Active Positive-Definite Matrix Completion

Charalampos Mavroforakis Dóra Erdös Mark Crovella Evimaria Terzi
{cmav, edori, crovella, evimaria}@cs.bu.edu
Boston University

Abstract

In many applications, e.g., recommender systems and biological data analysis, the datasets of interest are positive definite (PD) matrices. Such matrices are usually similarity matrices, obtained by the multiplication of a matrix of preferences or observations with its transpose. Oftentimes, such real-world matrices are missing many entries and a fundamental data-analysis task, known by the term *PD-matrix completion*, is the inference of these missing entries. In this paper, we introduce the *active* version of PD-matrix completion, in which we assume access to an oracle that, at a given cost, returns the value of an unobserved entry of the PD matrix. In this setting, we consider the following question: "given a fixed budget, which entries should we query so that the completion of the new matrix is much more indicative of the underlying data?". The main contribution of the paper is the formalization of the above question as the ActivePDCompletion problem and the design of novel and effective algorithms for solving it in practice.

1 Introduction

Positive-definite (PD) matrices are widely used in many applications, including recommender systems [12, 20] and analysis of biological data [6, 15, 28] In these applications, the PD matrices are capturing the similarity between objects of interest (e.g., users, items, observations or features) and they are usually the result of the multiplication of a data matrix with its transpose.

Limitations in the real-life observations usually lead to partially-observed PD matrices. For data-analysis purposes, such matrices are usually completed using a matrix-completion method. One of the most common completion methods is the *maximum-determinant completion*; among all the PD matrices that are consistent with the observed entries, this method picks the one that has the maximum determinant. Algorithms for obtaining such completions have been developed since the late 1980's [2, 25]. The maximum-determinant completion is interesting because it makes the *least possible assumptions* about the underlying true matrix; it corresponds to the most general model of the underlying matrix that respects the input observations.

If the number of missing entries is large, the results obtained by such completion methods might be uninformative; the completed matrix is too generic to provide any useful insight about the underlying data. In these cases, more information needs to be collected in order to obtain useful completions. Throughout this paper we assume that this information can be provided by an oracle that, for some given cost, returns the true value of an unobserved entry. In practice, this may entail giving additional incentives for users to provide extra information about their preferences or – in the case of scientific data – running some additional experiments.

Given such oracle access, we study the following problem: Which entries of a partially-observed PD matrix should we query so that we minimize the uncertainty (and thus the determinant) of the maximum-determinant completion of the observed and queried entries? In this paper, we formally define this problem and call it ActivePDCompletion, since it is the *"active"* version of the PD matrix completion.

Even though the completion problem for PD matrices has been widely studied, to the best of our knowledge we are the first to define the active version of the problem and design algorithms for solving it. This problem is also related to other recent work on active matrix completion (for a thorough review of the related work see Section 2), however the constraint that our matrix is PD imposes new algorithmic challenges and solving our problem requires very specific considerations.

Our algorithm for solving the ActivePDCompletion problem has fundamental connections to the graph-theoretic view of PD matrix completion. A key concept that we use throughout the paper is that of the *mask graph* of a partially-observed PD matrix. The mask graph corresponds to the graph formed by using the rows (or columns) of the partially-observed PD matrix as nodes and its observed entries as indicators for the edges. It is known that a partially-observed PD matrix has a PD-completion *iff* the mask graph is chordal [9]. As a result, the algorithm for selecting the entries to query has to not only identify informative

entries, but also to guarantee that the mask graph is chordal and remains chordal after these queries.

At a high level, this task is hard since the level up to which an entry can be "informative" needs to be decided *before* seeing the actual value of the entry. We quantify the potential information capacity of an entry via a principled approach that takes into consideration the range of values of the determinant of a well-specified principal minor that contains the entry of interest.

In terms of computational time, the input and output to our problem is quadratic, imposing the same lower bound on the complexity as that of any completion algorithm. In our case, we provide speedups of our initial algorithm by utilizing an appropriate data structure known in the literature as a *clique tree*. Our extensive experimental evaluation with synthetic as well as real data demonstrate the significant reduction in the determinant of the maximum-determinant completion that we achieve using our algorithm.

2 Related work

To the best of our knowledge we are the first to define and solve the ActivePDCompletion problem. However, both matrix completion and, more recently, active matrix completion problems have been considered in the past. We review some of this work here.

Matrix completion: For the past decades there has been continuous interest in the reconstruction of matrices from a few or noisy entries [4, 10, 11]. Although related, all the above works do not focus specifically on completions of PD matrices and therefore both the applications but also the algorithmic problems that arise there are very different from ours. In our work, we are concerned specifically with the completion of partially-observed positive definite matrices. Positive-definite matrix completion (without queries) has been studied since the 1980's [2, 7, 9], but it is still an active area of research [1, 3, 14, 26] While all of the above works have motivated our own work, they do not tackle the problem of active completion at all, which is the main topic of our paper.

The focus of this paper is on determinant maximizing completions, which arise in many fields. Vandenberghe *et al.* [24] give a survey on determinant-maximizing applications. Even though these works provide a good motivation as of why the maximum determinant is a good measure of the uncertainty of a matrix, they do not necessarily focus on the active version of the maximum-determinant matrix completion problem.

Active matrix completion: It is often assumed that missing entries in a matrix can be queried at some cost. Active matrix completion methods aim at identifying cells of the matrix with missing values that have presumably high "information content". In other words, this assumes that by revealing these entries' content, the quality of the completed matrix would be substantially improved. The notion of querying cells based on information content is studied for example in the context of adaptive sampling for matrix completion in [13, 5]. It is used in many specific applications, such as querying values to improve on a Bayesian method for matrix factorization [22], query pairwise similarities to improve on spectral clustering [21, 27] or even semi-supervised link prediction [17]. One of the recent works to unify a matrix completion approach and a query strategy is by Ruchansky *et al.* [19]. Although these works consider an active version of a matrix completion problem, as we also do, they are quite distinct from ours. First, none of these works considers active completions of PD matrices. Secondly, none of these works focuses on querying missing entries so that the maximum-determinant completion of the resulting matrix is maximized. Both the types of matrices we consider and the objective function we try to optimize raise unique computational problems that have not been addressed before.

