda}{|S_t|} + \frac{\lambda}{|T_t|}. \tag{19}$$

Interpretation of $\mathcal{C}$. The three terms in (19) have distinct roles:

- **Density reward** $-2\alpha_t\, d_{\mathbf{A}}(S_t,T_t)$: rewards bicliques that are aligned with the block structure of $\mathbf{A}$. For nonnegative $\mathbf{A}$, this term favours $\alpha_t = +1$ when the block density is positive. For signed input, it favours the sign of $d_{\mathbf{A}}(S_t,T_t)$.
- **Signed overlap penalty** $2(\alpha_t\alpha_i^*)\frac{|S_t\cap S_i^*|}{|S_t|}\frac{|T_t\cap T_i^*|}{|T_t|}$: acts as an *implicit memory mechanism*. Same-sign overlap ($\alpha_t\alpha_i^* = +1$) raises $\mathcal{C}$, discouraging the algorithm from covering the same region twice with the same sign. Opposite-sign overlap ($\alpha_t\alpha_i^* = -1$) lowers $\mathcal{C}$, encouraging self-correction via an opposite-sign biclique on a previously covered region.
- **Size penalty** $\lambda/|S_t| + \lambda/|T_t|$: tends to zero as both supports grow and equals 2λ for a singleton, penalising small bicliques more strongly. The parameter λ directly controls this bias.

We denote *overlap-corrected density* as $\tilde{d}_t(S_t, T_t) = d_{\mathbf{A}}(S_t, T_t) - \sum_{i=0}^{t-1} \alpha_i^* \frac{|S_t \cap S_i^*|}{|S_t|} \frac{|T_t \cap T_i^*|}{|T_t|}$.

$\mathcal{C}(S_t, T_t, \alpha_t) < 0$ implies:

$$\alpha_t \tilde{d}_t(S_t, T_t) > \frac{1}{2}\left(1 + \frac{\lambda}{|S_t|} + \frac{\lambda}{|T_t|}\right). \tag{20}$$

Since the right-hand side is strictly positive, $\alpha_t^\star = \operatorname{sign}(\tilde{d}_t(S_t, T_t))$ and

$$|\tilde{d}_t(S_t, T_t)| > \frac{1}{2}\left(1 + \frac{\lambda}{|S_t|} + \frac{\lambda}{|T_t|}\right). \tag{21}$$

Acceptance requires the absolute residual density to exceed $\frac{1}{2}(1+\lambda/|S_t|+\lambda/|T_t|)$. Here $\tilde{d}_t(S_t, T_t) = d_{\mathbf{R}_0^{(t)}}(S_t, T_t)$ is the current residual density. This has three consequences: (i) the threshold is at least $\frac{1}{2}$; for $\lambda > 0$, it approaches $\frac{1}{2}$ from above for large blocks and grows for small ones, so small blocks must be denser to be accepted; (ii) larger λ raises the threshold; in particular, when residual entries are bounded by 1 in magnitude (e.g. binary $\mathbf{A}$ before any overlap), $|S_t| \times |T_t|$ block can be extracted, even when fully dense, only if $\lambda/|S_t| + \lambda/|T_t| < 1$; (iii) same-sign past overlap reduces $\alpha_t \tilde{d}_t$, requiring higher raw density $d_{\mathbf{A}}$ to justify revisiting an already-covered region.

B.2 Derivation of the AO closed-form update

Separability of the gain at fixed S. Fix the sign α and the source set S. Recall that the gain reads

$$G_\lambda(\alpha, S, T, \mathbf{R}_0) \;=\; 2\sum_{j \in T} \underbrace{\alpha \sum_{i \in S} \mathbf{R}_0(i,j)}_{c_j(S,\alpha)} \;-\; |S|\,|T| \;-\; \lambda\big(|S| + |T|\big), \tag{22}$$

where $c_j(S, \alpha) = \alpha \sum_{i \in S} \mathbf{R}_0(i, j)$. G_λ decomposes as

$$G_\lambda(\alpha, S, T, \mathbf{R}_0) = \sum_{j \in T} \big(2\, c_j(S) - |S| - \lambda\big) \;-\; \lambda|S|. \tag{23}$$

Each j contributes independently: including j in T improves the gain if and only if

$$\frac{c_j(S, \alpha)}{|S|} = \frac{1}{|S|} \sum_{i \in S} \alpha\, \mathbf{R}_0(i, j) \;>\; \frac{1}{2} + \frac{\lambda}{2|S|}. \tag{24}$$

The optimal T at fixed S therefore consists exactly of all columns whose average connection to S exceeds $\frac{1}{2} + \frac{\lambda}{2|S|}$. Denoting this set by

$$T^\star = \left\{ j : \frac{c_j(S)}{|S|} > \frac{1}{2} + \frac{\lambda}{2|S|} \right\}, \tag{25}$$

we have

$$G_\lambda(\alpha, S, T^\star, \mathbf{R}_0) \geq G_\lambda(\alpha, S, T, \mathbf{R}_0) \qquad \text{for any } T. \tag{26}$$

Symmetrically, for fixed T, the corresponding update $S^\star$ satisfies

$$G_\lambda(\alpha, S^\star, T, \mathbf{R}_0) \geq G_\lambda(\alpha, S, T, \mathbf{R}_0) \qquad \text{for any } S. \tag{27}$$

Thus, starting from an initial seed, alternating these two updates yields an alternating-optimization procedure whose gain is non-decreasing at each step.

Minimum-density floor $d_{\min}$ (binary matrices). For a current residual $\mathbf{R}_0 \in \{0,1\}^{n \times n}$ and sign $\alpha = +1$, entries $(i, j) \in S \times T$ with $\mathbf{R}_0(i, j) = 0$ are *holes*: subtracting $\mathbf{1}_S \mathbf{1}_T^\top$ from $\mathbf{R}_0$ sets them to -1, adding nonzero residual entries at those positions. The hole count equals exactly $|S||T|(1 - d_{\mathbf{R}_0}(S, T))$, so controlling $d_{\mathbf{R}_0}(S, T)$ directly limits holes.

We impose a per-node density floor: include j only if $c_j(S, \alpha)/|S| > d_{\min}$. Combined with (24), this gives

$$2\alpha \sum_{i \in S} \mathbf{R}_0(i, j) \;>\; \max\big(|S| + \lambda,\; 2d_{\min}|S|\big), \tag{28}$$

Setting $d_{\min} = 0$ recovers the pure half-step. Averaging gives $\alpha d_{\mathbf{R}_0}(S, T) > d_{\min}$ for a nonempty AO output. For a binary current residual and positive sign, this bounds holes by $|S||T|(1 - d_{\min})$.

Neighbourhood initialisation. Seeds are ranked separately by positive and negative residual. We take up to M seeds per sign and orientation (row or column), giving at most $4M$ starts per extraction. Algorithm 1 shows the row-seed case.

