Data Mining and Knowledge Discovery manuscript No.
(will be inserted by the editor)

Discovering Bands from Graphs

Nikolaj Tatti

Abstract Discovering the underlying structure of a given graph is one of the fundamental goals in graph mining. Given a graph, we can often order vertices in a way that neighboring vertices have a higher probability of being connected to each other. This implies that the edges form a band around the diagonal in the adjacency matrix. Such structure may rise for example if the graph was created over time: each vertex had an active time interval during which the vertex was connected with other active vertices.

The goal of this paper is to model this phenomenon. To this end, we formulate an optimization problem: given a graph and an integer K, we want to order graph vertices and partition the ordered adjacency matrix into K bands such that bands closer to the diagonal are more dense. We measure the goodness of a segmentation using the log-likelihood of a log-linear model, a flexible family of distributions containing many standard distributions. We divide the problem into two subproblems: finding the order and finding the bands. We show that discovering bands can be done in polynomial time with isotonic regression, and we also introduce a heuristic iterative approach. For discovering the order we use Fiedler order accompanied with a simple combinatorial refinement. We demonstrate empirically that our heuristic works well in practice.

Categories and Subject Descriptors

H.2.8 [**Database management**]: Database applications–*Data mining*

Keywords monotonic segmentation, log-linear models, isotonic regression, Fiedler order, bands

N. Tatti
HIIT, Department of Information and Computer Science, Aalto University, Finland, and
Department of Computer Science, KU Leuven, Leuven, Belgium
E-mail: nikolaj.tatti@aalto.fi

1 Introduction

Consider a dataset given in Figure 1(a). This data contains 139 species discovered at 501 sites (Fortelius, 2005). As different species live in different eras, the dataset can be sorted[1] such that the data points form a band. Let us construct a similarity matrix between the sorted species, where the weight between two species is the number of sites. Since a large number of the species-pairs will have do not share a single site, it is beneficial to view the matrix as a weighted graph. We see from the graph, given in Figure 1(b), that most of the edges will be located close to the diagonal, forming a band.

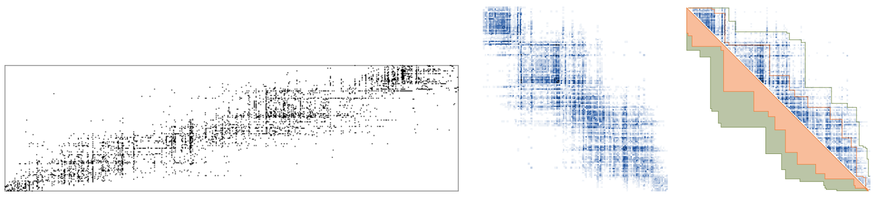

(a) Data matrix (b) Similarity graph (c) Discovered bands

Fig. 1 139 different species discovered from 501 different paleontological sites

The phenomenon of having edges near the diagonal is not uncommon. For example, assume that the graph was constructed over time and that each vertex had an active time interval during which it connected to other active vertices with a higher probability. In such case, we should be able to arrange the vertices such that the edges are concentrated around the diagonal. As another example consider a graph with a(n overlapping) clustering structure. If we can rearrange clusters such that only neighboring clusters have significant overlap, if any, then there exists a vertex order such that the edges are close to the diagonal.

The goal of this paper is to quantify this phenomenon, see Figure 1(c) for an example. More formally, we introduce the following optimization problem. Given a graph and an integer K, order vertices and segment the adjacency matrix into K bands such that a) each band respects the vertex order—when drawn the boundary of the band must either move down or right, b) the edge density in inner bands is higher than the density in the outer bands, and c) segments are as homogeneous as possible, according to some score function. Note that these bands may have varying thickness as illustrated in Figure 1(c).

In order to score the bands, we model the graph as a mixture model, where each band is a Erdős-Rényi model (Erdos and Renyi, 1960). Our goal is to find bands that optimize the likelihood of this model. As an application, discovered bands can be used for determining communities for individual vertices: if (u, v) belongs to the kth band, then we say that v belongs to the kth community of u, small ks corresponding to the inner circles.

[1] Here, we sorted the data using the Fiedler order, see Section 5.

We break this optimization problem into two natural subproblems. The first problem is to find optimal K bands given the order and the second problem is to find a good order. We approach the latter problem by using Fiedler order (Fiedler, 1975) accompanied with a simple greedy refinement heuristic.

Most of this work is devoted into solving the first subproblem which happens to have a polynomial solution. In fact, this problem resembles a monotonic segmentation problem (see Haiminen and Gionis, 2004), however, it is much more intricate and there is no obvious technique for solving such problem. We will show that for certain scores, we do not have to consider all possible segmentations. We introduce a concept of *borders*. Roughly speaking, a border divides the adjacency matrix into two parts such that the inner part has a higher average. We will show that the optimal segmentation can be constructed from the borders. This allows us to transform the original optimization problem into two separate problems. First we need to discover all the borders, secondly we need to select the optimal segmentation using borders as candidates. Surprisingly, the second subproblem turns out to be an instance of the sequence segmentation problem, and can be solved using a standard dynamic program given by Bellman (1961).

We consider two techniques for discovering borders. The first approach is based on isotonic regression (Spouge et al., 2003), and gives us an exact solution. Despite being a polynomial-time solution, this approach requires that the graph is stored in a full form. Hence, we also present an iterative heuristic technique that uses sparsity of the graph to its advantage.

The rest of the paper is organized as follows. We introduce preliminaries and formally state our problem in Section 2. We introduce the concept of borders in Section 3 and present algorithms for discovering borders in Section 4. We consider discovering orders in Section 5. We present related work in Section 6 and experimental evaluation in Section 7. Finally, we present our conclusions in Section 8.

2 Preliminaries and Problem Statement

In this section we present our notation and give the formal problem statement. We first introduce the optimization problem for graphs and then cast this problem into a more general setup.

2.1 Discovering Bands from Graphs

Our first task is to define formally what we mean by a band. In order to do this, assume an undirected graph $H = (V, F)$. If we are given an order o on vertices, essentially a mapping $o : 1, \ldots, |V| \to V$, we say that H respects the order if the neighborhood of each vertex can be seen as a segment w.r.t. the order, that is, for every $v \in V$, there exist integers s and e such that $\{u \in V \mid (v, u) \in F\} \cup \{v\} = \{o(i) \mid s \leq i \leq e\}$. If we order the vertices according to o, then H will have all its edges next to diagonal. Our goal is given a graph G, find o and H optimizing a certain score.

Let us now define the score that we wish to optimize. Assume that we are given a graph $G = (V, E)$. Let $X \subseteq V \times V$ be a subset of vertex pairs. Let us define

$$s(X) = |X \cap E| \log a(X) + |X \setminus E| \log(1 - a(X)), \quad \text{where} \quad a(X) = \frac{|X \cap E|}{|X|},$$

which is a maximum log-likelihood of a Bernoulli variable. We can now formulate the optimization problem.

Problem 1 (2-band discovery) Given a graph $G = (V, E)$ find an order o on vertices and a graph H respecting that order o and maximizing $s(E(H)) + s((V \times V) \setminus E(H))$ such that $a(E(H)) \geq a((V \times V) \setminus E(H))$.

In other words, we are modelling edges in G as a mixture of two Bernoulli variables. The last constraint requires that the density of G in the edges of H, that is, next to diagonal, should be higher than in the non-edges of H. As mentioned in the introduction, we are interested in a more general setup where we can discover several bands. This gives us the following optimization problem.

Problem 2 (K-band discovery) Given a graph $G = (V, E)$ and integer K, find an order o on vertices and $K + 1$ graphs $H_0, \ldots H_K$ respecting order such that $\emptyset = E(H_0) \subsetneq \cdots \subsetneq E(H_K) = V \times V$, the density is decreasing, $a(C_i) \geq a(C_{i+1})$, and the score $\sum_{i=1}^{K} s(C_i)$ is maximized, where $C_i = E(H_i) \setminus E(H_{i-1})$.

In order to approach this optimization problem we will split it into two subproblems. The first subproblem is to find the bands for a fixed order.

Problem 3 (ordered K-band discovery) Given a graph $G = (V, E)$ an integer K and an order o on vertices, find $K + 1$ graphs $H_0, \ldots H_K$ respecting order such that $\emptyset = E(H_0) \subsetneq \cdots \subsetneq E(H_K) = V \times V$, the density is decreasing, $a(C_i) \geq a(C_{i+1})$, and the score $\sum_{i=1}^{K} s(C_i)$ is maximized, where $C_i = E(H_i) \setminus E(H_{i-1})$.

The second subproblem is to find the actual order. Our main focus will be the first subproblem for which we develop a polynomial exact solution. We address discovering the order in Section 5.

2.2 Band Discovery as Monotonic 2D-segmentation

Graph H in ordered band discovery has a special property: if we order H based on the order o and consider the upper-half of the adjacency matrix, then we see that all 1s are concentrated next to the diagonal. We will use this observation to cast band discovery into a more general segmentation problem.

In order to do so, assume that we are given a dataset of size $M \times N$. Define $A = \{(a, b) \mid 1 \leq a \leq M, 1 \leq b \leq N\}$ to be the set of all entries. We say that $U \subseteq A$ is a *corner* if for every $(a, b) \in U$, and for every x and y such that $1 \leq x \leq a$ and $1 \leq y \leq b$, an entry (x, y) is a member of U.

Given an integer K and a corner B, we define a *K-segmentation* to be the set of $K+1$ corners, $U_0, \ldots, U_K$ such that $U_0 = B$, $U_K = A$, and $U_{i-1} \subseteq U_i$ for each $i = 1, \ldots, K$. We will refer to the difference set $U_i \setminus U_{i-1}$ as a *segment*.