3 Background

In this section, we give the necessary notation and background about the matrix completion problem in the class of positive definite (PD) matrices.

Positive-definite matrices: Throughout the paper we assume that there exists a symmetric $n \times n$ true matrix $\mathbf{T}$ that is *positive definite* (PD). Recall that a symmetric square matrix $\mathbf{T}$ is *positive definite* (PD) *iff* for any vector $z \in \mathbb{R}^n$ $z^T\mathbf{T}z > 0$.

Now, assume that $I \subseteq \{1, \ldots, n\}$ is of cardinality k and $\mathcal{I} = I \times I$. Then, we denote by $\mathbf{T}(\mathcal{I})$ the $k \times k$ submatrix of $\mathbf{T}$ with entries in the locations indicated by $\mathcal{I}$. This submatrix is also known as a *principal minor of order* k. A symmetric square matrix $\mathbf{T}$ is PD *iff* all of its principal minors are also positive definite.

A big family of PD matrices are the *similarity* matrices, i.e., the matrices that are a result of the product of a non-singular matrix (i.e., a matrix of preferences or observations) with its transpose. Another case of PD matrices are the *covariance* matrices over linearly independent features. Let us consider a matrix of user preferences over products $\mathbf{A}$ of size $n \times m$, such that $n \ll m$. Such matrices $\mathbf{A}$ usually have *low effective rank*, meaning that they have some singular values that are significantly larger than the rest. Yet, real-life matrices $\mathbf{A}$ never have zero singular values. As a result, the $n \times n$ matrix $\mathbf{T} = \mathbf{A}\mathbf{A}^T$ will not have any zero eigenvalues, which makes it positive definite.

Usually some of the eigenvalues of $\mathbf{T}$ are significantly larger than the rest, but oftentimes even the smaller eigenvalues are far from zero. These are the PD matrices we are interested in completing here.

Partially-observed PD (PPD) matrices: Throughout the paper we assume that not all entries of the PD matrix $\mathbf{T}$ are observed and that we only have access to a subset Ω of its entries. We denote this partially-observed matrix by $\mathbf{T}_\Omega$ and we call Ω the *mask* of $\mathbf{T}$. $\mathbf{T}_\Omega$ has the same values as $\mathbf{T}$ in the cells defined by the pairs in Ω and the rest of its values are unknown. Observe that since we know that $\mathbf{T}$ is PD then we know that it is symmetric and therefore if $(i,j) \in \Omega$, then also $(j,i) \in \Omega$. We also assume throughout that $(i,i) \in \Omega$ for every $i \in \{1,\ldots,n\}$.

We say that $\mathbf{T}_\Omega$ is *partial positive definite* (PPD) if every one of its complete leading principal minors is PD. Since $\mathbf{T}_\Omega$ comes from a PD matrix, we also assume that all the $\mathbf{T}_\Omega$'s in this paper are PPD.

Completability of PPD matrices: A matrix $\widehat{\mathbf{T}}$ is a *PD completion* of $\mathbf{T}_\Omega$ if $\widehat{\mathbf{T}}$ is PD and $\mathbf{T}(i,j) = \widehat{\mathbf{T}}(i,j)$ for every $(i,j) \in \Omega$. We say that $\mathbf{T}_\Omega$ is *PD-completable* if there exists a PD completion for it.

It is a known fact that deciding whether a PPD matrix $\mathbf{T}_\Omega$ is PD-completable can be done in polynomial time [9]. In fact, as Theorem 3.1 indicates, PD-completability is a property of Ω and not of the actual values observed in $\mathbf{T}_\Omega$. In order to discuss this result we introduce the notion of the *mask graph*.

Mask graph: Given a PPD matrix $\mathbf{T}_\Omega$, we define the *mask graph* $G_\Omega(V,E)$ as follows: the set V has n nodes, one for each row/column of $\mathbf{T}$. The edge set E contains edge (i,j) iff $(i,j) \in \Omega$ and $i < j$[1].

Now we are ready to state the following theorem that shows how one can decide in polynomial time whether a PPD matrix is PD-completable.

THEOREM 3.1. (DIRAC [9]) *A PPD matrix $\mathbf{T}_\Omega$ is PD-completable with completion $\widehat{\mathbf{T}}$* iff *the mask graph G_Ω is chordal.*

The power of this theorem is that it translates a problem about completing a PPD matrix into checking the *chordality* of the mask graph. [2] Deciding whether a graph is chordal can be done in polynomial time [23].

Obtaining a PD completion: A key question is the following: given a PD-completable PPD matrix $\mathbf{T}_\Omega$ can we obtain a completion of $\mathbf{T}_\Omega$? The answer to this question is positive. The algorithm itself is the heart of the proof of Theorem 3.1 and since it is central for our paper we describe it in detail below.

First, let us develop some intuition using some ideas by Johnson [9]: Let $\mathbf{H}_x$ be a principal minor in $\mathbf{T}_\Omega$ that is complete except for one of its elements. If we denote this unknown value by x, and by appropriate reordering of the rows and columns of the principal minor, then it looks as follows:

$$\mathbf{H}_x = \begin{pmatrix} a & b^T & x \\ b & \mathbf{A} & c \\ x & c^T & d \end{pmatrix} \tag{3.1}$$

Here, we use the following notation: $\mathbf{H}_x$ is of size $n' \times n'$, $\mathbf{A}$ denotes a completely specified symmetric matrix of size $(n'-2) \times (n'-2)$ and a, b and c refer to known/observed entries of $\mathbf{H}_x$.