For sign α, we expand the neighbourhood along entries with $\alpha \mathbf{R}_0(i,j) > 0$. A row seed first reaches columns, then rows, then columns again; a column seed uses the transposed initialisation. We search both seed types. AO updates rows first for a row seed, and columns first for a column seed. Before AO, we trim the candidate sets using alternating row and column updates. If either set is empty, the initial window is kept. Trimming reduces search cost but can exclude positive-gain blocks. AO uses residual weights within the retained sets. Acceptance checks the full gain.

B.3 Positive-gain certificates

The following conditions identify seeds that can yield positive-gain. Let $T_0 \subseteq V$ be the current nonempty column set and let $d = |T_0|$. For each row i, define

$$c_i(T_0, \alpha) = \alpha \sum_{j \in T_0} R(i,j).$$

The row update keeps i whenever

$$2c_i(T_0, \alpha) > h_d, \qquad h_d = \max\{d + \lambda,\, 2d_{\min} d\}.$$

Proposition 1 (Shared-neighborhood certificate)**.** *Assume that there is a nonempty set $P \subseteq U$ such that*

$$c_i(T_0, \alpha) \geq c_{\min} \qquad \textit{for all } i \in P,$$

with

$$2c_{\min} > h_d$$

and

$$|P|(2c_{\min} - d - \lambda) > \lambda d.$$

Then the next row update has positive gain. Later updates preserve positive gain if they maximise the gain with the other support fixed. An active density floor need not preserve this property.

Proof. Every row in P passes the update threshold. At fixed T_0, the gain can be written as

$$G_\lambda(\alpha, S, T_0, R) = \sum_{i \in S} (2c_i(T_0, \alpha) - d - \lambda) - \lambda d.$$

Let S_1 be the row support returned by the update. Since every selected row has positive contribution,

$$\begin{aligned} G_\lambda(\alpha, S_1, T_0, R) &\geq \sum_{i \in P} (2c_i(T_0, \alpha) - d - \lambda) - \lambda d \\ &\geq |P|(2c_{\min} - d - \lambda) - \lambda d > 0. \end{aligned}$$

Each later exact coordinate update maximizes the gain with the other support fixed. Hence the gain cannot decrease. □

For the initialization $T_0 = \mathcal{N}(u)$ and $\alpha = +1$,

$$c_i(T_0, +1) = \sum_{j \in \mathcal{N}(u)} R(i,j).$$

For a binary residual, this is the number of neighbors shared by i and the seed u.

A second certificate uses the clustering coefficient.

Proposition 2 (Clustering certificate)**.** *Let R be the adjacency matrix of a simple undirected binary graph. Let u have degree $d_u > 1$ and clustering coefficient C_u. Then the block*

$$S = T = \mathcal{N}(u)$$

has positive gain for $\alpha = +1$ if and only if

$$C_u > \frac{d_u + 2\lambda}{2(d_u - 1)}.$$

Proof. Let e_u be the number of edges between the neighbors of u. Then

$$2e_u = C_u d_u(d_u - 1).$$

For $S = T = \mathcal{N}(u)$,

$$G_\lambda(+1, S, T, R) = 4e_u - d_u^2 - 2\lambda d_u.$$

Thus $G_\lambda > 0$ if and only if

$$C_u > \frac{d_u + 2\lambda}{2(d_u - 1)}.$$

□

The two certificates have different roles. The shared-neighborhood condition applies to the AO update. The clustering condition shows that the neighborhood block has positive gain, but it does not guarantee that AO finds this block.

B.4 Exact-recovery guarantees

Fix R, a sign α, and scoped domains U, V. Let $(S^\star, T^\star)$ be a positive-gain fixed point of the actual updates, including the density floor and tie rule. Let $T_0 \subseteq V$ be the nonempty initial column set and define

$$h_0 = \max\{|T_0| + \lambda,\, 2d_{\min}|T_0|\}.$$

Proposition 3 (Recovery of a fixed point)**.** *Assume*

$$2c_i(T_0, \alpha) \begin{cases} > h_0, & i \in S^\star, \\ \leq h_0, & i \in U \setminus S^\star. \end{cases}$$

One row update followed by one column update recovers $(S^\star, T^\star)$.

Proof. The separation condition makes the first row update select exactly $S^\star$. Since the target is a fixed point, the next column update returns $T^\star$. □

If the target also maximises gain over the scoped domain, the recovered block is optimal there. This does not imply optimality outside that domain. Global optimality alone does not ensure a fixed point of updates with an active density floor or a different tie rule. For a binary residual and $T_0 = \mathcal{N}(u)$, $c_i(T_0, +1)$ counts shared neighbours. Recovery requires the threshold to separate target rows from all other candidates after scoping.

B.5 Bounds from empty updates

If a column update from nonempty S returns no column in V, then

$$\alpha d_R(S, \{j\}) \leq \max\left\{\frac{1}{2} + \frac{\lambda}{2|S|}, d_{\min}\right\} \quad \text{for every } j \in V.$$

This follows by reversing the update inequality and dividing by $2|S|$. An empty row update gives the symmetric bound. Checking both signs bounds the absolute density. The result concerns the final remainder only when the update was evaluated on that remainder. It covers only the searched domain.

C Additional analysis

C.1 Local positive-gain structure

Table 5 reports the rates at $\lambda = 1$. The script evaluates the original binary graph, before scoping, with $T_0 = \mathcal{N}(u)$. Its local test uses $h_d = d + \lambda$, without an active density floor. These rates do not certify the candidates retained by the implemented search.

Table 5: Percentage of nodes satisfying the clustering or shared-neighborhood positive-gain condition at $\lambda = 1$. Tests use the original binary graph before scoping and do not apply a density floor. These rates describe available local structure, not the success rate of the implemented search.

Dataset	Clustering (%)	Shared neighbors (%)
Cora	0.1	9.1
PubMed	0.0	12.1
ca-HepPh	22.5	43.7
ca-AstroPh	30.2	57.0
coPapersDBLP	78.0	91.6
coPapersCiteseer	83.3	93.9
COLLAB	84.3	96.5

The certificate rates are low on Cora and PubMed and high on the coauthorship and copaper graphs. Higher rates generally come with larger CF values. These rates describe local positive-gain structure. They are not bounds on the final CF.

C.2 Impact of the Regularization Parameter λ

We first derive the gain thresholds for several block shapes. We then measure how changing λ affects the residual and compression.