Our goal is to segment given data into K segments such that this segmentation maximizes the likelihood of a log-linear model for each segment. By log-linear models, also known as exponential family, we mean models whose probability density function can be written as

$$p(x \mid r) = \exp(q(x) + Z(r) + rS(x)),$$

where S is a function mapping each data point to a real number, r is the parameter of the model, and $Z(r)$ is the normalization constant. Many standard distributions such as Bernoulli, binomial, Gaussian, and Poisson are log-linear distributions. Interestingly, one can show with a direct computation that using a Gaussian distribution with a fixed variance corresponds to minimizing L_2 error.

Before we define our score, let us demonstrate that we can safely assume that $S(x) = x$ and $q(x) = 0$. In order to do that, let $C_1, \ldots, C_K$ be K subsets of $A \setminus B$ such that $C_1 \cup \cdots \cup C_K = A \setminus B$ and $C_i \cap C_j = \emptyset$ for $i \neq j$. Assume that we have assigned to each C_i, a parameter r_i.

We will measure the goodness of a segmentation by the log-likelihood of the model,

$$\begin{aligned}\sum_{i=1}^{K} \sum_{c \in C_i} \log p(D(c) \mid r_i) &= \sum_{i=1}^{K} \big(|C_i| Z(r) + \sum_{c \in C_i} q(D(c)) + \sum_{c \in C_i} r_i S(D(c))\big) \\ &= \sum_{c \in A \setminus B} q(D(c)) + \sum_{i=1}^{K} \big(|C_i| Z(r) + r_i \sum_{c \in C_i} S(D(c))\big) \quad .\end{aligned}$$

Note that the first term does not depend on C_i or r_i. Consequently, we can ignore it and by doing so ignore $q(x)$ term. We can also safely assume that $S(x) = x$. Otherwise, we can transform data D to a new dataset D' by setting $D'(c) = S(D(c))$.

We can now formally define our score. Given a segment C and a parameter r, we define the score $s(C \mid r)$ as

$$s(C \mid r) = |C|(Z(r) + r a(C)), \text{ where } a(C) = \frac{1}{|C|} \sum_{c \in C} D(c) \quad .$$

We also define $s(C) = \sup_r s(C \mid r)$ to be the score of the optimal model. Given a K-segmentation $\mathcal{U} = (U_0, \ldots, U_K)$, we define the score $s(\mathcal{U})$ as

$$s(\mathcal{U}) = \sum_{i=1}^{K} s(U_i \setminus U_{i-1}) \quad .$$

We say that a K-segmentation $\mathcal{U}$ is *monotonic* if $a(U_{i+1}) \leq a(U_i)$ for each $1 \leq i \leq K-1$ such that $U_i \neq \emptyset$. Our goal is to solve the following problem.

Problem 4 (2D-segmentation) Given a dataset D, a corner B, a log-linear model, and an integer K, find a monotonic K-segmentation $\mathcal{U}$ maximizing $s(\mathcal{U})$.

We can now see that band discovery is in fact an instance of 2D-segmentation. The dataset D is in fact the adjacency matrix of G, the corner B is a diagonal, and the score model is Bernoulli variable. We should point out that solving the more general problem has its advantages. If we have weights on edges, we can use some other log-linear model, such as Poisson model, to score the segmentation. Moreover, we do not have to restrict ourselves to graphs, we can segment any given matrix. On the other hand, discovering ordered bands is essentially as difficult as solving 2D-segmentation, that is, the amount of additional work we need to do to solve the more general case is negligible.

From now on, we will ignore B for the sake of simplicity, and assume that we want to segment the whole dataset. The algorithms that we present can be easily adjusted to handle the more general case when B is given.

The next two sections are devoted to solving Problem 4. We discuss discovering the vertex order in Section 5.

3 Borders

We can easily show that there are $\binom{N+M}{M}$ different corners for a data of size $M \times N$. This implies that we cannot enumerate all possible corners in order to solve Problem 4. However, we can show that we do not have to consider all possible corners.

In order to do so, we say that a corner U is a *border* if there are no corners X and Y such that $X \subsetneq U \subsetneq Y$ and $a(Y \setminus U) \geq a(U \setminus X)$. We denote all borders by $brd(D)$. An example of a non-border is given in Figure 2(a).

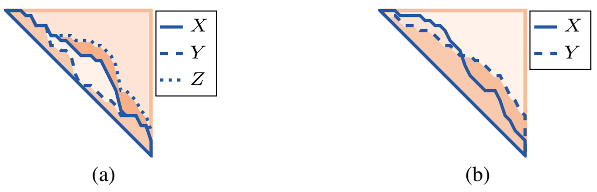


Fig. 2 Toy examples of corners. In Figure 2(a) X is a corner but not a border since $a(Z \setminus X) \geq a(X \setminus Y)$. Figure 2(b) shows an example of Proposition 2. Both X and Y are corners but X cannot be a border since $a(Y \setminus X) \geq a(X \setminus Y)$. The density of the corners is represented by the shade of the color.

The next key proposition shows that we can safely ignore all corners that are not borders.

Proposition 1 *Let $\mathcal{U}$ be a K-segmentation. Then there exists a K-segmentation $\mathcal{V} = (V_0, \ldots, V_K)$ such that $V_i \in brd(D)$ for every $i = 0, \ldots, K$ and $s(\mathcal{V}) \geq s(\mathcal{U})$.*

We present the proof for Proposition 1 in Appendix A.

The following proposition states that the borders form a chain, a key property, which we also illustrate in Figure 2(b).

Proposition 2 *Let D be a dataset. Let $U,\ V \in brd(D)$ be two borders. Then either $U \subseteq V$ or $V \subseteq U$.*

Proof Assume that the proposition does not hold, then both sets $U \setminus V$ and $U \setminus V$ are non-empty. Assume that $a(U \setminus V) \geq a(V \setminus U)$, otherwise swap U and V. Let $X = U \cap V$ and $Y = U \cup V$. We have $a(Y \setminus V) = a(U \setminus V) \geq a(V \setminus U) = a(V \setminus X)$. Consequently, V is not a border, which is a contradiction. □

Proposition 2 has several key implications. Assume that we have a dataset D of size $N \times M$. Then the number of borders is bounded by $NM + 1$. From now on we will assume that $brd(D) = U_0, \ldots, U_L$ is ordered from smallest to largest. In order to store this list efficiently, we define $C_i = U_i \setminus U_{i-1}$ for $i = 1, \ldots, L$. Instead of storing U_i we simply store C_i. Since C_i are disjoint and the union of C_i is equal to A, the set of all entries, we can store the borders in $O(NM)$ space.

Propositions 1–2 allow us to divide Problem 4 into two subproblems. The first problem is to find all the borders.

Problem 5 Given a dataset D, compute $brd(D)$.

After we have discovered the borders, we can now use them to find the optimal segmentation. We will restate the problem in a slightly different manner, using directly segments instead of corners. In order to do so, given a list of segments $C_1, \ldots, C_L$, let us define $C_{[a,b]}$ to mean $\bigcup_{i=a}^{b} C_i$. Note that if we set $C_i = U_i \setminus U_{i-1}$, then it follows that $C_{[a,b]} = U_b \setminus U_{a-1}$. This implies that we can reformulate the optimization problem as follows.

Problem 6 Given a sequence of L segments $C_1, \ldots, C_L$ and an integer K, find K intervals, $[b_i, e_i]$, such that $b_1 = 1$, $e_K = L$, and $b_i = e_{i-1} + 1$, for $i = 2, \ldots, K$, optimizing

$$\sum_{i}^{K} s\big(C_{[b_i, e_i]}\big) \quad .$$

Note that we have dropped the monotonicity condition from the definition of the problem. We will see later in Corollary 1 that if C_i are constructed from borders, that is, $C_i = U_i \setminus U_{i-1}$, then $a(C_{i+1}) < a(C_i)$. Thus monotonicity will automatically be guaranteed.

Problem 6 is in fact an instance of a classic sequence segmentation problem, where the goal is to split a sequence of length L into K homogeneous segments. This can be solved by a dynamic program in $O(L^2K)$ time and $O(KL)$ space (Bellman, 1961). To see this let us write $O_k(i)$ to be an optimal k-segmentation for a sequence $C_1, \ldots, C_i$. Then $O_k(i)$ is equal to $O_{k-1}(j)$ augmented with $[j+1, i]$, and $j < i$ is selected such that the score

$$s(O_{k-1}(j)) + s\big(C_{[j+1,i]}\big)$$

is maximized. We can iteratively compute this by first computing $O_1(i)$ for each $i = 1, \ldots, L$, and use the above identity to $O_k(i)$ from $O_{k-1}(i)$ until we reach K segments.

To summarize, we discover bands in 3 steps:

1. Order the dataset, if one is not given.
2. Compute borders $brd(D)$ (Problem 5).
3. Segment borders to obtain K-segmentation (Problem 6).

We have already shown that Problem 6 can be solved by using the classic segmentation technique. In the next two sections we discuss how to obtain the borders, either exactly or approximately. Finally, in Section 5 we discuss how to obtain the order. As discovering the order seems to be computationally intractable we resort to spectral heuristics.

4 Discovering borders

In this section we focus on discovering borders (Problem 5). Namely, we provide a polynomial-time dynamic program that discovers borders correctly. In addition we provide a heuristic for the cases when the exact discovery is too time-consuming.