Let $\mathbf{B}$ and $\mathbf{C}$ be the largest specified principal minors of $\mathbf{H}_x$. That is,

$$\mathbf{B} = \begin{pmatrix} a & b^T \\ b & \mathbf{A} \end{pmatrix} \quad \mathbf{C} = \begin{pmatrix} \mathbf{A} & c \\ c^T & d \end{pmatrix}$$

Then, the determinant of $\mathbf{H}_x$ can be expressed as:

$$\begin{aligned} \det(\mathbf{H}_x) &= \frac{\det(\mathbf{B})\det(\mathbf{C}) - \left[\det\begin{pmatrix} b^T & x \\ \mathbf{A} & c \end{pmatrix}\right]^2}{\det(\mathbf{A})} \\ &= \frac{\det(\mathbf{B})\det(\mathbf{C})}{\det(\mathbf{A})} - \det(\mathbf{A})(x - b^T\mathbf{A}^{-1}c)^2 \end{aligned} \tag{3.2}$$

We can see that $\det(\mathbf{H}_x)$ is nonnegative if x is in the interval $I_x \subset \mathbb{R}$ defined by the two roots of Equation (3.2). We call I_x the *completion interval* of x. Choosing any value $x \in I_x$ yields a feasible completion.

The above discussion shows us how to complete a principal minor that is missing a single entry. The algorithm for computing a PD-completion of a PPD matrix $\mathbf{T}_\Omega$, which we call `Completion` (Algorithm 1), uses the above intuition. In the first step (line 1) the algorithm finds a suitable ordering of the nodes of G_Ω - which also determines the ordering with which principal minors $\mathbf{H}_x$ are going to be considered -to complete $\mathbf{T}_\Omega$. This ordering is called a *perfect elimination order* (PEO), and we find it using the minimum-efficiency algorithm [18]. The `Completion` algorithm uses the PEO to determine the next missing entry x to be completed. Due to properties of the PEO the addition of the edge that corresponds to the missing entry x to the mask graph will not violate its chordality. Let $\mathbf{H}_x$ be the maximal principal minor corresponding to x of the form shown in Equation (3.1). `Completion` samples the value for x at random from the completion interval

[1] This means that we omit any self-loop edges from E and do not consider edge multiplicities.

[2] Recall that a graph is *chordal* if every one of its cycles of length 4 or larger has at least one chord, i.e., an edge that connects two non-adjacent nodes of the cycle.

Algorithm 1 The `Completion` algorithm.

Input: PPD matrix $\mathbf{T}_\Omega$ with observed entries Ω
Output: PD matrix $\widehat{\mathbf{T}}$

1: $\alpha \leftarrow$ PEO of V from $G_\Omega = (V, E)$
2: $\widehat{\mathbf{T}} \leftarrow \mathbf{T}_\Omega$
3: **while** G_Ω is not a complete graph **do**
4: $\quad k \leftarrow \max\{i \mid \exists v \in V,\ \{v, \alpha_i\} \notin E\}$
5: $\quad r \leftarrow \max\{i \mid \{\alpha_i, \alpha_k\} \notin E\}$
6: $\quad \mathbf{H}_x \leftarrow$ largest complete principal minor of $\widehat{\mathbf{T}}$ containing element (α_r, α_k)
7: $\quad \widehat{\mathbf{T}}(\alpha_r, \alpha_k) = \widehat{\mathbf{T}}(\alpha_k, \alpha_r) = x \in I_x$.
8: $\quad E = E \cup \{\alpha_r, \alpha_k\}$

return: $\widehat{\mathbf{T}}$

I_x. Observe that this results in one possible completion among potentially infinite ones.

Maximum-determinant completions: Although a PD-completable PPD matrix can be completed in polynomial time, there are potentially infinite many such completions. The most widely used method for ranking these completions is using the *determinant* of the completed matrix $\widehat{\mathbf{T}}$, denoted by $\det(\widehat{\mathbf{T}})$. Thus, we are trying to find the matrix with the *maximum determinant*. In terms of notation, if $\mathcal{T}_\Omega$ are all possible PD-completions of $\mathbf{T}_\Omega$, we use $\mathbf{T}^*_\Omega$ to denote the completion in $\mathcal{T}_\Omega$ with the maximum determinant.

Finding the maximum-determinant completion has been a common objective for PD-matrix completion [2, 7]. This is an intuitive objective for the following reason: the eigenvectors of the PD-matrix define a base and each eigenvector is scaled by the corresponding eigenvalue such that all data points are enclosed by the hyper-rectangle defined by the scaled eigenvectors. Since the determinant of a matrix is the product of its eigenvalues, the maximum-determinant completion aims to find the matrix that has the maximum-volume hyper-rectangle. Thus, the maximum-determinant completion is the *most general* PD matrix that agrees with the PPD matrix in the observed entries.

Finding the maximum determinant completion: The maximum determinant completion of a PD-completable PPD matrix can be found in $O(n^2 O_{\text{INV}})$ where O_{INV} is the time required for a matrix inversion – see Theorem 6.1 of [2]. In fact the algorithm for finding the maximum-determinant completion of a PPD matrix is the `Completion` with the difference that in line 7 we now have to pick $x \in I_x$ such that $\det(\mathbf{H}_x)$, given in Equation (3.2), is maximized.

In fact, Theorem 6.1 of [2] proves that these local maximizations of the determinant will lead to the *unique global determinant-maximizing solution*. We call this version of the `Completion` the `MaxDetCompletion` algorithm.

4 Active PD-completion

In this section, we define the ActivePDCompletion problem. Our key assumption is that the entries of $\mathbf{T}$ which are not observed in $\mathbf{T}_\Omega$ can be queried at some cost. Formally, we define the problem as follows.

Problem 1. (ActivePDCompletion) *Given a PPD matrix $\mathbf{T}_\Omega$ with observed entries Ω and budget k, identify a set Q of k entries from the underlying true matrix $\mathbf{T}$ so that, once Q is revealed,* $\det\left(\mathbf{T}^*_{\Omega\cup Q}\right)$ *is minimized.*

The intuition behind this problem formulation is that some of the unobserved entries of $\mathbf{T}$ carry high information content. If we then choose to query such entries and enhance the set of initial observations Ω accordingly, the determinant of the maximum-determinant completion $\mathbf{T}_{\Omega\cup Q}$ will be considerably less than that of the completion of $\mathbf{T}_\Omega$, since the solution space for completions $\widehat{\mathbf{T}}$ is going to be smaller. Thus, in this case $\widehat{\mathbf{T}}$ will be closer to the true matrix $\mathbf{T}$.