Threshold interpretation. Consider a single nonzero residual entry $\mathbf{R}_0^{(t)}(i,j) \neq 0$. For a singleton biclique $S_t = \{i\}$, $T_t = \{j\}$, the overlap-corrected density is the entry itself, $\tilde{d}_t(\{i\},\{j\}) = \mathbf{R}_0^{(t)}(i,j)$, and the acceptance condition reads

$$|\mathbf{R}_0^{(t)}(i,j)| > \tfrac{1}{2}(1+2\lambda),$$

which is exactly the singleton gain condition used in the proof of Theorem 1, claim 2. Since residual entries are integers, every nonzero entry satisfies $|\mathbf{R}_0^{(t)}(i,j)| \geq 1$, so for $\lambda < \frac{1}{2}$ every nonzero residual entry yields positive singleton gain. Exact recovery then requires the singleton stopping certificate of Corollary 1. For $\lambda \geq \frac{1}{2}$, entries of magnitude 1 can no longer be removed by a singleton, and singleton corrections are never accepted on residuals with entries in $\{-1, 0, 1\}$.

Structure-dependent thresholds. The parameter λ controls which blocks have positive gain. Acceptance requires

$$|\tilde{d}_t(S_t, T_t)| > \frac{1}{2}\left(1 + \frac{\lambda}{|S_t|} + \frac{\lambda}{|T_t|}\right).$$

The threshold is at least $\frac{1}{2}$ and increases with λ. When residual entries are bounded by 1 in magnitude (e.g. binary $\mathbf{A}$ before any overlap), we have $|\tilde{d}_t| \leq 1$, so a block of shape $|S_t| \times |T_t|$ can be extracted, even when fully dense, only if

$$\frac{\lambda}{|S_t|} + \frac{\lambda}{|T_t|} < 1.$$

We call this bound the *extraction limit* for the block shape. Equation (21) gives:

- **Singleton** ($|S_t| = |T_t| = 1$): threshold is $|\tilde{d}_t| > \frac{1}{2}(1+2\lambda)$; extraction limit $\lambda < \frac{1}{2}$.
- **Star** ($|S_t| = 1$, $|T_t| = k$): threshold is $|\tilde{d}_t| > \frac{1}{2}(1+\lambda+\lambda/k)$; extraction limit $\lambda < \frac{k}{k+1}$. For large k, the threshold approaches $\frac{1}{2}(1+\lambda)$, below the singleton threshold, making wide stars easier to accept than singletons.

Table 6: Extraction limits for fully dense blocks when residual entries have magnitude at most one. Larger blocks allow higher λ.

Structure	$\lvert S_t\rvert$	$\lvert T_t\rvert$	Extraction limit ($\lambda <$)
Singleton	1	1	$\frac{1}{2}$
Star ($k = 5$)	1	5	$\frac{5}{6}$
2×2	2	2	1
2×3	2	3	$\frac{6}{5}$

- 2×2 **biclique** ($|S_t| = |T_t| = 2$): threshold is $|\tilde{d}_t| > \frac{1}{2}(1 + \lambda)$; extraction limit $\lambda < 1$.
- 2×3 **biclique** ($|S_t| = 2$, $|T_t| = 3$): threshold is $|\tilde{d}_t| > \frac{1}{2}(1 + \frac{5\lambda}{6})$; extraction limit $\lambda < \frac{6}{5}$.

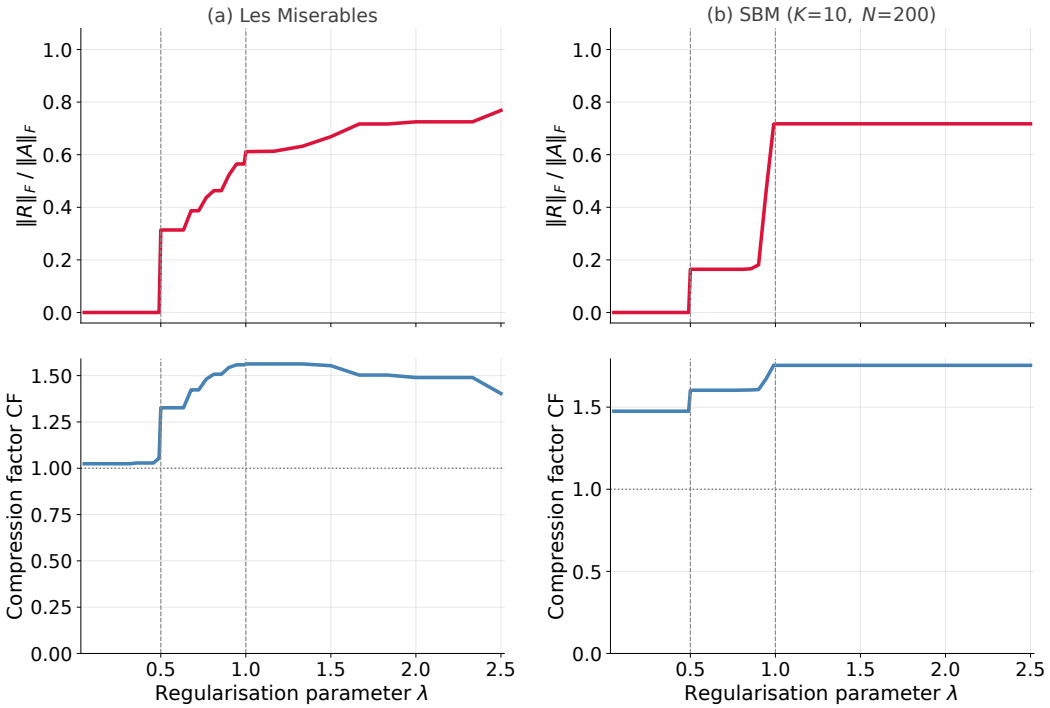


Figure 4: Normalised residual $\|\mathbf{R}_0^{(K)}\|_F / \|\mathbf{A}\|_F$ and compression factor CF as a function of λ. Dashed vertical lines mark $\lambda = \frac{1}{2}$ (unit-singleton threshold) and $\lambda = 1$ (the range where the storage bound applies).

Effect on residual norm. For a fixed residual, increasing λ raises the acceptance threshold. The search path can change, so the final residual norm need not increase with λ.

- $\lambda < \frac{1}{2}$: every nonzero integer entry gives positive singleton gain. A stopping certificate covering all signed singleton supports therefore gives $\mathbf{R}_0^{(K)} = 0$ by Corollary 1. A finite seed search or budget limit alone does not give this certificate.
- $\lambda \in [\frac{1}{2}, 1)$: unit entries cannot be removed by singleton updates. When residual entries have magnitude at most one, a star with k leaves requires $\lambda < k/(k+1)$. Larger dense blocks may still be accepted. The singleton stopping certificate implies $|\mathbf{R}_0^{(K)}(i,j)| \leq 1$.
- $\lambda \geq 1$: when residual entries have magnitude at most one, stars and 2×2 blocks are rejected, and acceptance requires $1/|S_t| + 1/|T_t| < 1/\lambda$. Theorem 1 gives a lower bound on CF, not compression optimality or monotone CF.