4.1 Maximal and Minimal Corners

In order to define the discovery algorithm, we need to introduce the notion of maximal and minimal corners. We will show that these notions are closely related to borders.

Let U be a corner. We define $\mathit{left}(U)$ to be the *minimal corner* $V \subsetneq U$ such that $a(U \setminus V)$ is the smallest possible. If there are several candidates, we choose the smallest one.[2] We also define $\mathit{right}(U)$ to be the *maximal corner* $V \supsetneq U$ such that $a(V \setminus U)$ is the largest possible. If there are several candidates, we choose the largest one.

We can use maximal and minimal corners to describe borders.

Proposition 3 *Let U and V be two consecutive borders. Then $U = \mathit{left}(V)$ and $V = \mathit{right}(U)$.*

We prove this proposition in Appendix B.

This proposition has an important corollary that shows why we can ignore the monotonicity condition in Problem 6. Indeed segments between borders will have automatically decreasing average.

Corollary 1 *Let $U, V, W \in brd(D)$ be three consecutive borders, $U \subsetneq V \subsetneq W$. Then $a(V \setminus U) > a(W \setminus V)$.*

Proof Proposition 3 implies that $\mathit{right}(U) = V$. Since $V \subsetneq W$, Lemma 1 (given in Appendix B) implies that $a(V \setminus U) > a(W \setminus V)$. □

[2] We can easily show that this choice is unique.

4.2 Exact discovery

In this section we present an algorithm for computing the borders.

Discovering borders exactly is in fact an instance of isotonic regression for a grid. In this regression we are given a grid, and a set of values on each grid entry. The goal is to find another set of values such that they decrease as we move towards the one selected corner and L_2 error is minimized as we move towards that corner.

Problem 7 Let D be a dataset of size $M \times N$. Find a function f such that $f(x, y) \geq f(x, y + 1)$ and $f(x, y) \geq f(x + 1, y)$ minimizing the cost

$$\sum_{(x,y)} (f(x, y) - D(x, y))^2 \quad .$$

Once these these values are discovered, we can reconstruct the borders using the following proposition.

Proposition 4 *Let D be a dataset and let f be the solution to the grid isotonic regression. Let B be a border. Then there is σ such that $B = \{(x, y) \mid f(x, y) \geq \sigma\}$.*

Proof Let $m = \min_{(x,y)\in B} f(x, y)$. Define $Y = \{(x, y) \notin B \mid f(x, y) \geq m\}$. We need to show that $Y = \emptyset$. Assume otherwise. Note that since f is monotonic, $B \cup Y$ is a corner. Since B is a border, Proposition 3 implies that $a(Y) < m$. An entry, if such exists outside $B \cup Y$ must be lower than m. Hence, we can decrease the values of f in Y by a small amount, say ϵ, without violating the monotonic constraint. Let us consider the effect. In order to do this, consider the following derivative,

$$\frac{d}{dc} \sum_{(x,y)\in Y} (f(x, y) - c - D(x, y))^2 = 2 \sum_{(x,y)\in Y} -f(x, y) + D(x, y) < 0,$$

where the inequality follows from the fact that $a(Y) < m \leq f(x, y)$ for any $(x, y) \in Y$. Hence, we can decrease the values of f in Y by a small amount such that the monotonicity still holds and the score is decreased. This contradicts the fact that f is the optimal solution. □

This proposition gives us a simple approach to discover borders by varying σ.

Finding the solution for grid isotonic regression can be done in $O((NM)^2)$ time and $O(NM)$ space by an algorithm of Spouge et al. (2003). However, the time complexity of the algorithm is overly pessimistic. The algorithm runs in $O((NM)L)$, where L is the number of borders. The number of borders is NM, at worst, but in practice it is much smaller. The algorithm of Spouge et al. (2003) is a conquer-and-divide algorithm. The running time $O((NM)L)$ is based on the pessimistic assumption that each division is imbalanced, that is, only one point is separated. If these divisions are (nearly) balanced, then in practice the running time will be closer to $O((NM) \log L)$. Additional speed-ups are possible if instead of considering the full matrix of size NM we first compute the smallest corner containing the whole data, and apply the algorithm to the corner. The points that are left outside constitute a border with 0 average. This trick may speed-up the search significantly but it is highly vulnerable to noise as one non-zero point is enough to counter this optimization.

4.3 Heuristic discovery

The computational complexity of isotonic regression may become too high in practice, especially due to the $O(NM)$ term. Hence, in this section we present a practical heuristic approach. The idea here is to sort individual entries of D into a sequence. Once we have this sequence we can use it to compute candidates for borders. We can then use these candidates to rearrange the entries again, and repeat the procedure until convergence.

To make this more formal, assume that we are given a dataset D of size $M \times N$. Let $T = (t_1 = (x_1, y_1), \ldots, t_{NM} = (x_{NM}, y_{NM}))$ be a sequence of all entries of D. We say that T is a *monotonic entry order* if for any k, $1 \leq k \leq NM$, the set $t_1, \ldots, t_k$ is a corner.

Given a monotonic entry order $T = t_1, \ldots, t_{NM}$ and an integer i, we define

$$\mathit{right}(i; T) = \{t_1, \ldots, t_j\}\,, \text{ where } j = \arg\max_{j > i} a(t_{i+1}, \ldots, t_j)\ .$$

If there are ties, we select the largest index.

Given a monotonic entry order T, consider the following process. Set $U_0 = \emptyset$ and then iteratively compute $U_i = \mathit{right}(|U_{i-1}|; T)$ until we reach U_L containing all the entries. We will write $\mathit{brd}(T) = U_0, \ldots, U_L$.

We say that a monotonic entry order $T = t_1, \ldots, t_{NM}$ is *compatible* with $\mathit{brd}(D)$ if for each border $U \in \mathit{brd}(D)$ there exists an index k such that $t_1, \ldots, t_k = U$. Such order always exists.

Assume that we are given an order T compatible with $\mathit{brd}(D)$. Select and fix a border $U_i \in \mathit{brd}(D)$. Let $k = |U_i|$. We must have $U_{i+1} = \mathit{right}(k; T)$. This implies that if we are given an order T that is compatible with $\mathit{brd}(D)$, then $\mathit{brd}(T) = \mathit{brd}(D)$.

A naive approach to compute an individual $\mathit{right}(i; T)$ requires $O(NM)$ time, however, we can compute $\mathit{right}(i; T)$ for each i *simultaneously* in $O(NM)$ total time using the approach given by Calders et al. (2007), where the goal of the authors was to discover the densest interval, essentially $\mathit{right}(i; T)$, from a stream T in an amortized constant time for each i. This algorithm is actually a variation of PAVA algorithm for solving isotonic regression for a total order, see (Ayer et al., 1955), though the goal and the output of the algorithms are different. Connection to isotonic regression is natural as we saw previously that the exact borders can be discovered by solving the grid isotonic regression in $O(N^2M^2)$ time. Since we no longer consider a grid but a monotonic entry order, we can use a more efficient algorithm whose copmputational complexity is $O(NM)$.

Let us now consider the optimization problem from an another angle. Assume that we do not know $\mathit{brd}(D)$ but instead we know only the average of values in segments, that is, for each entry (i, j), we know the average, say $w(i, j)$, of the segment that contains (i, j). We can construct the order compatible with $\mathit{brd}(D)$ from the weights. Corollary 1 implies that weights should decrease. Hence, if we build a monotonic entry order by greedily selecting the entry with the largest weight while at the same time making sure that the order is indeed monotonic, we end up with an order that is compatible with $\mathit{brd}(D)$. We present the pseudo-code for this approach in Algorithm 1.

We can now construct our algorithm for discovering approximate borders. Given a monotonic entry order $T = t_1, \ldots, t_{NM}$, we can compute the borders $\mathit{brd}(T) =$

Algorithm 1: FINDORDER, constructs a monotonic order from weights

```
   input  : weights $w$
   output : a monotonic entry order $T$
1  $H \leftarrow (1, 1)$; $T \leftarrow$ empty list;
2  while $H$ is not empty do
3      pop $(i, j)$ from $H$ with the largest weight $w(i, j)$;
4      add $(i, j)$ to $T$;
5      mark $(i, j)$ as visited;
6      if $i < M$ and $(j = 0$ or $(i + 1, j - 1)$ is marked then
7          add $(i + 1, j)$ to $H$;
8      if $j < N$ and $(i = 0$ or $(i - 1, j + 1)$ is marked then
9          add $(i, j + 1)$ to $H$;
10 return $T$;
```

$U_0, \ldots, U_L$. Assume that we are given an entry p. Let $U_k \in brd(T)$ be the border such that $p \in U_k \setminus U_{k-1}$. Let us define $w_S(p; T) = a(U_k \setminus U_{k-1})$ to be the density of the segment containing p. Once these weights are computed, we can use them to compute a new order using FINDORDER. Computing weights can be done in $O(NM)$ time while computing a new order can be done in $O(NM \log \min(N, M))$ time.

Using just w_S is problematic in practice. The reason for this is that during FIND-ORDER there will be often several entries in H that belong to the same segment, and so will have the same values of w_S. Hence, we need a weight function that that will break these ties. Breaking ties properly is important since it is possible to design a weight w that $T = \text{FINDORDER}(w(\cdot; T))$ for any T. In other words, iterating FINDORDER and computing weights will never improve the current order.

Given an order T, we say that a weight function w is a *tie-breaker* if $w_S(p; T) < w_S(q; T)$ implies $w(p; T) < w(q; T)$, where p and q are entries in T. Before considering any specific tie-breakers, let us first show that using a tie-breaker w leads to a convergence.