Discussion: In this definition we assumed that the cost of querying any unknown entry of $\mathbf{T}$ is constant. However, our algorithms can take into account varying querying prices for different entries.

The main difficulty of the problem comes from the fact that we need to decide which entries to query *before* actually knowing their value.

Note also that this problem definition assumes that the mask graph of the input matrix $\mathbf{T}_\Omega$ is chordal. If this is not the case, then we first need to make the mask graph chordal (by using $k' \leq k$ queries to reveal the unknown values) and then solve the ActivePDCompletion problem on the chordal graph with a budget of $k - k'$. Transforming a non-chordal graph into chordal with the minimum number of edge additions is known as the minimum fill-in problem [29], for which known heuristics exist [23].

5 Algorithms

In this section, we describe our algorithms for the ActivePDCompletion problem. The common setup in our algorithms is that we query entries from $\mathbf{T}$ one-at-a-time. The entry (i, j), that is to be queried in the next iteration, is one that once it is included in the mask graph it maintains the graph's chordality. At the same time, among all the edges with this property we pick the one with the highest *score*. The idea behind the score of an entry is that edges with high score correspond to entries with high uncertainty. Hence, querying a high score location in the PPD matrix reduces our uncertainty about the hidden matrix the most.

The `Select` algorithm: We first give a high-level description of our basic query strategy in `Select` Algorithm 2. Then we discuss the technicalities of the computationally-challenging tasks it performs. `Select` first builds a data structure (line 2), a so called *clique tree*, that enables us to find candidate entries (in the `FindCandidates` function in line 4) which can be queried without violating the chordality of the mask graph. Next, we compute the *score* of each candidate (line 5). Finally we add the candidate with the highest score to the query set (line 7) and update the data structure accordingly (line 10).

Algorithm 2 The `Select` algorithm.

Input: PPD matrix $\mathbf{T}_\Omega$ with observed entries Ω, mask graph $G_\Omega = (V, E)$ and budget k.
Output: PPD matrix $\mathbf{T}_{\Omega \cup Q}$ which is PD-completable.

1: $Q = \emptyset$
2: $C = \texttt{CliqueTree}(G_\Omega)$
3: **for** $i = 1, \ldots, k$ **do**
4: $\quad S = \texttt{FindCandidates}(G_{\Omega \cup Q}, C)$
5: $\quad$ **for** $e \in S$ **do**
6: $\quad\quad$ compute *score*(e)
7: $\quad e^*(i,j) = \arg\max \{\textit{score}(e) \mid e \in S\}$
8: $\quad Q = Q \cup (i,j) \cup (j,i)$
9: $\quad E = E \cup e^*$
10: $\quad$ Update C by inserting (i,j)
return $\mathbf{T}_{\Omega \cup Q}$

The function `FindCandidates` (line 4) in `Select` enumerates all the edges that can be inserted to the current mask graph G_Ω while maintaining its chordal structure. To do that, we make use of the *clique tree* data structure as introduced by Ibarra [8]. In the supplementary material, we provide more information about this data structure. Finally, after an edge insertion (query), we update the clique tree in $O(n)$.

Scoring functions: A key step of our method is the computation of *score*(e) for every edge $e(v, v') \in E$. We compute this score as follows: for $e(v, v') = x$ assume that $\mathbf{H}_x$ is the maximal leading minor that contains x and it has the form given in Equation (3.2). Then, we compute the *score*(e) as follows:

$$\textit{score}(e) = \frac{\int_{x \in I_x} \det(\mathbf{H}_x)}{\det(\mathbf{A})}, \tag{5.3}$$

where I_x is the completion interval of x given in the maximum leading minor $\mathbf{H}_x$.

Intuitively, this computation isolates the expected impact of the missing value of x in the maximum-determinant completion of the input PPD matrix. The larger the value of *score*(e), the larger the contribution of x in the maximum-determinant completion and thus the larger the uncertainty of entry x; alternatively, knowing x is expected to significantly reduce the determinant of the maximum determinant completion. Therefore, large values of *score*(e), indicate entries that need to be queried.

Algorithm 3 `FindCandidates`: generation of all insertable edges of a graph G.

Input: Mask graph $G = (V_G, E_G)$, clique tree structure $C = (V_C, E_C)$
Output: Set of insertable edges S

1: $S = \emptyset$
2: **for** $v \in V_G$ **do**
3: $\quad$ Let $c \in V_C$ be a clique containing v
4: $\quad$ **for** $c_i \in \texttt{DFS}(C)$ starting from c **do**
5: $\quad\quad$ **for** $v_i \in$ clique c_i **do**
6: $\quad\quad\quad$ **if** (v, v_i) is insertable in G **then**
7: $\quad\quad\quad\quad S = S \cup \{(v, v_i)\}$
return S

Our comparison between the above-defined and other candidate scoring functions (presented in the supplementary material) indicate that the *score* defined in Equation (5.3) outperforms all scoring schemes we considered

Complexity: The running time of `Select` is

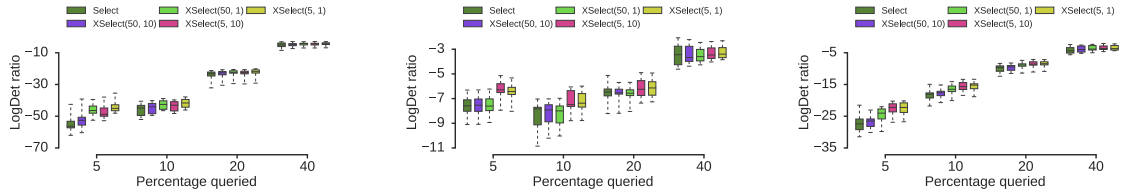
$$O(\underbrace{kn^3}_{\text{candidate generation}} + \underbrace{kn^2 O_{\text{INV}}}_{\text{score computation}} + \underbrace{kn}_{\text{cliquetree update}}) \tag{5.4}$$

where k is the given budget and O_{INV} is the time required to invert an $n \times n$ matrix.