Binary input does not ensure bounded residual entries after overlapping updates. Figures 4 and 5 show the measured trends. Small late blocks can reduce CF when their storage cost exceeds the residual entries they remove.

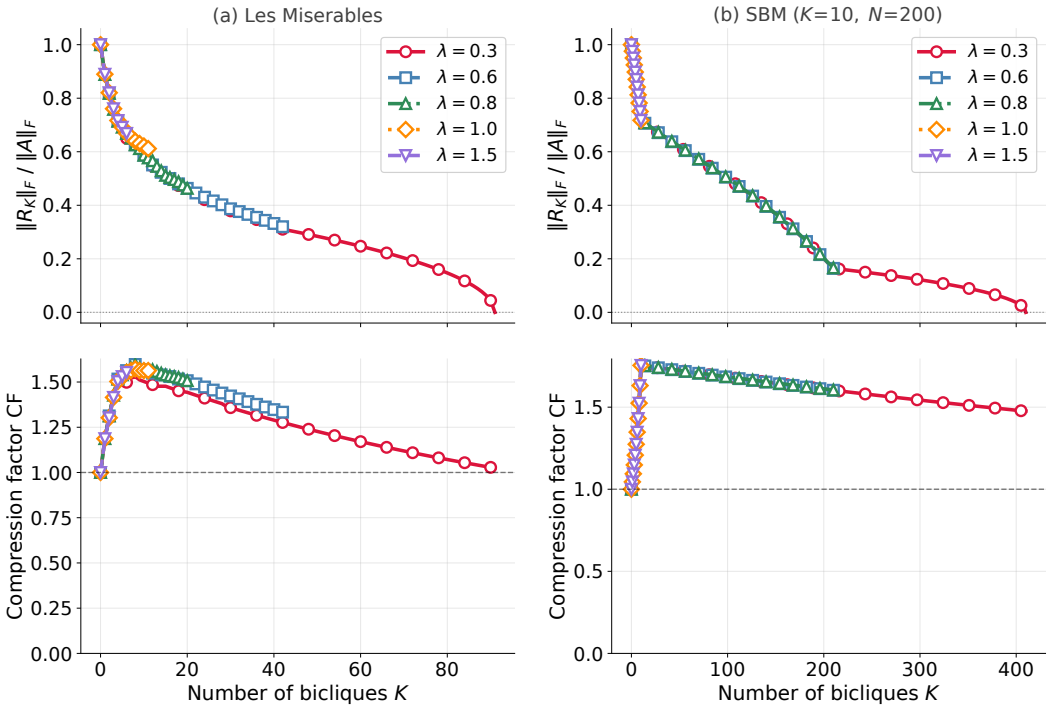


Figure 5: Normalised residual $\|\mathbf{R}_0^{(K)}\|_F/\|\mathbf{A}\|_F$ (top row) and compression factor CF_K (bottom row) as a function of the number of extracted bicliques K, for five values of λ. **(a)** Les Misérables ($N = 77$). **(b)** SBM ($K = 10$, $N = 200$). Dashed horizontal line in bottom row: CF$= 1$ (break-even point). On the SBM, $\lambda \in \{1.0, 1.5\}$ converge after exactly 10 bicliques in this run. At the plotted $\lambda = 0.3$, the residual reaches zero in both graphs. For small λ, CF declines at large K once the remaining extractions are singletons and small stars whose index cost exceeds the edges they remove.

C.3 Comparison with a rank-one solver

We compare the AO heuristics with a mixed-integer solver for each greedy rank-one problem. Each sign is solved as a binary quadratic problem with a 60-second limit. We save the incumbent, solver status, bound and gap for both signs. A step is certified only when both signs are solved to optimality. Greedy and Fast use the current deterministic seed ordering.

We run the comparison on Karate Club, Les Misérables, and Dolphins for

$$\lambda \in \{0, 0.5, 1, 2, 4\}.$$

For each method, we report the number of extracted bicliques K, the Compression Factor, and the residual Frobenius norm.

The methods often give similar CF and residual norms, but can return different numbers of blocks. This compares final decompositions; it does not establish the per-step gap to an exact oracle or test Proposition 3.

C.4 Why SVD-based Methods are Mismatched to our Objective

Standard sparse-plus-low-rank methods such as RPCA [8] decompose $\mathbf{A} = \mathbf{L}_0 + \mathbf{R}_0$ where $\mathbf{L}_0$ is a real-valued low-rank matrix and $\mathbf{R}_0$ is sparse. These are designed for settings where a *dense continuous signal* is corrupted by sparse noise. Our objective is different by construction: we seek a decomposition into *binary integer* rank-one bicliques plus a sparse integer residual. The two problems differ in what the low-rank component is allowed to be.

Table 7: Rank-one solver and AO heuristics on three small graphs as λ varies. We report the number of bicliques K, compression factor CF, and residual norm after decomposition. Each signed solver subproblem has a 60-second limit. A dagger marks a decomposition with an uncertified step. Rank-one optima do not imply a global optimum for the final decomposition.

	Rank-one MIQP			SBDGreedy			SBDFast		
λ	K	CF	$\|R_0\|_F$	K	CF	$\|R_0\|_F$	K	CF	$\|R_0\|_F$
Karate									
0.0	33	0.79	0.00	30	0.88	0.00	33	0.63	2.00
0.5	24	0.94	2.45	24	0.94	2.45	24	0.96	3.61
1.0	9	1.17	7.48	8	1.17	7.87	9	1.18	7.35
2.0	0	1.00	12.5	0	1.00	12.5	0	1.00	12.5
4.0	0	1.00	12.5	0	1.00	12.5	0	1.00	12.5
Les Misérables									
0.0	88	1.03	0.00	88	1.03	0.00	73	0.96	5.57
0.5	43	1.33	6.93	45	1.28	6.71	50	1.23	8.12
1.0	12	1.56	13.5	12	1.54	13.7	16	1.46	13.8
2.0	4	1.50	16.2	4	1.49	16.3	4	1.40	17.1
4.0	1	1.18	20.1	1	1.18	20.1	1	1.15	20.4
Dolphins									
0.0	86	0.69†	0.00	89	0.69	0.00	79	0.62	5.29
0.5	62	0.81†	5.39	61	0.80	5.57	58	0.82	7.48
1.0	14	1.09	13.6	10	1.09	14.4	16	1.07	13.4
2.0	0	1.00	17.8	0	1.00	17.8	0	1.00	17.8
4.0	0	1.00	17.8	0	1.00	17.8	0	1.00	17.8

SVD factors and community supports. For an isolated loopless clique of size B, the leading eigenvalue is $B-1$. Its scaled factor is $\sqrt{(B-1)/B}\,\mathbf{1}_{S_k}$. Equal-sized cliques have repeated eigenvalues, so their eigenvectors can mix. Inter-community edges can also spread the factors across communities, as in Figure 6. SVD does not enforce binary factors or a sparse remainder.