Proposition 5 *Let w be a tie-breaker. Let T^0 be any monotonic entry order and define $T^{i+1} = \text{FINDORDER}\big(w(\cdot; T^i)\big)$. Then there exists k such that $brd\big(T^i\big) = brd\big(T^k\big)$ for any $i \geq k$.*

Proof Fix m and let $U = T^m$ and $V = v_1, \ldots, v_{NM} = \text{FINDORDER}(w(\cdot; U))$. Let $B_0, \ldots, B_L = brd(U)$ and define $C_0, \ldots, C_K = brd(V)$. Define a vector α of length $2L$ such that $\alpha_{2i-1} = a(B_i \setminus B_{i-1})$ and $\alpha_{2i} = |B_i|$. Define similarly β using $brd(V)$.

Assume two entries p and q such that $p \in B_i$ and $q \notin B_i$. This means that $w_S(p; U) > w_S(q; U)$. Since w is a tie-breaker, this means that p will occur earlier than q in V. In other words, $\{v_1, \ldots, v_{|B_j|}\} = B_j$ as sets, for any $j = 1, \ldots, L$.

If $brd(U) = brd(V)$, then it follows immediately that $\alpha = \beta$.

Assume that $brd(U) \neq brd(V)$ and let j be the largest index such that $B_i = C_i$ for any $i \leq j$. This implies that $\alpha_i = \beta_i$, for $i \leq 2j$. Since $\{v_{|B_j|+1}, \ldots, v_{|B_{j+1}|}\} = B_{j+1} \setminus B_j$ and $C_j = B_j$, we know that $C_{j+1} \setminus C_j$ will be as dense as $B_{j+1} \setminus B_j$, that is, it follows that either $a(C_{j+1} \setminus C_j) > a(B_{j+1} \setminus B_j)$, or $a(C_{j+1} \setminus C_j) =$

$a(B_{j+1} \setminus B_j)$ and $|C_{j+1}| > |B_{j+1}|$. That is, either $\alpha_{2j+1} < \beta_{2j+1}$, or $\alpha_{2j+1} = \beta_{2j+1}$ and $\alpha_{2j+2} < \beta_{2j+2}$. Consequently, β is larger than α with respect to the lexicographical order.

We have shown that if $brd\left(T^i\right) \neq brd\left(T^{i+1}\right)$, then for any $j > i$, $brd\left(T^i\right) \neq brd\left(T^j\right)$. Since there are only finite number of possible borders, there exists an index k such that $brd\left(T^i\right) = brd\left(T^k\right)$ for any $i \geq k$. $\square$

We consider two tie-breakers. The first tie-breaker, w_R breaks the ties in a random order. Formally, we define $w_R(p;T) = (w_S(p;T), r)$ to be a vector of length 2, where T is an order, p is an entry in T, and r is a random number between 0 and 1. The weights are compared in lexicographical order. This immediately implies that w_R is a tie-breaker. Our second tie-breaker tries to flip the original order as much as possible. Formally, we define $w_F(p;T) = (w_S(p;T), i)$, where T is an order, p is an entry in T, and i is the index of p in T. Obviously, w_F is a tie-breaker, and it will favor the entries that appear later in T.

Proposition 5 states that the iteration will converge. However, we need means to detect this convergence. This is difficult since while the borders themselves will converge, the actual orders may cycle. Fortunately, we can show that w_F has a cycle of at most 2, we prove this proposition in Appendix C.

Proposition 6 *Let T^0 be any monotonic entry order and define iteratively $T^{i+1} = \textsc{FindOrder}\left(w_F(\cdot; T^i)\right)$. There exists m such that $T^{m+2} = T^m$.*

We can easily detect convergence for w_F by remembering the second last order and compare it to the current one. If the orders are the same, then we have converged.

Unfortunately, we cannot detect convergence easily for w_R. Hence we adopt the following hybrid approach. We begin with an random monotonic order, and apply w_F until convergence. Once converged, we apply w_R once, and repeat applying w_F until we converge again. We repeat this until the borders have not changed after 20 applications of w_R. This is summarized in Algorithm 2.

Algorithm 2: Heuristic discovery of borders

```
1 assign all entries to a single border;
2 while the borders have not changed for $L$ iterations do
3     apply FindOrder using random tie-breaker $w_R$;
4     while convergence do
5         find borders in ordered entries;
6         apply FindOrder using flip tie-breaker $w_F$;
```

4.4 Using sparsity to speed-up the discovery

So far we have operated with ordinary matrices, and consequently the running time for our heuristic discovery is $O(N^2 \log N)$ when applied to a graph with N vertices. In practice, graphs are sparse and we can use this sparsity to our advantage.

Assume that we are given a graph G with positive weights and a corner B. Let us define

$$F = \{(x, y) \in B \mid (x + 1, y) \notin B \text{ and } (x, y + 1) \notin B\}$$

to be the set of *frontier* points, that is, points in B from which you cannot advance away from the diagonal, see Figure 3(a). Now, Proposition 3 implies that if B is a border, then F must be a subset of edges. Otherwise, we can always remove a non-edge frontier point and by doing so increase the density of B. Note also that given F, we can always recover B by taking the smallest corner containing F.

This suggests that instead of ordering all entries of the adjacency matrix it is enough to order just the edges. This, however, poses two complications. First, we need to be able keep the edges in a monotonic order. While this was easy when dealing with the full matrix since visiting one entry revealed only 2 adjacent entries, at maximum, visiting an edge may reveal any number of unvisited edges. Secondly, in order to compute the density of B, whenever we visit an edge we need to compute the number of non-edges we visited implicitly in order to reach that edge, see Figure 3(c).

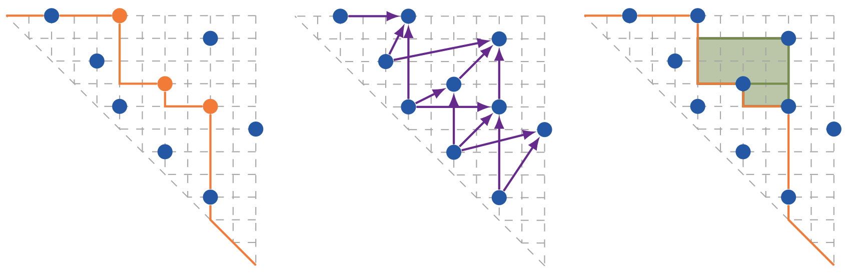

(a) corner and its frontier set (b) lattice imposed on edges (c) encapsulated non-edges

Fig. 3 Toy examples related to computing borders from sparse graphs

In order to solve the first problem, we define a lattice structure on the *edges*. More formally, we define a directed acyclic graph L such that the vertices correspond to the edges of G and the edges in $E(L)$ are formed as follows: vertex (x_1, y_1) is connected to (x_2, y_2) in L if $x_1 \leq x_2$ and $y_1 \leq y_2$ and there is no third vertex (x_3, y_3) such that $x_1 \leq x_3 \leq x_2$ and $y_1 \leq y_3 \leq y_2$, see Figure 3(b), for example. In order to guarantee that the edges are visited in a monotonic order, we simply visit the edges in a topological order of L. This modification of FINDORDER can be done to run in $O(|E(L)| + |E(G)| \log |V(G)|)$ time. Moreover, the lattice can be constructed in $O(|E(L)| \log |V(G)|)$ time.

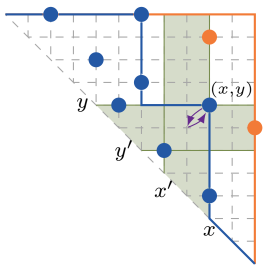
In order to solve the second problem, that is, to compute the number of non-edges encapsulated by visiting a new entry, see Figure 3(c), first note that we need this information when computing the borders from the monotonic order. Consequently, at this point we have an order at our disposal. We enumerate the entries in the order and during this enumeration we compute and update the frontier set of the border corresponding to the entries visited so far. This update can be done efficiently by keeping the entries, which are edges in G and have a form, say (x, y), in a red-black tree indexed by x. Adding a new entry into a border will delete at most k entries from the frontier set, where k is the number of parents in the lattice. There can be at most $|V(G)|$ frontier entries, at any time. Hence, the running time for this enumeration is $O(|E(L)| \log |V(G)|)$. Assume now that we have computed the frontier set for $i - 1$ entries, and we need to compute the encapsulated non-edges for ith entry, say (x, y). We do this by finding the entry from the frontier set (u, v) such that u is the largest possible value and $u \leq x$. We then use this entry as a starting point and iterate to the following entries w.r.t. the red-black tree. At each entry we compute the number of non-edges captured between two adjacent nodes of the frontier set and (x, y), see Figure 3(c) for illustration. We need to visit only $O(k)$ entries, where k is the number of parents of ith entry in L. Hence, computing the areas for all entries can be done in $O(|E(L)| \log |V(G)|)$ time.

5 Discovering Order

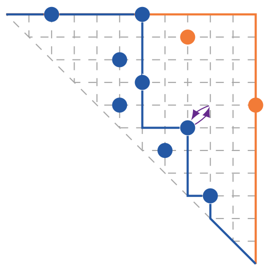
Our main focus so far was to compute the segmentation from already ordered data. This order may be given, for example, if each vertex has a time stamp. If the order is not known, we will compute the order using the Fiedler vector (Fiedler, 1975), a popular technique for ordering matrices and graphs. Fiedler order has a tendency of pushing the edges towards the diagonal. In fact, we can show that if there *exists* an order of vertices of the graph G such that the edges respect the order, that is, all edges are around diagonal, then Fiedler vector will find this order (Fiedler, 1975).