The `XSelect` algorithm: The complexity of `Select` is a strict limit for its practical applicability – particularly as the size of the input matrix grows. For this reason, we revisit our candidate generation algorithm and make a simple observation; instead of retrieving all possible insertable edges S, we could limit its cardinality $|S|$ to be constant. Whenever the size of S grows larger than β, the algorithm will return. Additionally, instead of repeating it n times (one traversal for each node), we can sample a set of *seed* nodes (line 2) for the depth-first-search of size α. We will call this faster version of our query algorithm $\texttt{XSelect}(\alpha, \beta)$. After the above modifications, the time complexity of $\texttt{XSelect}(\alpha, \beta)$ becomes $O(k \min(\alpha n^2, \beta n) + k\beta O_{\text{INV}} + kn)$.

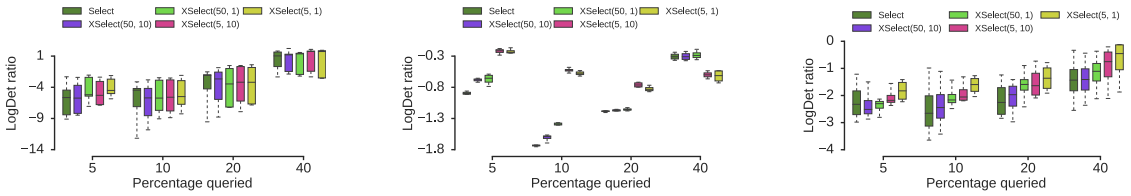
6 Experiments

The aim of this section is twofold; it showcases the performance and efficiency of `Select` compared to a clever random baseline and it investigates the trade-off



(a) Random hiding (b) Diagonal hiding (c) Block hiding

Figure 1: Budget experiment: The performance of our methods compared to the baseline for different budget percentages on synthetic data. Only 25% of the entries of the matrix are observed.



(a) Random hiding (b) Diagonal hiding (c) Block hiding

Figure 2: Budget experiment: The performance of our methods for different budget percentages on the `proteins` dataset. Only 25% of the entries of the matrix are observed.

between speed and performance when using `XSelect`. We make all our code and data available online[3].

6.1 Datasets We run our algorithms on both synthetic and real-world matrices. This way, we achieve a deeper and more systematic understanding of the performance of our methods, and at the same time showcase their applicability in real-world situations.

Synthetic data: We generate synthetic matrices that are symmetric and positive definite, with some block-like structure in the following way:

(i) Pick the size of the matrix n, a number of clusters c, a number of features k.

(ii) Generate c clouds of points in a k-dimensional space, each with $n_1, n_2, \ldots n_c$ members such that $n_1 + n_2 + \ldots + n_c = n$. Denote this $n \times k$ matrix as D.

(iii) Compute $D' = DD^T + R$, where $R \sim \text{Uniform}(1)^{n \times n}$ is a noise matrix.

(iv) Return $\mathbf{T} = D'D'^T$.

In our experiments we chose $n = 100$, $c = 5$, $k = 5$, but the results are consistent for all other choices.

Real data: In order to experiment with real data we need to generate symmetric PD matrices from matrices that encode observations of data points with respect to different features. We do this in the following way:

(i) Start with a real matrix D of size $n \times k$.

(ii) If $n \leq k$, return $\mathbf{T} = DD^T$, else set $\mathbf{T} = D^T D$.

The resulting matrices are known either as *similarity* or *covariance* matrices. We apply this procedure to the ***proteins*** dataset. This is a 1080×77 matrix describing the the expression level of various proteins in the cerebral cortex of mice [16]. We disregard any rows with missing data and compute the 77×77 covariance matrix of the expression levels.

6.2 Generating partial matrices We experiment with three different ways to hide h elements from a given matrix $\mathbf{T}$ of size $n \times n$; *random*, *diagonal* and *block* hiding. All of these methods take as argument the number of cells whose values are hidden.

Random hiding: This method is generating a partial matrix $\mathbf{T}_\Omega$ with $|\Omega| = n^2 - h$ observed entries. The choice of entries to hide is made randomly, while respecting the constraint that the mask graph G_Ω. In order to do that, we start with the full mask graph G_{Ω_0}, with $\Omega_0 = E$ meaning that all the cells of $\mathbf{T}$ are observed, we generate the clique tree of G_{Ω_0} as described in [8] and sequentially remove h edges from that graph. The resulting graph, i.e. G_{Ω_h}, corresponds to the partial matrix $\mathbf{T}_{\Omega_h}$ that we aim for.

Diagonal hiding: This method only keeps observations that belong to the ℓ first diagonals. Given h, we find the number of diagonals ℓ that we need to observe

[3] `http://cs-people.bu.edu/cmav/active_complete`

so as to keep the number of observed elements as close to $n^2 - h$ as possible.

Block hiding: This hiding method works as follows: we split the matrix $\mathbf{T}$ into b blocks and will observe only the elements of the blocks that intersect the diagonal. To make sure that we can control the number of observed elements, we formulate the problem of picking the block sizes as a quadratic integer program (QIP)

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{i=1}^{b} x_i^2 \quad - (n^2 - h) \\ \text{s.t} \quad & \sum_{i=1}^{b} x_i = n, \quad \text{and} \\ & x_i \geq 2. \end{aligned} \tag{6.5}$$

The result is the hiding pattern we use.

6.3 Baseline algorithm Since there is no existing work on the ActivePDCompletion, the baseline that we compare against is the random query algorithm, denoted by `RandComplete`. At each iteration, this method uses a clique tree [8] to track the edges that are insertable in the mask graph without violating its chordality. It randomly picks one of these edges, executes the query and fills in the corresponding cell in the matrix. Finally, it updates the clique tree and repeats the previous steps for as long as there is remaining budget. Essentially, it works the same way as Algorithm 2, with the main difference being that in lines 5–7 the query is selected from S at random.

Note that this baseline is a "clever" random baseline as it guarantees that the queried entries maintain the chordal property of the mask graph.