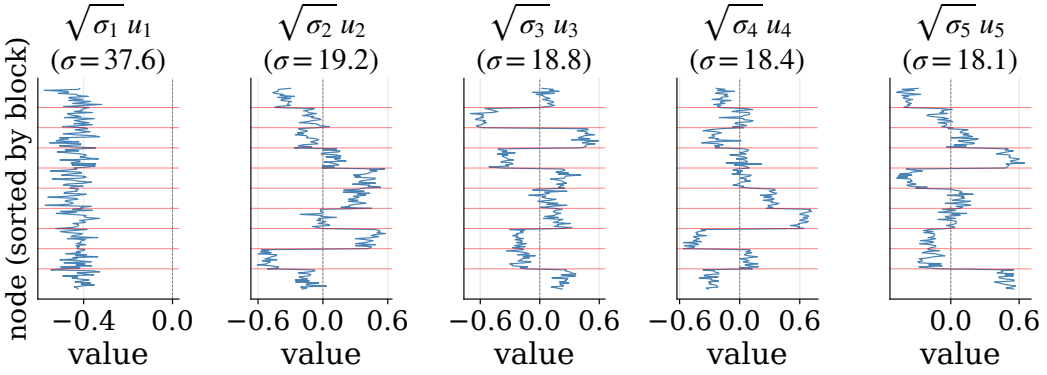


Figure 6: Top-K symmetric factors $\sqrt{\sigma_k}\,u_k$ of an SBM adjacency matrix ($K = 10$, $N = 200$, $p_{\text{in}} = 1.0$, $p_{\text{out}} = 0.1$), nodes sorted by ground-truth community (red lines = block boundaries). An isolated loopless clique has scaled factor $\sqrt{(B-1)/B}\,\mathbf{1}_{S_k}$; repeated eigenvalues allow mixtures. In this example, factors spread across communities, producing a real-valued, non-binary vector.

Sparsity on factors does not imply binariness. Our sparseSVD baseline alternates soft-thresholded row and column updates with rank-one deflation. It promotes sparsity on the factors, which might seem closer to our setting. However, sparsity does not make the factors binary. The low-rank component $\mathbf{L}_0^{\text{SparseSVD}} = \sum_k \sigma_k u_k v_k^\top$ is a sum of sparse real-valued outer products, entries take arbitrary values in $\mathbb{R}$, not in $\{0, 1\}$. We verify this even in the most favorable case: SBM with $K = 10$, $N = 200$, $p_{\text{in}} = 1.0$, $p_{\text{out}} = 0.1$, SparseSVD rank set to the true K, $\lambda = 0.1$. Even here, **75.6%** of entries of $\mathbf{L}_0^{\text{SparseSVD}}$ exceed 0.05 in magnitude, compared to the original graph's **18.5%** nonzeros (Figure 7).

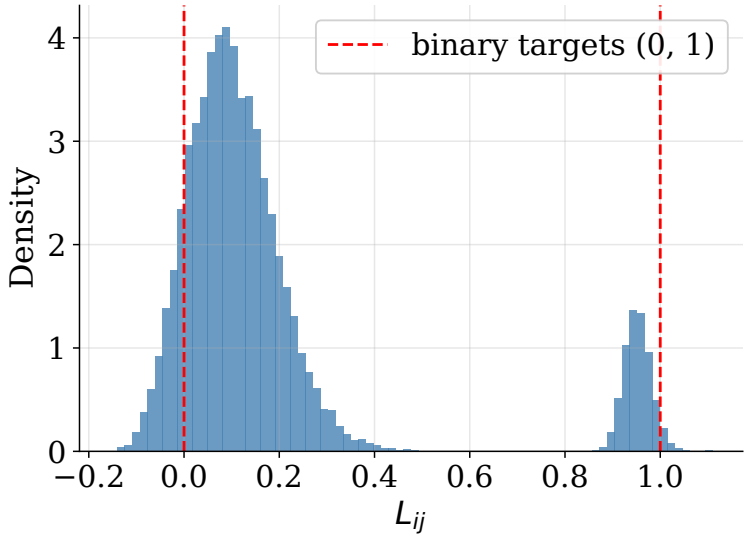


Figure 7: Distribution of $\mathbf{L}_0^{\text{SparseSVD}}$ entries for SBM ($K = 10$, $N = 200$, $p_{\text{in}} = 1.0$, $p_{\text{out}} = 0.1$), SparseSVD rank is 10, $\lambda = 0.1$.

Scalability. On sparse graphs, each matrix-vector product costs $O(E)$. RPCA repeatedly applies singular-value thresholding to dense intermediate matrices, which increases memory use and runtime.

C.5 Full kernel comparison

Table 8 gives the per-width results for Figure 2.

Figure 2 compares SpMM speedups over cuSPARSE. SBD_base uses the same kernel without bicliques, so the SBD gap measures their effect. The largest gains are on coPapersDBLP, coPapersCiteseer, and COLLAB, which have high CF. On COLLAB, mean SBD speedup is $6.37\times$ at $F = 32$, $3.89\times$ at $F = 128$, and $3.86\times$ at $F = 256$ (Appendix C.5). Factor multiplications add overhead on Cora and PubMed. Compression does not guarantee faster SpMM.

Table 8: Mean ± sample standard deviation of speedups over cuSPARSE across RTX 3090, A40, and L40S. SBD_base uses no bicliques. All six kernels, including CBM, were measured on each GPU.