We use Fiedler order as an initial order and introduce a simple heuristic refinement. Assume that we have computed a segmentation using the order. Let B one of the segments and let (x, y) be a frontier, as defined in Section 4.4, of B. Find the smallest vertex x' such that for any u, $x' \leq u \leq x$, it holds that for any v, (x, v) and (u, v) belong to the same segment. Similarly, find the largest vertex y' w.r.t. y, see Figure 4(a). It immediately follows that a permutation of vertices between x' and x, and y and y' cannot decrease the score since none of the entries will leave its segment. Assume that there is a non-edge entry (u, v) such that $x' \leq u \leq x$ and $y \leq v \leq y'$. Then if we swap x and u, and v and y, we essentially decrease the area of the segment since none of the entries will leave the segment but at the same time (u, v) cannot be a frontier since it is non-edge, see Figure 3(b). If there are several non-edges, we select the non-edge, say (u, v), minimizing $|(u, z) \in E(G) \mid y \leq z \leq y'| + |(z, v) \in E(G) \mid x' \leq z \leq x|$, that is, we select u and v with the smallest number of edges.

We will do these swaps until no swaps are possible. For the sake of efficiency, we do these swaps in a batch style, that is, for each frontier point (x, y) we first compute x' and y', and then perform swap. During these swaps we make sure that once an interval $[x', x]$ or $[y, y']$ is used for a swap, it will not be used again in the same batch. Once no swaps are possible, we recompute the segmentation, and repeat the refinement. This will eventually converge since the score will always increase after the refinement.



(a) corner before the swap



(b) corner after the swap

Fig. 4 Toy examples related to refining order: before and after swapping (x, y)

6 Related Work

To our knowledge our proposed optimization problem is novel. However, there are similar tasks. A common plotting technique is a contour plot, where a line follows a constant value. Our optimization technique is inherently different as we are considering all the points inside a segment and not just the data points within the vicinity of the boundary.

A related optimization problem was introduced by Mannila and Terzi (2007), where the authors introduced a concept of nestedness for binary data. A binary data is nested if for all pairs of rows, one row is either a superset or a subset of another. If data is fully nested, we can order it in a way that all 1s occur in the top-left corner. The authors considered minimizing number of 0s that needs to be replaced such data becomes fully nested. Authors also consider partitioning columns into K parts such that each part is almost fully segmented. While both approaches model the same phenomenon, there are significant differences. We can handle more general datasets such as counting data and real-valued data, and our scoring function is based on the log-likelihood. We are interested in finding K-segmentations, whereas Mannila and Terzi (2007) focus on 2-segmentation. On the other hand, we assume that our data is preordered whereas no such assumption is made by Mannila and Terzi (2007).

The discovered bands give rise for each vertex a set of expanding communities. Discovering similar structures have been suggested. For example, Alvarez-Hamelin et al. (2006) order nodes into a tree by deleting iteratively low-degree vertices. In another example, Tatti and Gionis (2013) suggested discovering nested communities given a set of seed nodes.

The spectral approach to discover an order falls into a larger category of approaches used for solving the seriation problem. In seriation, a typical goal is, given a similarity matrix between objects, to organize the objects that minimize some stress function (Hahsler et al., 2008). In the ideal case, the discovered order will rearrange the similarity matrix into a *Robinson* form, that is, the values of the matrix will get smaller as we move away from the diagonal. In practice, such a permutation rarely exists and hence the stress function reflects how far away we are from the Robinson form. Fiedler (1975) demonstrated that the spectral method, that is, ordering using the Fiedler vector, will find the Robinson form *if* such exists. If such an order does not exist, then the spectral order can be viewed as an algorithm for placing objects on a straight line with the goal of minimizing pair-wise distance weighted by the similarity matrix. Note that the goal here is to find the locations on a line, the order of the objects comes as a by-product. Optimizing stress functions that are directly based on the order of the similarity matrix is typically a **NP**-hard problem. In such cases the problem typically resembles a travelling salesman problem. Consequently, heuristics to discover a good order are variants of TSP solvers (Johnson et al., 2004).

Ordering data rows and columns and using this order to discover an underlying structure has been suggested. Gionis et al. (2004) and Tatti and Vreeken (2012) suggested discovering tile hierarchies from ordered binary datasets. Both work used Bernoulli models, a log-linear model, to score the hierarchies.

In our experiments, the number of borders is small. Hence, we are able to solve Problem 6 exactly using a dynamic program. This program requires quadratic time. If the number of borders becomes too large, that is, close to the number of entries, using an exact solver becomes impractical. In order to overcome this problem we can use heuristic approaches, such as top-down approaches where we select greedily a new change-point (see Bernaola-Galván et al. (1996); Douglas and Peucker (1973); Lavrenko et al. (2000); Shatkay and Zdonik (1996), for example) or bottom-up approaches where at the beginning each point is a segment, and points are combined in a greedy fashion (see Palpanas et al. (2004), for example). Yet another option is a randomized heuristic was suggested by Himberg et al. (2001), where we start from a random segmentation and optimize the segment boundaries. These approaches, although fast, are heuristics and have no theoretical guarantees of the approximation quality. A divide-and-segment approach, an approximation algorithm with theoretical guarantees on the approximation quality was given by Terzi and Tsaparas (2006).

7 Experiments

Setup: In our experiments we used 6 real-world datasets and one synthetic data. The first two graphs, *DblpCP* and *DblpCF*, are ego-networks of Christos Papadimitriou and Christos Faloutsos, that is, the graphs contain the collaborators obtained from DBLP,[3] two researchers are connected if they have a joint paper. The other two datasets, *Fb107* and *Fb1912*, were the two largest ego-networks taken from the Facebook dataset obtained from SNAP repository.[4] The *Mammals* presence data consists of presence records of European mammals within geographical areas of 50×50 kilometers (Mitchell-Jones et al., 1999).[5] We computed a similarity matrix between different locations, say i and j, by considering the observed joint number of different mammals, normalized by $c_i c_j$, where c_i and c_j are the number of mammals observed at i and j, respectively. Dataset *Paleo*[6] contains information of species fossils found in specific paleontological sites in Europe Fortelius (2005). We constructed a

[3] http://www.informatik.uni-trier.de/~ley/db/

[4] http://snap.stanford.edu/snap/

[5] The full version of the *Mammals* dataset is available for research purposes from the Societas Europaea Mammalogica at http://www.european-mammals.org

[6] NOW public release 030717 available from Fortelius (2005).

similarity matrix between two fossils, say a and b, by computing the number of sites in which both a and b have been discovered. We also created a synthetic dataset with 250 000 vertices and 279 223 edges where edges had a higher probability of being closer to the diagonal. The basic characteristics of the datasets are given in Table 1.

We ordered the vertices of each graph (except synthetic) according to Fiedler's vector using algorithm given by Atkins et al. (1999). Since Atkins' algorithm may produce several orders, this happens when the graph has disconnected components, we pick one order at random. Once the graph is ordered we apply our heuristic method given in Section 4.3 along with the refinement step described in Section 5. We also applied the algorithm for discovering for exact borders described in Section 4.2. We used Bernoulli model for unweighted graphs, Poisson model for *Paleo* data and L_2 error for *Mammals*. We set the number of bands to 3 in DBLP and *Paleo* datasets and 4 in Facebook and *Mammals* datasets. The statistics of the datasets and the experiments are given in Table 1. Due to the slow convergence of heuristic method in *Mammals* and *Synthetic*, we limited the number of iterations to 2000 and did not apply refinement. We also noticed that after each random tie-breaker the score gets a (relatively small) bump. Hence, we force random tie-breaker after each 50th iteration in these two datasets.

Table 1 Basic characteristics of the datasets, the number of edges in lattice, see Section 4.4.

Name	$\lvert V\rvert$	$\lvert E\rvert$	$E(L)$
DblpCF	176	457	794
DblpCP	154	348	717
Fb107	1034	26 749	61 279
Fb1912	747	30 025	62 842
Paleo	139	4428	8737
Mammals	2183	2 378 193	4 752 044
Synthetic	250 000	279 223	2 124 683

Table 2 Basic statistics from experiments using exact and approximate borders. The columns contain running times, the number of borders, the number of flip tie-breakers, the number of random tie-breakers, the number of refinement rounds, initial (negative) score based on a fiedler order, and final score after the refinement.

	Approximate borders							Exact borders		
Name	time	brd	iter	rnd	ref	initial	final	time	initial	final
DblpCF	0.2s	55	235	48	3	945	908	.02s	945	905
DblpCP	0.4s	53	701	135	3	966	927	.05s	966	918
Fb107	12m	476	7217	676	7	61 734	60 444	20s	61 723	60 427
Fb1912	5m	375	7357	813	4	43 212	42 909	3.2s	43 212	42 930
Paleo	4s	201	423	51	4	-8645	-8906	.13s	-8645	-8906
Mammals	33m	2975	2000	40		19 798		2m	19 798	19 798
Synthetic	37m	625	2000	40		6 956 048				

Results: Let us first focus on computational complexity. We see that the exact algorithm works very well in practice, it is faster than the heuristic discovery, and it has the benefit of producing the exact result. The main reason for this is that for medium-sized datasets the running time for the exact algorithm is very competitive and the slowness of the heuristic algorithm is due to high number of iterations it requires to converge. This dynamic changes when the size of the data increases. For large datasets, the fact that we need to store the graph in the full form will slow the algorithm down (or simply run out of memory). This is the case with the *Synthetic* data where we could not run the exact algorithm due to the memory limitations.