6.4 Evaluating the efficacy of our method As a first step, we compare the performance of `Select` and `XSelect` with `RandComplete`. For the comparison we proceed as follows: let Q_x be the entries queried by algorithm $x \in \{\texttt{Select}, \texttt{XSelect}\}$ and Q_r be the entries queried by `RandComplete`. Then, we define $\mathbf{X} = \mathbf{T}^*_{\Omega \cup Q_x}$ and $\mathbf{R} = \mathbf{T}^*_{\Omega \cup Q_r}$ to be the maximum-determinant completions of PPD matrices $\mathbf{T}_{\Omega \cup Q_x}$ and $\mathbf{T}_{\Omega \cup Q_r}$ respectively. We evaluate the performance of x by computing the following measure:

$$\text{LogDetRatio}(x) = \log \frac{\det \mathbf{X}}{\det \mathbf{R}}. \tag{6.6}$$

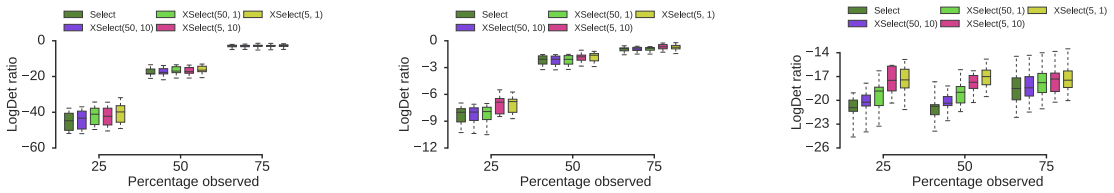
We call this measure the *log-determinant ratio* of x. The smaller the value of the log-determinant ratio the better the performance of x; small values of LogDetRatio(x) indicate that the numerator is much smaller than the denominator and thus the maximum-determinant completion achieved after querying using x is much smaller than the one achieved by `RandComplete`. Note that we take the logarithms of the determinants because the values of the determinants in our experiments are very large. This way we compare the performance with respect to the *order of magnitude* improvement.

Budget experiment: First, we explore the performance of the algorithms for different hiding schemes as we change the number of entries we can query. In all experiments we start with a matrix where only 25% of the total entries are known. The results across the different datasets are summarized in Figures 1 (synthetic) and 2 (`proteins`). In the x-axis of the panels we show the percentage of queried entries and on the y-axis we report the LogDetRatio for `Select` and `XSelect`. In these plots, we report the results of `XSelect` for $\alpha \in \{5, 50\}$ and $\beta \in \{1\%, 10\%\}$. Experimenting with more values demonstrated that the trends we observe do not change. Notice that α is in absolute numbers, while β is relative to the number of edges (hidden elements that are candidates for querying). Since each experiment is repeated at least 10 times, we use boxplots to display the performance of each algorithm. The displayed boxplots have information about the *min*, the 1st quartile, the median the 3rd quartile and the *max* of the LogDetRatio value achieved over different repetitions.

For both the synthetic and the real dataset across all panels, we observe that the values of LogDetRatio are significantly small and therefore `Select` and `XSelect` show significant improvement over `RandComplete`, independently of the hiding pattern.

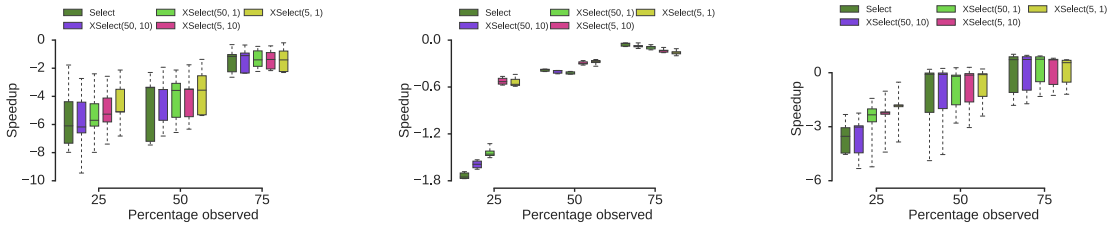
The performance of all algorithms for large number of queries is the same. Notably, the sampling algorithms do not suffer a significant drop in performance compared to `Select`. Indeed, `XSelect`$(5, 1)$ is only marginally worse than `Select`, and comes with an impressive speed-up in the execution time.

Hiding experiment: Here, we test the effect of the number of observations in the input in the performance of our algorithms. For this experiment, we fix the number of queries to be 10% of the elements of the matrix. We run our evaluations on matrices where only 25%, 50%, or 75% of the entries are observed and the rest are missing. The results of these evaluations are summarized in Figures 3 (synthetic) and 4 (`proteins`). In these plots, the x-axis corresponds to the percentage of observed entries and the y-axis corresponds to the LogDetRatio of `Select` and `XSelect`. Again, we see the same pattern as before; all of the methods are significantly better than the baseline, and the sampling algorithms only suffer a minimal performance drop compared to `Select`. The trend in the results shows that, as expected, if only a few elements are missing, there is not much that a smart query method can do.



(a) Random hiding (b) Diagonal hiding (c) Block hiding

Figure 3: Hiding experiment: An evaluation of our algorithms on synthetic data when varying the percentage of the observed entries. The query budget is 10% of the total number of entries.



(a) Random hiding (b) Diagonal hiding (c) Block hiding

Figure 4: Hiding experiment: Evaluating our algorithms on the `proteins` dataset for different percentage of observed entries. The query budget is 10% of the total number of entries.

6.5 Measuring the speedup achieved by `XSelect`

Here, we investigate in depth the gain in terms of execution time that we achieve by using `XSelect`. More specifically, we focus on the experiments that we ran for the budget experiment previously, and show the speedup of the different instantiations of the `XSelect` algorithm when compared to `Select`. The results are summarized in Figures 5 and 6; the x-axis on the plots shows the percentage of queried entries and the y-axis shows the speedup achieved by the different versions of `XSelect` when compared with `Select`; the speedup is computed as the ratio of the execution time of the latter over the execution time of the former. Depending on the hiding pattern and the budget, we can get up to almost 18x speedup when using `XSelect` instead of `Select`. It is worth mentioning that α appears to be very critical for the execution time; smaller values lead to much faster performance.