Dataset	F	Sputnik	dgSPARSE	GE-SpMM	SBD_base	CBM	SBD
Cora	32	$2.72_{\pm 0.37}$	$1.25_{\pm 0.16}$	$2.20_{\pm 0.18}$	$1.70_{\pm 0.28}$	$0.17_{\pm 0.06}$	$1.45_{\pm 0.19}$
Cora	128	$2.86_{\pm 0.62}$	$1.41_{\pm 0.12}$	$2.34_{\pm 0.53}$	$1.75_{\pm 0.40}$	$0.19_{\pm 0.08}$	$1.64_{\pm 0.26}$
Cora	256	$2.34_{\pm 0.35}$	$1.21_{\pm 0.04}$	$2.12_{\pm 0.47}$	$1.49_{\pm 0.21}$	$0.18_{\pm 0.06}$	$1.49_{\pm 0.20}$
PubMed	32	$2.68_{\pm 0.40}$	$1.43_{\pm 0.13}$	$2.83_{\pm 0.70}$	$1.83_{\pm 0.30}$	$0.21_{\pm 0.07}$	$1.68_{\pm 0.31}$
PubMed	128	$1.84_{\pm 0.26}$	$1.25_{\pm 0.02}$	$1.80_{\pm 0.25}$	$1.47_{\pm 0.11}$	$0.25_{\pm 0.10}$	$1.38_{\pm 0.05}$
PubMed	256	$1.73_{\pm 0.37}$	$1.17_{\pm 0.10}$	$1.86_{\pm 0.25}$	$1.39_{\pm 0.02}$	$0.34_{\pm 0.12}$	$1.34_{\pm 0.06}$
ca-HepPh	32	$2.21_{\pm 0.29}$	$0.91_{\pm 0.03}$	$1.72_{\pm 0.36}$	$1.11_{\pm 0.25}$	$0.19_{\pm 0.06}$	$1.39_{\pm 0.28}$
ca-HepPh	128	$1.70_{\pm 0.16}$	$0.91_{\pm 0.01}$	$1.20_{\pm 0.16}$	$1.03_{\pm 0.10}$	$0.23_{\pm 0.05}$	$1.47_{\pm 0.11}$
ca-HepPh	256	$1.60_{\pm 0.09}$	$0.85_{\pm 0.08}$	$1.24_{\pm 0.14}$	$1.05_{\pm 0.09}$	$0.31_{\pm 0.06}$	$1.36_{\pm 0.21}$
ca-AstroPh	32	$2.00_{\pm 0.19}$	$1.07_{\pm 0.07}$	$1.77_{\pm 0.28}$	$1.15_{\pm 0.18}$	$0.23_{\pm 0.07}$	$1.66_{\pm 0.26}$
ca-AstroPh	128	$1.40_{\pm 0.17}$	$1.02_{\pm 0.03}$	$1.24_{\pm 0.23}$	$1.11_{\pm 0.17}$	$0.29_{\pm 0.06}$	$1.37_{\pm 0.16}$
ca-AstroPh	256	$1.32_{\pm 0.19}$	$0.85_{\pm 0.13}$	$1.28_{\pm 0.25}$	$0.94_{\pm 0.07}$	$0.44_{\pm 0.07}$	$1.32_{\pm 0.37}$
coPapersDBLP	32	$1.56_{\pm 0.43}$	$1.70_{\pm 0.18}$	$2.13_{\pm 0.04}$	$1.76_{\pm 0.04}$	$1.89_{\pm 0.32}$	$5.03_{\pm 0.67}$
coPapersDBLP	128	$0.92_{\pm 0.34}$	$1.07_{\pm 0.04}$	$1.73_{\pm 0.15}$	$1.68_{\pm 0.23}$	$1.42_{\pm 0.30}$	$3.23_{\pm 0.94}$
coPapersDBLP	256	$0.92_{\pm 0.34}$	$0.95_{\pm 0.06}$	$1.72_{\pm 0.16}$	$1.45_{\pm 0.23}$	$1.50_{\pm 0.31}$	$3.21_{\pm 0.96}$
coPapersCiteseer	32	$2.11_{\pm 0.37}$	$1.79_{\pm 0.25}$	$2.12_{\pm 0.05}$	$1.78_{\pm 0.07}$	$2.18_{\pm 0.33}$	$5.80_{\pm 0.74}$
coPapersCiteseer	128	$1.26_{\pm 0.24}$	$1.03_{\pm 0.05}$	$1.98_{\pm 0.03}$	$2.15_{\pm 0.06}$	$1.79_{\pm 0.40}$	$3.92_{\pm 1.19}$
coPapersCiteseer	256	$1.27_{\pm 0.24}$	$0.92_{\pm 0.09}$	$1.95_{\pm 0.07}$	$2.09_{\pm 0.08}$	$1.98_{\pm 0.45}$	$3.89_{\pm 1.19}$
COLLAB	32	$2.21_{\pm 0.27}$	$1.82_{\pm 0.36}$	$2.09_{\pm 0.06}$	$1.75_{\pm 0.16}$	$2.60_{\pm 0.34}$	$6.37_{\pm 0.69}$
COLLAB	128	$1.26_{\pm 0.29}$	$0.97_{\pm 0.07}$	$1.91_{\pm 0.05}$	$2.45_{\pm 0.05}$	$1.86_{\pm 0.39}$	$3.89_{\pm 1.11}$
COLLAB	256	$1.27_{\pm 0.28}$	$0.89_{\pm 0.02}$	$1.91_{\pm 0.03}$	$2.78_{\pm 0.05}$	$2.02_{\pm 0.45}$	$3.86_{\pm 1.07}$

C.6 Feature width and retained blocks

On COLLAB, the fastest sampled prefix uses 30,242 of 53,646 blocks at $F = 32$, 1,722 at $F = 128$, and 308 at $F = 256$. Adding more blocks can slow down SpMM. These are measurements for the plotted configuration, not universal thresholds across GPUs. At $F = 32$, eight lanes per warp load the eight `float4` feature groups. At $F = 128$, all 32 lanes load one group. This may affect memory throughput, but the curves alone do not establish bandwidth saturation or exclude cache effects. Profiling is needed to test that explanation.

D Implementation details

D.1 SBDFast implementation (binary matrices)

SBDFast has two search phases. First, it proposes positive blocks using a binary mask of the positive residual entries. Then it searches for positive and negative blocks using the integer residual. Both phases maintain the exact residual and use it to check each proposal before acceptance. Negative entries can arise from holes or overlaps in positive blocks.

Parallel search and sequential acceptance. Workers process seed nodes in batches against a fixed residual. They return potential blocks, which are ordered by their search gains. We then check each block in this order. We recompute its gain on the current residual and accept it if the gain is positive and the density condition holds. After each acceptance, we subtract the block from the residual before checking the next proposal. Several blocks can therefore be accepted from one batch, reducing the number of parallel searches. In comparison, SBDGreedy accepts only the best block from each search.

Exact residual updates. The residual uses sparse row and column arrays, with separate lists for fill-ins. Each accepted block updates all entries in its support, including zeros (which can become fill-ins) and entries of the opposite sign. Compaction merges the extra lists into the sparse arrays and removes zeros. It changes the storage in the algorithm, not the residual.

Search settings. The seed budget controls search effort independently of the thread count. The main compression and runtime experiments use one signed sweep; the rank-one comparison uses up to 20 signed sweeps. The candidate row limit remains 400.

Guarantees. Every accepted block has positive gain on the current exact residual. Theorem 1 therefore gives residual decrease and objective descent for SBDFast. For integer input and $\lambda \geq 0$, there are at most $\|\mathbf{A}\|_F^2$ accepted blocks. The search remains heuristic: it does not certify that no positive-gain block remains. The stopping bounds require nonpositive gain checks on the final residual. Exact recovery requires the additional certificate in Corollary 1.

D.2 Experimental protocol and hyperparameters

D.2.1 Denoising

Search and evaluation. The Optuna sampler is TPE. Each graph has ten blocks of 20 nodes and $p_{\text{in}} = 1$. The noise levels are $q \in \{0.05, 0.15, 0.25, 0.35\}$.