Unlike the exact method, the heuristic discovery of bands can use sparsity to its advantage (see Section 4.4). More specifically, one iteration is linear w.r.t. the number of edges in the lattice constructed from the edges of the original graph. While it is possible in theory that the number of lattice edges is significantly higher than the number of edges in the original graph, third column in Table 1 demonstrates that in our experiments the number of lattice edges is about 2–3 times the number of edges in the original graph. Due to this property the running times stay reasonably small, minutes at worst.

The number of discovered borders, 5th column, is small, which in turns implies that the final segmentation is fast, and indeed we spend most of the time computing the borders.

Let us now study convergence of the heuristic in more detail. Table 2 shows that the heuristic converges close to the exact values w.r.t. the score function. The number of iterations needed in a single refinement round, given in 3th and 4th columns of Table 1, grows as the graph gets larger. The number of random tie-breaker iterations is about 10 percent. In order to study the convergence in more details, we plotted $s(\mathcal{B})$ in Figure 5, where $\mathcal{B}$ is the current set of approximate borders, as a function of rounds left to convergence. Since the initial border set comes from a random monotonic order, we repeated this experiment 10 times. Note that $s(\mathcal{B})$ is an upper bound for score of the final segmentation. From the results we see that initial orders have a bad score, and there is a high variance in the initial score and the convergence time. However, the score stabilizes quickly, close to its final value, and the final score has little dependence of the starting point. Majority of the rounds is spent in cosmetic improvement. This hints that if the convergence time is a factor, we may stop the border discovery early and still get a good score. This is demonstrated with the *Synthetic* dataset. The convergence of the score is given in Figure 5 and again we see the same behaviour where the initial order has a bad score but quickly obtains a stable score. Running the heuristic took 37 minutes and we were not able to run the exact algorithm due to the memory limitations.

Let us next look at the refinement. In our experiments, the number of refinement iterations is low, around 10. Moreover, the improvement in score is modest, around 1%. This is also the case when we apply to the datasets that are not ordered with the fiedler vector. This suggests that the refinement procedure should not be used alone but instead it should be used only to improve the already good order. We also tried discovering bands using random orders. Naturally, the score for discovered bands is weaker. The gain of using spectral orders depends how strongly banded is the data. For example, *Paleo* contains a strong banded structure, and the score using a random

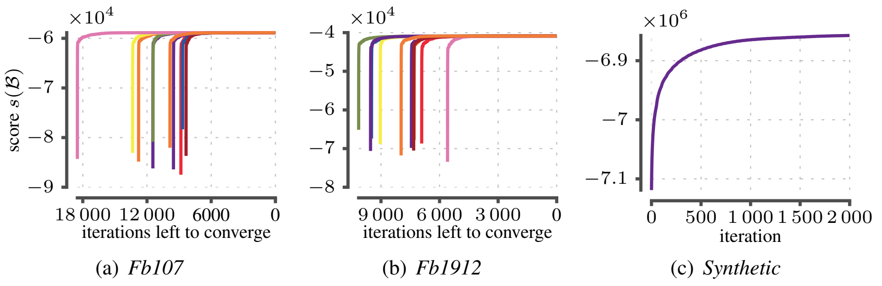


Fig. 5 Convergence during heuristic discovery. Score $s(\mathcal{B})$, where $\mathcal{B}$ is the current set of borders as a function of iterations left to convergence. Note that $s(\mathcal{B})$ is an upper bound for the score of final segmentation.

order was significantly worse than the spectral order. Finally, using the exact border discovery with the refinement may produce a worse score than using the approximate border discovery, as shown by the scores for *Fb1912*.

Finally, let us look at the discovered bands given in Figures 6–8. The first band in the DBLP datasets is relatively small, typically discovering small clusters that are due to joint publication with the author of the ego-network. Facebook graphs contain a strong clustering structure which is seen in the discovered bands. However, we also see the overlap effect: in *Fb1912* the clusters overlap. In Figure 8 we provide small snipets from DBLP graphs along with the authors. For example, in *DblpCP*, the inner band contains authors that have collaborate with each other but these authors do not form a clique. In *Paleo*, the bands correspond to the weights of the edges: the inner band has edges with higher weights while the outer bands have less edges. In *Mammals*, the spectral order—which was computed without the knowledge of the geological location—corresponds roughly to the latitude: one extreme being Scandinavia while on the other extreme being Greece and Crete. The order fails to capture Spain and Portugal suggesting that the data cannot be explained fully by a single order. The discovered bands, especially in the northern areas, suggest a continuous change in fauna as we move along the latitude.

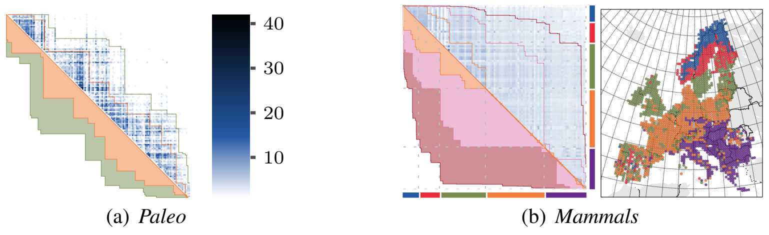
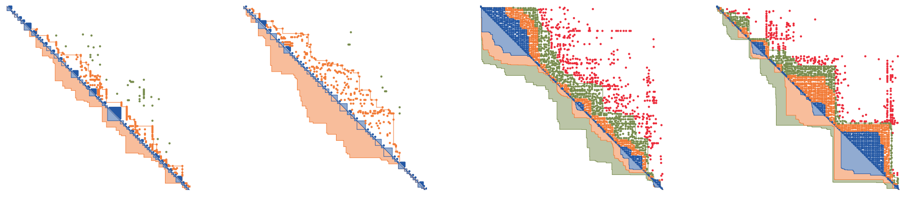
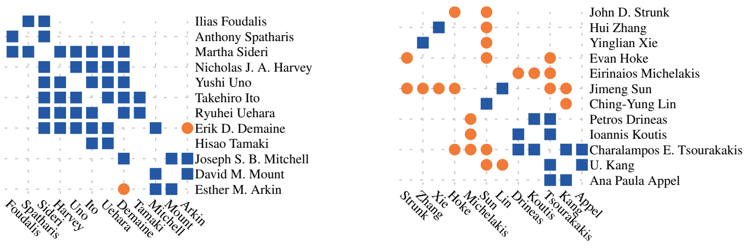


Fig. 6 Adjacency matrices and the obtained segmentations. The upper triangle depicts the weights of edges while the lower triangle shows the segmentation. *Mammals* was sparsified for visualization purposes. In 6(b), the colors of the bars along the axis correspond to the color of the locations in the map.



(a) *DblpCF* (b) *DblpCP* (c) *Fb107* (d) *Fb1912*

Fig. 7 Adjacency matrices and the obtained segmentations. The upper triangle depicts the distribution of edges while the lower triangle shows only the segmentation. The facebook graphs were sparsified for visualization purposes



(a) close-up of *DblpCP* (b) close-up of *DblpCF*

Fig. 8 Snipets of *DblpCP* and *DblpCF*

8 Conclusion

In this paper we introduced an optimization problem in order to model the concentration of edges next to the diagonal. More specifically, given a graph and an integer K, our goal is to order the vertices and find K bands around the diagonal, inner bands being more dense. As a measure of goodness we use the likelihood of a log-linear model, a family containing many standard distributions.

We divide the problem into two subproblems. The first problem is to find a good order while the second problem is to discover the bands. We introduced two solvers for the latter problem. The first solver is exact and based on isotonic regression, the second solver is a heuristic approach that sorts the entries into a linear order and applies one-dimensional isotonic regression. By doing so, the solver can exploit sparsity of the graph. Both approaches complement each other. If we can store the graph as a full adjacency matrix, then it is beneficial to use the exact algorithm. The algorithm runs in $O(N^4)$ time, at worst, but is closer to $O(N^2 \log P)$, where N is the number of vertices and P is the number of borders. On the other hand, if the number of vertices is large but the graph is sparse it is better to apply the heuristic approach with a limited number of iterations. A single iteration runs in $O(|E(L)|)$ time, where L is the lattice describing the relations between the nodes (described in Section 4.4). At worst, $|E(L)|$ is $O(N^2)$ but in practice it is smaller for sparse graphs.

We discover the order mainly by using Fiedler order. It would be fruitful to develop algorithms that induce an order while looking for the optimal segmentation simultaneously. The work done by Mannila and Terzi (2007) with nested datasets hints that this problem is **NP**-hard. However, it may be possible to develop an efficient heuristic approach.

Another open question, which we will leave to a future work, is the problem of choosing K, the number of segments. This question is not only specific to our problem but also occurs in many classic problems such as clustering or sequence segmentation. In our experiments computing the final segmentation from the set of borders is cheap. Hence, instead of studying a single segmentation we can compute several segmentations simultaneously and present all of them to the user. Alternatively, since our score is essentially a log-likelihood, we can design an MDL approach to select a segmentation with a good score.

Bibliography

J. I. Alvarez-Hamelin, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NIPS*, pages 41–50, 2006.

J. E. Atkins, E. G. Boman, and B. Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM J. Comput.*, 28(1):297–310, 1999.

M. Ayer, H. Brunk, G. Ewing, and W. Reid. An empirical distribution function for sampling with incomplete information. *The annals of mathematical statistics*, 26 (4), 1955.

R. Bellman. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM*, 4(6), 1961.

P. Bernaola-Galván, R. Román-Roldán, and J. L. Oliver. Compositional segmentation and long-range fractal correlations in dna sequences. *Physical Review E Statistical Physics Plasmas Fluids And Related Interdisciplinary Topics*, 53(5):5181–5189, 1996.