7 Conclusions

The key contribution of our work is the introduction of the concept of active PD-matrix completion, the formalization of the ActivePDCompletion problem and the design of effective querying strategies. Our experiments with both synthetic and real datasets demonstrate the effectiveness of these querying strategies: the completed matrices obtained after incorporating the queried entries are much more meaningful – i.e., the determinant of their maximum-determinant completion is significantly smaller than the corresponding determinant obtained with the original observations. These results demonstrate the advantage of incorporating an interactive step between the data collection and the data analysis parts of the data mining pipeline.

References

[1] C.-G. Ambrozie. Finding positive matrices subject to linear restrictions. *Linear Algebra and its Applications*, 2007.

[2] W. W. Barrett, C. R. Johnson, and M. Lundquist. Determinantal formulae for matrix completions associated with chordal graphs. *Linear Algebra and its Applications*, 1989.

[3] E. Ben-David and B. Rajaratnam. Positive definite completion problems for bayesian networks. *SIAM J. on Matrix Analysis and Applications*, 2012.

[4] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 2009.

[5] S. Chakraborty, J. Zhou, V. N. Balasubramanian, S. Panchanathan, I. Davidson, and J. Ye. Active matrix completion. In *IEEE ICDM*, 2013.

[6] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Muller. Identifying functional modules in protein protein interaction networks: an integrated exact approach. *Bioinformatics*, 2008.

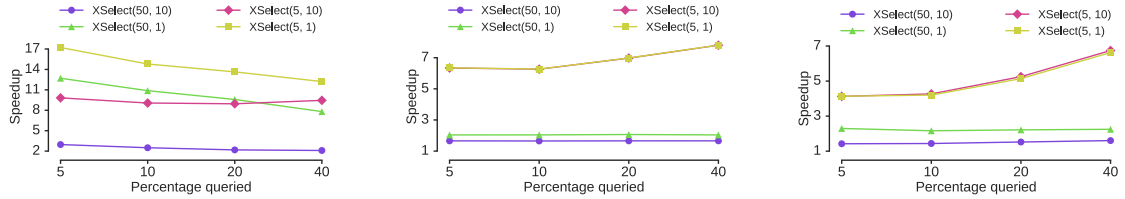


Figure 5: An analysis of the speed-up that `XSelect` achieves when compared to `Select`. The comparison is on synthetic data and only 25% of the entries observed.

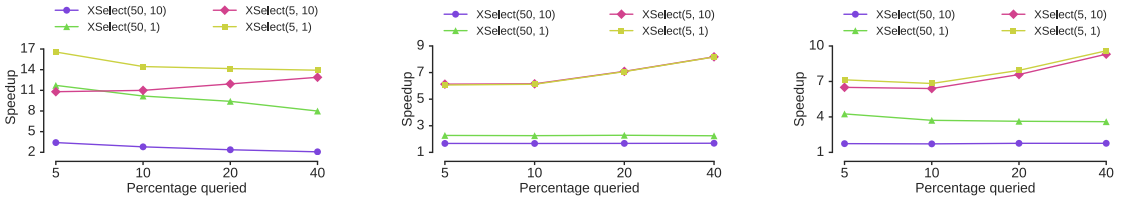


Figure 6: An analysis of the speed-up that `XSelect` achieves when compared to `Select`. The comparison is for the `proteins` dataset with 25% of the entries observed.

[7] R. Grone, C. R. Johnson, E. M. Sá, and H. Wolkowicz. Positive definite completions of partial hermitian matrices. *Linear algebra and its applications*, 1984.

[8] L. Ibarra. Fully dynamic algorithms for chordal graphs and split graphs. *ACM Transactions on Algorithms (TALG)*, 2008.

[9] C. R. Johnson. Matrix completion problems: a survey. *Proc. Symposia in Appl. Math.*, 1990.

[10] R. H. Keshavan, A. Montanari, and S. Oh. Matrix completion from a few entries. *IEEE Transactions on Information Theory*, 2010.

[11] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE TPAMI*, 2004.

[12] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *TKDD*, 2010.

[13] A. Krishnamurthy and A. Singh. Low-rank matrix and tensor completion via adaptive sampling. In *NIPS*, 2013.

[14] M. Laurent. *Encyclopedia of Optimization*, chapter Matrix completion problems. 2001.

[15] B. Lehne and T. Schlitt. Protein-protein interaction databases: keeping up with growing interactomes. *Human Genomics*, 2009.

[16] M. Lichman. UCI machine learning repository, 2013.

[17] R. Raymond and H. Kashima. Fast and scalable algorithms for semi-supervised link prediction on static and dynamic graphs. In *ECML/PKDD*, 2010.

[18] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. *Graph theory and computing*, 1972.

[19] N. Ruchansky, M. Crovella, and E. Terzi. Matrix completion with queries. In *ACM SIGKDD*, 2015.

[20] E. Scheinerman and K. Tucker. Modeling graphs using dot product representations. *Computational Statistics*, 2010.

[21] O. Shamir and N. Tishby. Spectral clustering on a budget. In *AISTATS*, 2011.

[22] J. G. Silva and L. Carin. Active learning for online bayesian matrix factorization. In *ACM SIGKDD*, 2012.

[23] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 1984.

[24] L. Vandenberghe, S. Boyd, and S. po Wu. Determinant maximization with linear matrix inequality constraints. *SIAM J. on Matrix Analysis and Applications*, 1998.

[25] L. Vandenberghe, S. Boyd, and S.-P. Wu. Determinant maximization with linear matrix inequality constraints. *SIAM J. on matrix analysis and applications*, 1998.

[26] C. Wang, D. Sun, and K. Toh. Solving log-determinant optimization problems by a newton-cg primal proximal point algorithm. *SIAM J. on Optimization*, 2010.

[27] F. L. Wauthier, N. Jojic, and M. I. Jordan. Active spectral clustering via iterative uncertainty reduction. In *ACM SIGKDD*, 2012.