Search spaces. SBDGreedy searches $\lambda \in [0.5, 20]$ log-uniformly, $d_{\min} \in [0.5, 1]$ uniformly, and an integer step cap in $\{5, \ldots, 100\}$. It uses 200 seeds and eight threads. RPCA, Kanada, and sparse SVD search $\lambda \in [10^{-3}, 2]$ uniformly. Sparse SVD fixes rank 10. Each search uses 30 trials. Savalle uses one 30-trial search with $\tau \in [10^{-3}, 20]$ log-uniformly and $\gamma \in [10^{-3}, 1]$ uniformly.

Denoising solver defaults. RPCA uses 1,000 iterations and tolerance 10^{-7}. Kanada uses 1,000 iterations and tolerance 10^{-6}. Sparse SVD uses 100 power iterations per component and tolerance 10^{-5}. Kanada uses ADMM penalty $\rho = 1$. Sparse SVD uses random initial vectors. Savalle uses 1,000 iterations for tuning and up to 5,000 for evaluation, with tolerance 10^{-6}. A failed convergence check gets one retry at fixed $\rho = 16$.

SBDGreedy uses up to eight scoping rounds and 20 AO rounds per start. These caps, the seed and thread counts are fixed.

Table 9: Selected denoising parameters for Table 1.

Parameter	$q = 0.05$	$q = 0.15$	$q = 0.25$	$q = 0.35$
SBD λ	4.55	4.55	6.81	8.15
SBD $d_{\min}$	0.58	0.58	0.51	0.58
SBD step cap	19	19	98	83
RPCA λ	0.12	0.06	0.06	0.06
Kanada λ	0.12	0.12	0.06	0.06
Sparse SVD λ	0.31	0.41	0.50	0.75
Savalle τ	3.81	6.57	7.55	7.57
Savalle γ	0.21	0.07	0.08	0.09

D.2.2 Decomposition runtime

Fixed settings. Both methods use $\lambda = 0.5$ and eight threads. SBDGreedy uses at most $2n$ extractions, 64 seeds per sign and orientation, and $d_{\min} = 0$. SBDFast uses a nominal seed budget of $2n$, 32 local starts, batches of 32 seeds, one signed sweep, and $d_{\min} = 0.7$. Kanada uses $\lambda = 0.5$. RPCA uses $1/\sqrt{n}$. Both allow up to 5,000 iterations. Sparse SVD extracts up to n components, with $\lambda = 0.1$ and up to 100 iterations per component. Savalle uses the float64 symmetry-based ADMM solver, with $\tau = \gamma = 0.1$, up to 5,000 iterations, and adaptive penalty starting at $\rho = 1$. If it does not converge, it retries with fixed $\rho = 16$ under the same total time limit. All continuous solvers use a tolerance of 10^{-6} with their respective stopping criteria. Each run has a 60-second compute limit.

D.2.3 Compression and cache selection

Search and fixed settings. Each graph uses 30 TPE trials with sampler seed 42. The search samples $\lambda \in [0.5, 4]$ log-uniformly and $d_{\min} \in [0.5, 1]$ uniformly. Each trial uses SBDFast with a nominal seed budget of 500,000, 32 local starts, batches of 32 seeds, one signed sweep, and eight threads. We select the highest CF (Table 10). CBM fixes $\alpha = 0$.

Table 10: Parameters recorded in the selected decomposition caches used for Table 3 and SpMM.

Dataset	λ	$d_{\min}$
Cora	1.31	0.53
PubMed	1.31	0.53
ca-HepPh	1.23	0.50
ca-AstroPh	1.19	0.63
coPapersDBLP	1.51	0.67
coPapersCiteseer	1.11	0.74
COLLAB	1.35	0.70

D.2.4 Compression accounting

Entry-count model. Let $s = \sum_k |S_k|$, $t = \sum_k |T_k|$, and $r = \|\mathbf{R}_0\|_0$. CF counts one unit per membership, sign, and residual entry:

$$\mathrm{CF} = \frac{\|\mathbf{A}\|_0}{s + t + K + r}.$$

CF does not depend on integer width and excludes pointer arrays.

Kernel storage. The input CSR uses $B_A = 4(N + 1) + 8\|\mathbf{A}\|_0$ bytes. SBD stores int32 pointers, membership indices, and residual indices and values. Signs use int8. Factor values are one and are not stored. The row-factor and residual pointers have length $N + 1$. The column-factor pointers have length $K + 1$. Thus

$$B_{\mathrm{SBD}} = 4(2N + K + 3) + 4(s + t) + 8r + K, \qquad \mathrm{CR}_{\mathrm{SBD}} = B_A / B_{\mathrm{SBD}}.$$

These counts exclude features, outputs, temporary buffers, and allocator overhead.

CBM storage. We retain the authors' accounting: $B_{\mathrm{CBM}} = 4(N+1) + 8d + 8e$, where d counts delta entries and e counts forest edges. This includes delta CSR and two forest-edge arrays. It excludes GPU branch offsets. Neither ratio measures total GPU memory.

CBM implementation. Our CBM wrapper uses a custom OpenMP decomposition and the authors' GPU update kernels. It converts the forest to branch offsets, adjusts the branch count, and skips empty update launches. We fix $\alpha = 0$.

D.2.5 SpMM and block-prefix measurements

We evaluate $F \in \{32, 128, 256\}$ with float32 features. The kernel scripts do not sweep decomposition parameters or kernel configurations. CBM uses $\alpha = 0$. Both kernel comparisons use ten warm-up launches and 50 timed launches. Features use seed 42. Reference CSR indices use int32. Every timed kernel is checked against sparse matrix multiplication within numerical tolerance. For the block-prefix experiment, we sort bicliques by area. We request 20 logarithmically spaced prefix lengths from one to all blocks and round them to integers. We rebuild the exact residual for each prefix. We vary K and F without repeating the decomposition search. The script times the reference and candidate in pairs, with defaults of 10 warm-up and 50 timed launches.

D.2.6 Regularization sweep

SBDGreedy uses a 500-step cap, $d_{\min} = 0$, and eight threads. We use 64 seeds on Les Misérables and 200 on the SBM. The summary evaluates 36 values of λ between 0.05 and 2.5. The trajectories use $\lambda \in \{0.3, 0.6, 0.8, 1, 1.5\}$.

D.3 CUDA Kernel Design

This appendix describes the CUDA kernels used for SpMM. SBD_base uses our kernel without bicliques, so comparison with SBD measures the effect of the decomposition.

Warp mapping. The kernel assigns one warp to each output row. Lanes traverse the same CSR neighbour list and process different groups of four features using `float4` loads.