T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. In *ICDM*, pages 83–92, 2007.

D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10 (2):112–122, 1973.

P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.

M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975.

M. Fortelius. Neogene of the old world database of fossil mammals (NOW). University of Helsinki, `http://www.helsinki.fi/science/now/`, 2005.

A. Gionis, H. Mannila, and J. K. Seppänen. Geometric and combinatorial tiles in 0-1 data. In *ECML PKDD*, pages 173–184, 2004.

M. Hahsler, K. Hornik, and C. Buchta. Getting things in order: An introduction to the R package seriation. *Journal of Statistical Software*, 25(3):1–34, 2008.

N. Haiminen and A. Gionis. Unimodal segmentation of sequences. In *ICDM*, pages 106–113, 2004.

J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmäki, and H. Toivonen. Time series segmentation for context recognition in mobile devices. In *ICDM*, pages 203–210, 2001.

D. S. Johnson, S. Krishnan, J. Chhugani, S. Kumar, and S. Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In *VLDB*, pages 13–23, 2004.

V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan. Mining of concurrent text and time series. In *KDD Workshop on Text Mining*, pages 37–44, 2000.

H. Mannila and E. Terzi. Nestedness and segmented nestedness. In *KDD*, pages 480–489, 2007.

A. Mitchell-Jones, G. Amori, W. Bogdanowicz, B. Krystufek, P. H. Reijnders, F. Spitzenberger, M. Stubbe, J. Thissen, V. Vohralik, and J. Zima. *The Atlas of European Mammals*. Academic Press, 1999.

T. Palpanas, M. Vlachos, E. J. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *ICDE*, pages 339–349, 2004.

H. Shatkay and S. B. Zdonik. Approximate queries and representations for large data sequences. In *ICDE*, pages 536–545, 1996.

J. Spouge, H. Wan, and W. J. Wilbur. Least squares isotonic regression in two dimensions. *Journal of Optimization Theory and Applications*, 117, June 2003.

N. Tatti and A. Gionis. Discovering nested communities. In *ECML PKDD*, pages 32–47, 2013.

N. Tatti and J. Vreeken. Discovering descriptive tile trees - by mining optimal geometric subtiles. In *ECML PKDD*, pages 9–24, 2012.

E. Terzi and P. Tsaparas. Efficient algorithms for sequence segmentation. In *SDM*, 2006.

A Proof of Proposition 1

In order to prove this proposition, we first need to establish a series of lemmas. We will use the following fact. Let B be a set of entries, and let $C \subsetneq B$ be a non-empty subset of B such that $a(C) < a(B)$, then $a(B \setminus C) > a(B)$.

The first lemma states that if you cut a segment from a maximal corner, then that segment will be more dense. Conversely, any segment adjacent to a maximal corner will be more sparse.

Lemma 1 *Let U be a corner. Let $V = right(U)$ be a corner. Let $X \supseteq U$ be a corner. If $V \setminus X \neq \emptyset$, then $a(V \setminus X) \geq a(V \setminus U)$. If $X \setminus V \neq \emptyset$, then $a(X \setminus V) < a(V \setminus U)$.*

Proof Assume that $a(V \setminus X) < a(V \setminus U)$. Then, since

$$(V \setminus U) \setminus (V \setminus X) = (V \cap X) \setminus U,$$

$a((V \cap X) \setminus U) > a(V \setminus U)$, which contradicts the definition of V. The second case holds because otherwise $a((V \cup X) \setminus U) \geq a(V \setminus U)$, which contradicts the definition of V. □

The next lemma essentially states that if a corner cuts a maximal corner, then it cannot be a border.

Lemma 2 *Let U be a corner. Write $V = right(U)$. Let $X \supsetneq U$ be a corner such that $V \setminus X \neq \emptyset$, then $a((V \cup X) \setminus X) \geq a(X \setminus U)$. Consequently, X is not a border.*

Proof By definition of V, $a(V \setminus U) \geq a(X \setminus U)$.

Lemma 1 implies that $a(V \setminus X) \geq a(V \setminus U)$. As $(V \cup X) \setminus X = V \setminus X$, we can combine the two inequalities and prove the first claim. X cannot be a border since $U \subsetneq X \subsetneq V \cup X$. □

Lemmas 1–2 can be modified to hold for minimal corners. The proofs for these lemmas are similar to the proofs of Lemmas 1–2.

Lemma 3 *Let U be a corner. Let $V = left(U)$ be a corner. Let $X \subsetneq U$ be a corner. If $X \setminus V \neq \emptyset$, then $a(X \setminus V) \leq a(U \setminus V)$. If $V \setminus X \neq \emptyset$, then $a(V \setminus X) > a(U \setminus V)$.*

Lemma 4 *Let U be a corner. Write $V = left(U)$. Let $X \subsetneq U$ be a corner such that $X \setminus V \neq \emptyset$, then $a(X \setminus (V \cap X)) \leq a(U \setminus X)$. Consequently, X is not a border.*

Assume that we are given two K-segmentations, $\mathcal{U} = U_0, \ldots, U_K$ and $\mathcal{V} = V_0, \ldots, V_K$. We write $\mathcal{U} \subsetneqq \mathcal{V}$ if there exists i such that $U_j = V_j$ for $j < i$, and $U_i \subsetneq V_i$. The idea behind the proof of Proposition 1 is that we can replace a non-border in a segmentation, say $\mathcal{U}$, either with its minimal or maximal corner such that either the score will increase or that the resulting segmentation, say $\mathcal{V}$, is smaller wrt. partial order, $\mathcal{V} \subsetneqq \mathcal{U}$.

Next lemma makes sure that we can always pick a non-border in a segmentation that such that its maximal and minimal corners form a chain with the other corners in the segmentation.

Lemma 5 *Let $U_0, \ldots, U_K$ be a K-segmentation. Assume that, say, U_i is not a border. Then there exists U_j with $0 < j < K$, and two corners X and Y such that $U_{j-1} \subseteq X \subsetneq U_j \subsetneq Y \subseteq U_{j+1}$ and $a(Y \setminus U_j) \geq a(U_j \setminus X)$.*

Proof Since U_i is not a border, there exist X and Y such that $X \subsetneq U_i \subsetneq Y$ and $a(Y \setminus U_i) \geq a(U_i \setminus X)$. We need to show that $U_{i-1} \subseteq X$ and $Y \subseteq U_{i+1}$, or possibly modify X, Y, and i such that inclusions hold.

We can safely assume that $Y = right(U_i)$.

Assume that $Y \nsubseteq U_{i+1}$, that is, $Y \setminus U_{i+1} \neq \emptyset$. Lemma 2 implies that $a(Y \setminus U_{i+1}) \geq a(U_{i+1} \setminus U_i)$. Hence, we can redefine $X = U_i$ and $Y = right(U_{i+1})$, and then increase i by one and repeat the argument. This process will eventually stop since we increase i every step. When we finally stop, note that both inclusions will hold, and we have proved the lemma.

If we start with the case where $Y \subseteq U_i$ and $U_{i-1} \nsubseteq X$, we proceed to modify X, Y, and i in the opposite direction. □

Our next goal is Corollary 2 which will make sure that we can always make sure that the segmentation is monotonic. For that we need the following two lemmas, that follow immediately from the fact that our scoring function is a log-linear model.

Lemma 6 *$s(X \mid u)$ is a concave function of u. Let U and V be two segments such that $a(U) \leq a(V)$. Let $r = \arg\max_u s(U \mid u)$ and $t = \arg\max_u s(V \mid u)$. Then $r \leq t$.*

Proof Let X be a segment. A straightforward calculation shows that $\partial s(X \mid u) / \partial u = |X|(a(X) - \mathrm{E}_u[x])$, where $\mathrm{E}_u[x] = \sum_x x p(x \mid u)$ is the mean of the log-linear model.

We also have $\partial\, \mathrm{E}_u[x] / \partial u = \mathrm{Var}_u[x] \geq 0$. This immediately proves that $s(X \mid u)$ is a concave function. The optimal (possibly infinite) value is reached when $a(X) = \mathrm{E}_u[x]$. Since $\mathrm{E}_u[x]$ is a monotone function of u, optimal value for $s(U \mid u)$ will be smaller or equal than $s(V \mid u)$. □

Lemma 7 *Let U and V be two segments such that $a(U) < a(V)$. Let r and t be two parameters, $r \geq t$. Then there exists u, $r \geq u \geq t$ such that $s(U \mid r) + s(V \mid t) \leq s(U \mid u) + s(V \mid u)$.*

Proof Let r^* be such that $s(U \mid r^*)$ is optimal, and define t^* similarly. Note that r^* or t^* can be infinite. Lemma 6 implies that $r^* \leq t^*$.

Assume that $r \leq t^*$. This implies that $t \leq r \leq t^*$. Since the score function is concave, $s(V \mid r) \geq s(V \mid t)$. Set $u = r$ to prove the lemma.

Assume that $r > t^*$. Set $u = \max(t^*, t)$. Since $r \geq u \geq r^*$, due to concavity of $s(U \mid u) \geq s(U \mid r)$ and by definition $s(V \mid u) \geq s(V \mid t)$. This proves the lemma. □

Corollary 2 *Let $\mathcal{U}$ be a K-segmentation. Let $r_1, \ldots, r_K$ be K parameters such that $r_i \geq r_{i-1}$. There exists a* monotone *K-segmentation $\mathcal{V} \preceq \mathcal{U}$ such that $s(\mathcal{V}) \geq s(\mathcal{U} \mid r_1, \ldots, r_K)$.*

Proof Assume that $\mathcal{U}$ is not monotone, that is, there exists U_i and U_{i+1} such that $a(U_i \setminus U_{i-1}) < a(U_{i+1} \setminus U_i)$. Lemma 7 implies that we can replace r_i and r_{i+1} with a common parameter, say u, and not decrease the score. We can now join the ith and $i+1$th segments into one segment with a parameter u and add a new empty corner at the beginning of the segmentation (with an infinitely large parameter). This modification makes the segmentation smaller w.r.t our order. We repeat this step until convergence. When converged, we have obtained a monotone K-segmentation whose score is at least as good as the original segmentation. □

We will now show that we can modify a segmentation containing a non-border.