[28] Z. Xia, L.-Y. Wu, X. Zhou, and S. T. Wong. Semi-supervised drug-protein interaction prediction from heterogeneous biological spaces. *BMC Systems Biology*, 2010.

[29] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM J. on Algebraic and Discrete Methods*, 1981.

Active Positive-Definite Matrix Completion
Supplementary Material

Charalampos Mavroforakis Dóra Erdös Mark Crovella Evimaria Terzi
{cmav, edori, crovella, evimaria}@cs.bu.edu
Boston University

1 The `FindCandidates` function of Algorithm 3

In the `FindCandidates` function (line 4), `Select` finds all the single edges that can be added to the current mask graph G_Ω while maintaining its chordal structure. To do that, we make use of the *clique tree* data structure as introduced by Ibarra [1]. Given a graph $G = (V, E)$, the clique tree is a tree $C = (V_C, E_C)$, in which each node is a maximal clique of G, i.e., $V_C \subset 2^V$. In our case the number of nodes $|V_C|$ is $O(n)$ because G is chordal [1]. Two cliques are connected in the clique tree if they share common nodes. According to Ibarra, retrieving the insertable edges $e = (i, \cdot)$, i.e., those that are attached to node $i \in V$, requires running a depth-first search on C (line 4). For every clique $c \in V_C$ that we visit and for each node $j \in c$, we check if the edge (i, j) is insertable. As a result, it takes $O(n^3)$ time to retrieve the set of all insertable edges. The pseudocode of this procedure is summarized in Algorithm 3.

2 Exploring different score functions

In all the experiments that we reported above, we run our algorithms using the edge scoring scheme given by function *score*, which we defined in Equation (5.3). In this experiment, we explore the performance of three other scoring schemes and show that indeed using *score* was our best option.

More specifically, we consider the following three scoring alternatives, which in turn define variations of `Select`.

data_score: We compute the *data_score* as follows: for $e(v, v') = x$ assume that $\mathbf{H}_x$ is the maximal leading minor that contains x and it has the form given in Equation (3.2). Then, we define *data_score*(e) to be the value

$$\frac{1}{\det(\mathbf{A})|\{x \in Z : x \in I_x\}|} \sum_{x \in Z : x \in I_x} \det(\mathbf{H}_x),$$

where Z is the set of observations in $\mathbf{H}_x$.

That is, for the calculation of the "expected" determinant, instead of using the integral over all values of $x \in I_x$ (as in Equation (5.3)), we only use the values which appear in the matrix and are observations from the ground truth.

norm_score: In the *norm_score*, we normalize the "expected" determinant of $\mathbf{H}$ by its maximum value. Using the same notation as above the *norm_score* is defined as follows:

$$norm\_score(e) = \frac{1}{\det(\mathbf{A})} \frac{1}{\max\limits_{x \in I_x} \det(\mathbf{H}_x)} \int_{x \in I_x} \det(\mathbf{H}_x)$$

det_score: Finally, in the *det_score* we remove the normalization by $\det(\mathbf{A})$; in this scoring scheme the edge that maximizes *det_score* is the edge that maximizes the expected determinant of the its corresponding maximal leading minor. That is, using the same notation as above *det_score* is defined as follows:

$$det\_score(e) = \int_{x \in I_x} \det(\mathbf{H}_x).$$

We run `Select` with these different score variants as well as the original score function, defined as *score* in Equation. (5.3). We also run for all the data the `RandComplete` algorithm and we report the LogDetRatio of the different versions of `Select`. For this experiment we use the synthetic partial matrices generated by random hiding. As in the budget experiment, we only reveal 25% of the original matrix as our observed entries.

Figure 1 summarizes our findings; the x-axis corresponds to the query budget, while the y-axis reports the LogDetRatio for *score*, *data_score*, *norm_score* and *det_score*. The dashed line corresponds to the LogDetRatio of an algorithm that has the same performance as `RandComplete`, i.e., our baseline. Anything above this line corresponds to an algorithm that performs worse than the baseline and anything below the dashed line corresponds to an algorithm that performs better than `RandComplete`. Clearly, using the *score* scoring with `Select` performs consistently the best. Almost as good is the performance of *data_score*. This is somehow expected as the *score* and *data_score* are defined using the same rationale in mind: they try to see

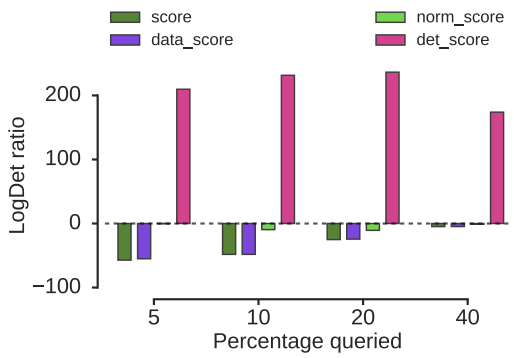


Figure 1: A performance analysis for different choices of scoring functions. Values above the dashed line imply a performance worse than our random baseline.

the impact of the value of x in $\det(\mathbf{H}_x)$ by subtracting the value of $\det(\mathbf{A})$, which consists of entries independent of x. The only difference between these two scoring schemes is that *score* computes the expected value of $\det(\mathbf{H}_x)$ using all possible values in I_x, while *data_score* computes the same value using the entries that appear in $\mathbf{H}_x$ and are observations from the ground truth.

The results also showcase that *norm_score* is not as good as *score* or *data_score*, as it performs almost as good as the random baseline. Finally, *det_score* is significantly worse than all other scoring schemes, including the random baseline. We conjecture that the reason for this is that *det_score* does not isolate the impact of x in the value of $\det(\mathbf{H}_x)$, as it does not subtract the impact of $\mathbf{A}$. As a result, the values appearing in $\mathbf{A}$ may dominate the *det_score*, preventing this version of the algorithm from really evaluating x itself.

References

[1] L. Ibarra. Fully dynamic algorithms for chordal graphs and split graphs. *ACM Transactions on Algorithms (TALG)*, 2008.