Register accumulation. Each lane accumulates its feature groups in registers. Column aggregation uses additions, signed biclique combination uses additions or subtractions, and residual accumulation uses FMAs. The kernels use no shared-memory staging. SBD stores the first-pass output in a global buffer of shape $K \times F$.

Memory accesses. Lanes read contiguous feature groups, giving coalesced accesses to the feature matrix. Actual memory traffic depends on cache behaviour.

Synchronisation. Each warp keeps its partial sums in registers and computes its row independently. The kernels use no `__syncthreads()` or programmer-managed shared memory. This avoids block barriers and shared-memory bank conflicts. Ready warps can overlap memory requests, helping to hide memory latency. The two passes execute in order on the same CUDA stream.

Two-pass computation. We compute $\sum_k \alpha_k \mathbf{w}_k \mathbf{q}_k^\top \mathbf{x}$ in two passes. The first pass (`QTX`) assigns one warp per biclique and computes $\mathbf{z}_k = \sum_{j \in \mathbf{q}_k} \mathbf{x}[j]$. The second pass (`WSZ_RX`) assigns one warp per output row and computes

$$\sum_{k:\, i \in \mathbf{w}_k} \alpha_k \mathbf{z}_k + \sum_j R_{ij} \mathbf{x}[j].$$

The intermediate column sum $\mathbf{z}_k$ is computed once and reused by every row in $\mathbf{w}_k$. This amortises column aggregation across rows sharing the same target set.

Performance regimes. The largest gains occur on graphs with many shared-neighbourhood blocks. On weakly compressible graphs, the cost of the two-pass computation can outweigh the benefit of reusing column sums. Table 8 gives the results for all graphs and widths.

D.4 SpMM baselines

Table 8 compares Sputnik [13], dgSPARSE, GE-SpMM [18], SBD_base, CBM, and SBD. SBD_base uses the raw adjacency matrix without bicliques.

D.5 Computational Resources

Hardware platforms. The SpMM experiments use three GPU platforms with a common software stack: PyTorch 2.11.0, Python 3.12 and CUDA 13.0. The SpMM benchmarks are run on each platform independently.

Table 11: Hardware platforms used in this work.

GPU	VRAM	L2 Cache	Host CPU
NVIDIA RTX 3090	24 GB	6MB	Intel Xeon Silver 4116
NVIDIA A40	48 GB	6MB	Intel Xeon Silver 4210
NVIDIA L40S	48 GB	96MB	AMD EPYC 9124

Timing protocol. We report median per-launch latency measured with CUDA events. Both comparisons use ten warm-ups and 50 timed launches. Inputs stay on the GPU. SBD allocates output and intermediate buffers on each call; some baselines preallocate output. Timing excludes input transfer and decomposition.

Decomposition. The SBDFast CPU decomposition uses a pool of **8 worker threads** on all platforms, matching the SLURM allocation (`-cpus-per-task=8`). Decomposition time is excluded from all SpMM figures; it is reported separately in the decomposition tables.

Reproduction settings. Appendix D.2 lists search budgets and timing settings. Total reproduction time depends on hardware and solver stopping behaviour. All experiments can be rerun in several hours.

D.6 Datasets

We evaluate our method on a diverse collection of benchmark graph datasets commonly used in graph representation learning and graph neural network (GNN) research. They cover citation, co-authorship, co-purchase, co-paper, and collaboration graphs.

Citation Networks. The citation network benchmarks include **Cora**, **CiteSeer**, and **PubMed**, which are standard node classification datasets introduced in early graph learning literature [32]. Cora contains scientific publications classified into machine learning topics, CiteSeer consists of scientific articles grouped into research categories, and PubMed contains biomedical publications from the PubMed database. Nodes correspond to documents and edges denote citation relationships.

Co-authorship Networks. We additionally evaluate on the co-authorship datasets **CA-AstroPh** and **CA-HepPh** [23], where nodes represent authors and edges indicate co-authorship relationships. These datasets are derived from collaboration networks in the astrophysics and high-energy physics communities and are widely used for large-scale graph representation benchmarking.

Co-paper Networks. The **coPapersDBLP** and **coPapersCiteseer** datasets [14] represent paper similarity and co-occurrence relationships derived from the DBLP and CiteSeer corpora.

Social Interaction Graphs. Finally, we evaluate on the **COLLAB** dataset from the TUDataset collection [28], which is a scientific collaboration network commonly used for graph classification tasks. Graphs correspond to ego-networks of researchers and edges denote collaborative relationships. We use the disjoint union of all graphs in the dataset. This gives the single adjacency matrix reported in the tables.

All datasets are publicly available and have been extensively adopted in prior graph representation learning studies.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the main contributions: the SBD formulation, the greedy algorithm with theoretical guarantees, and the SpMM acceleration application. All claims are supported by Theorem 1 and the experimental results in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in Section 5.7.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Theorem 1 states its assumptions, and its proof is provided in Appendix A.1. Appendix B gives additional analysis of the SBD objectives and updates. Appendix C studies the effect of λ and provides additional experiments.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 5 and Appendix D.2 describe the experimental setup, hyperparameter ranges and evaluation protocols. Appendix D.5 lists the hardware. Algorithm 1 gives the greedy procedure, and Appendix B.2 derives the AO updates.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The datasets are publicly available. Code and reproduction instructions are available at `https://github.com/IAntoineV/sbd`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 and Appendix D.2 describe the evaluation protocols and hyperparameters. Denoising searches and the SBDFast compression search use 30 Optuna trials. Runtime comparisons use fixed settings, and SpMM experiments reuse the selected decompositions.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Table 1 reports mean F_1 over ten evaluation graphs per noise level. Table 8 reports the mean and sample standard deviation of speedups across three GPUs. Each latency is the median of 50 timed launches after ten warm-up launches. The reported standard deviations describe variation across GPUs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix D.5 details the computer resources used, and the time needed to reproduce the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and find no ethical concerns with this research.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

Justification: This work is foundational research on graph matrix decomposition. We do not foresee direct negative societal impacts from this method.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: This paper does not release pretrained models, generative models, or scraped datasets that pose misuse risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All datasets used are publicly available benchmark datasets. Original papers are cited in the references. Baseline implementations are cited accordingly. Sputnik is licensed under Apache 2.0; dgSPARSE, GE-SpMM, and CBM-GPU use the MIT license. cuSPARSE follows the NVIDIA CUDA Toolkit EULA. SuiteSparse distributes coPapersDBLP, coPapersCiteseer, ca-HepPh, and ca-AstroPh under CC BY 4.0. We could not identify licenses for the remaining datasets.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We introduce the SBD algorithm and a CUDA kernel for SpMM acceleration. Implementation details are provided in Appendix D.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This paper does not involve human subjects research and therefore does not require IRB approval.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs were not used as part of the core methodology. They were used only for writing assistance, code implementation, and do not impact the scientific contributions of this work.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.