Proposition 7 *Let $U_0, \ldots, U_K = \mathcal{U}$ be a monotone K-segmentation. Assume that there exist U_i, X, and Z such that $U_{i-1} \subseteq X \subsetneq U_i \subsetneq Z \subseteq U_{i+1}$ and $a(Z \setminus U_i) \geq a(U_i \setminus X)$.*

Then there exists a monotone K-segmentation $\mathcal{V}$ such that either $\mathcal{V} \subsetneq \mathcal{U}$ and $s(\mathcal{V}) \geq s(\mathcal{U})$ or $s(\mathcal{V}) > s(\mathcal{U})$.

In order to prove the proposition we will introduce some helpful notation. First, given two parameters r and t, we define

$$h(X; r, t) = s(X \setminus U_{i-1} \mid r) + s(U_{i+1} \setminus X \mid t) \quad .$$

We also define

$$g(l, \delta; r, t) = l(Z(r) - Z(t) + (r - t)\delta) \quad .$$

This function is essentially the difference between two scores.

Lemma 8 *Let $U_{i-1} \subseteq X \subsetneq Y \subseteq U_{i+1}$. We have $h(Y; r, t) - h(X; r, t) = g(|Y \setminus X|, a(Y \setminus X)\,; r, t)$.*

Proof Let us define $cs(Z) = \sum_{z \in Z} D(z)$. Note that

$$\begin{aligned} h(X; r, t) &= (|X| - |U_{i-1}|)Z(r) + r(cs(X) - cs(U_{i-1})) \\ &\quad + (|U_{i+1}| - |X|)Z(t) + t(cs(U_{i+1}) - cs(X)) \\ &= |X|(Z(r) - Z(t)) + (r - t)cs(X) + \text{const}, \end{aligned}$$

where const does not depend on X. Let us write $d = Z(r) - Z(t)$ and $Z = Y \setminus X$. This allows us to write

$$\begin{aligned} &h(Y; r, t) - h(X; r, t) \\ &\quad = |Y|d + (r - t)cs(Y) - |X|d - (r - t)cs(X) \\ &\quad = |Z|\big(d + (r - t)\frac{cs(Z)}{|Z|}\big) = g(|Y \setminus X|, a(Y \setminus X)\,; r, t) \quad . \end{aligned}$$

This completes the proof. □

Proof (of Proposition 7) Replace U_i with Z and let $\mathcal{U}^*$ be the resulting segmentation. Similarly, replace U_i with X and let $\mathcal{U}'$ be the resulting segmentation. Define $y = \sup_{r,t} h(U_i; r, t)$.

Fix $\epsilon > 0$. Lemma 6 implies that there exist $r_1, \ldots, r_K$ s.t.

$$r_{j-1} > r_j \text{ for } j = 2, \ldots, K \quad \text{and} \quad s(\mathcal{U}^* \mid r_1, \ldots, r_K) \geq s(\mathcal{U}) - \epsilon \quad .$$

From now on we will write $h(X)$ to mean $h(X; r_i, r_{i+1})$ and $g(k, \delta)$ to mean $g(k, \delta; r_i, r_{i+1})$. Note that we must have $h(U_i) \geq y - \epsilon$.

Assume that $h(Z) > y + \epsilon$. Then $s(\mathcal{U}^* \mid r_1, \ldots, r_K) > s(\mathcal{U} \mid r_1, \ldots, r_K) + \epsilon \geq s(\mathcal{U})$. Corollary 2 applied to $\mathcal{U}^*$ shows that there is a monotone segmentation with a score better than $s(\mathcal{U})$.

Assume that $h(Z) \leq y + \epsilon$. We must have $h(U_i) + \epsilon \geq y \geq h(Z) - \epsilon$ or, equivalently, $2\epsilon \geq h(Z) - h(U_i)$.

Define $\beta = a(Z \setminus U_i)$ and $\alpha = a(U_i \setminus X)$, $n = |Z \setminus U_i|$, $m = |U_i \setminus X|$. Define $c = n/m$. We now have

$$\begin{aligned} 2\epsilon &\geq h(Z) - h(U_i) = g(n, \beta) = cg(m, \beta) \\ &= cg(m, \alpha) + cm(r_i - r_{i+1})(\beta - \alpha) \geq cg(m, \alpha) \\ &= c(h(U_i) - h(X)) \geq c(y - \epsilon - h(X)), \end{aligned}$$

which implies $y - h(X) \leq \epsilon(1 + 2c^{-1}) \leq \epsilon(1 + 2|U_K|)$. Corollary 2 now implies that there exists a monotone segmentation $\mathcal{V}$ with $\mathcal{V} \precneqq \mathcal{U}$ such that $y - s(\mathcal{V}) \leq \epsilon(1 + 2|U_K|)$. Since this holds for any $\epsilon > 0$, we have proved the proposition. □

Proof (of Proposition 1) Assume that U_j is not a border, then Lemma 5 implies that there exist U_i, X, and Y such that the conditions in Proposition 7 are satisfied. Apply Proposition 7 to obtain a new monotone segmentation, $\mathcal{V}$. Reapply the step to $\mathcal{V}$ until $\mathcal{V}$ consists only of borders. This procedure terminates since at each step we either increase score or move segmentation is moved to the left w.r.t the partial order $\prec$. There are finite number of segmentations and no segmentation is visited twice, hence we converge to a segmentation consisting only of borders. □

B Proof of Proposition 3

Proof (of Proposition 3) Define $Z = \mathit{right}(U)$. Let us first prove that Z is a border.

Let $X = \mathit{left}(Z)$. If $U \setminus X \neq \emptyset$, then Lemma 4 implies that U is not a border. Hence $U \subseteq X$. Lemma 1 implies that $a(Z \setminus X) \geq a(Z \setminus U)$.

Let $Y \supsetneq Z$ be a corner, then Lemma 1 implies that $a(Y \setminus Z) < a(Z \setminus U) \leq a(Z \setminus X)$. By definition, $Z \setminus X$ has the smallest possible average. Consequently, Z is a border.

Assume that $Z \setminus V \neq \emptyset$, then Lemma 2 implies that V is not a border, which is a contradiction. Hence $Z \subseteq V$. Since V is the border next to U, we must have $Z = V$.

The proof in other direction is similar. □

C Proof of Proposition 6

Proof Let k be as in Proposition 5. Fix $U_j \in \mathit{brd}\left(T^k\right)$ with $j > 0$. Let us write V^i to be the portion of T^i that corresponds to the entries in $C = U_j \setminus U_{j-1}$. Since $\mathit{brd}\left(T^i\right) = \mathit{brd}\left(T^k\right)$ for $i > k$, it is enough to prove the result by showing that there exists m such that $V^m = V^{m+2}$. We will prove the result by induction over the size of C.

To that end, consider a DAG G where the nodes are the entries in C. Two distinct nodes (a, b), (c, d) are connected if and only if $a \leq c$ and $b \leq d$. Let p be a sink in G. Define V_p^k to be equal to V^k without the entry p and let V_p^i, for $i > k$, be the order obtained from V_p^{i-1} by simulating FINDORDER with w_F. Since p is a sink, it does not block any entries as FINDORDER updates the order. This implies that V_p^i is equal to V^i with p deleted for any $i \geq k$.

By the induction assumption there exists m_p such that $V_p^{m_p} = V_p^{m_p+2}$ for each sink p. Define $m = \max m_p$. Note that $V_p^m = V_p^{m+2}$ for each sink p.

Assume now that we have only one sink, say p. Then p will always be last entry in V^i and it follows immediately that $V^m = V^{m+2}$. We can safely assume that we have more than one sink.

Assume that we have exactly two sinks, say p and q. Assume that V_p^m does not end on q. Then it must be that V_p^m ends on a parent of p and p is the last entry in V^m. Since $V_p^m = V_p^{m+2}$, the proposition follows. Similar argument holds for V_q^m, V_p^{m+1}, and V_q^{m+1}. Hence, we can safely assume that V_p^m, V_q^m, V_p^{m+1}, and V_q^{m+1} all have p or q as their last entry. This implies that V^m, V^{m+1}, and V^{m+2}. have p and q as their last entries. Assume that p occurs before q in V^m. There will be a point during FINDORDER$(w_F(\cdot; T^m))$ when p and q are in the heap H at the same time, otherwise there will be an entry between p and q in V^{m+1}. This implies that q occurs before p in V^{m+1}. We apply the same argument to V^{m+1} to conclude that p occurs before q in V^{m+2}. This implies that $V^{m+1} = V^{m+2}$.

Assume that we have more than two sinks. Let p, q, and r be three sinks. We can deduce from V_r^m whether p occurs before q, or vice versa. Since we can do this for any sink triplet, we can deduce the order of sinks in V^m. The last sink, say p, will be the last in V^m. By definition of m, the order will be the same in V^{m+2} and the same sink will be also the last in V^{m+2}. Since $V_p^m = V_p^{m+2}$, the proposition follows. $\square$