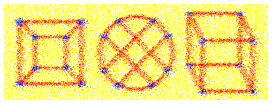

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 22, no. 2, pp. 297–327 (2018)
DOI: 10.7155/jgaa.00471

An I/O-Efficient Algorithm for Computing Vertex Separators on Multi-Dimensional Grid Graphs and Its Applications

Junhao Gan[1] *Yufei Tao*[2]

[1]University of Queensland, Australia
[2]Chinese University of Hong Kong, Hong Kong

Abstract

A vertex separator, in general, refers to a set of vertices whose removal disconnects the original graph into subgraphs possessing certain nice properties. Such separators have proved useful for solving a variety of graph problems. The core contribution of the paper is an I/O-efficient algorithm that finds, on any d-dimensional grid graph, a small vertex separator which mimics the well-known separator of [Maheshwari and Zeh, SICOMP'08] for planar graphs. We accompany our algorithm with two applications. First, by integrating our separators with existing algorithms, we strengthen the current state-of-the-art results of three fundamental problems on 2D grid graphs: finding connected components, finding single source shortest paths, and breadth-first search. Second, we show how our separator-algorithm can be deployed to perform density-based clustering on d-dimensional points I/O-efficiently.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
May 2017	January 2018	February 2018	June 2018	July 2018
		Published: August 2018		
	Article type: Regular paper		Communicated by: P. Ferragina	

E-mail addresses: j.gan@uq.edu.au (Junhao Gan) taoyf@cse.cuhk.edu.hk (Yufei Tao)

1 Introduction

Given an integer $d \geq 1$, a *d-dimensional grid graph* is an undirected graph $G = (V, E)$ with two properties:

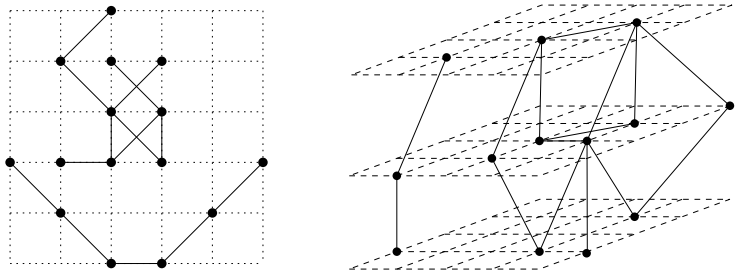
- Each vertex $v \in V$ is a distinct d-dimensional point in $\mathbb{N}^d$, where $\mathbb{N}$ represents the set of integers.
- If E has an edge between $v_1, v_2 \in V$, the two points v_1, v_2 must (i) be distinct (i.e., no self-loops), and (ii) differ by at most 1 in coordinate on *every* dimension.

See Figure 1 for two illustrative examples. We will limit ourselves to $d = O(1)$, under which a d-dimensional grid graph is *sparse*, that is, $|E| = O(|V|)$, because each vertex can have a degree at most $3^d = O(1)$.

Past research on grid graphs has largely focused on $d = 2$, mainly motivated by the practical needs to work with *terrains* [2, 5, 6], also known as *land surfaces* [8, 20, 26, 31]. A terrain or land surface is essentially a function $f : \mathbb{R}^2 \to \mathbb{R}$ that maps every point on the earth's longitude-altitude plane to an elevation. To represent the function approximately, the plane is discretized into a grid, such that functional values are stored only at the grid points. Real-world networks (of, e.g., roads, rail-ways, rivers, etc.) are represented by "atom" line segments each of which connects two points v_1, v_2 in $\mathbb{N}^2$ whose coordinates differ by at most 1 on each dimension. The atom segment is augmented with a weight, equal to the Euclidean distance between the 3D points v_1' and v_2', where v_1' has the same x- and y-coordinates as v_1, and has z-coordinate $f(v_1)$ (v_2' is obtained from v_2 similarly). The modeling gives a 2D grid graph where an atom segment becomes a weighted edge. A variety of topics — e.g., *flow analysis* [5, 6], *nearest-neighbor queries* [8, 26, 31], and *navigation* [20] — have been studied on gigantic networks which may not fit in the main memory of a commodity machine. Crucial to the solutions in [5, 6, 8, 20, 26, 31] are algorithms settling fundamental problems (such as finding connected components, finding single-source shortest paths, and breadth-first search, etc.) on massive 2D grid graphs I/O-efficiently.

On the other hand, d-dimensional grid graphs of $d \geq 3$ seem to have attracted less attention, maybe because few relevant applications have been identified in practice ([25] is the only work on grid graphs of $d \geq 3$ we are aware of, but no concrete applications were described there). We will fill the void in this paper by elaborating on an inherent connection between such graphs and density-based clustering.

The main objective of our work is to understand how a grid graph can be I/O-efficiently decomposed using "vertex separators" that are reminiscent of the well-known vertex separators on planar graphs [12, 19, 21]. In particular, the separator of Maheshwari and Zeh [21] can be found I/O-efficiently, and has proved to be extremely useful in solving many problems on planar graphs with small I/O cost (see, e.g., [4, 15, 21, 32]). This raises the hope that similar separators on grid graphs would lead to I/O-efficient algorithms on those problems as well (note that grid graphs are not always planar, even in 2D space). Following [21], we focus on vertex separators defined as follows:



(a) A 2-dimensional grid graph (b) A 3-dimensional grid graph

Figure 1: Multidimensional grid graphs

Definition 1 *Let $G = (V, E)$ be a d-dimensional grid graph with $d = O(1)$. Given a positive integer $r \leq |V|$, a set $S \subseteq V$ is an **r-separator** of G if*

1. $|S| = O(|V|/r^{1/d})$
2. *Removing the vertices in S disconnects G into $h = O(|V|/r)$ subgraphs $G_1 = (V_1, E_1)$, ..., $G_h = (V_h, E_h)$, such that for each $i \in [1, h]$:*
   (a) $|V_i| = O(r)$;
   (b) *The vertices of V_i are adjacent to $O(r^{1-1/d})$ vertices of S.*

*The subgraphs $G_1, ..., G_h$ are said to be **induced** by S.* □

Previous work [23, 28] has shown that such vertex separators definitely exist for any $r \in [1, |V|]$. The r-separators of [23, 28] are constructed by repetitively partitioning a d-dimensional grid graph with "surface cuts". More specifically, such a cut is performed with a closed d-dimensional surface (which is a sphere in [23] and an axis-parallel rectangle in [28]). All vertices near the surface are added to the separator, while the process is carried out recursively inside and outside the surface, respectively. However, it still remains as a non-trivial open problem how to find the separators of [23, 28] I/O-efficiently.

For grid graphs of $d = 2$, the existence of an r-separator is implied by the planar separator of [21], as shown in [15]. The separator of [21] can be computed I/O-efficiently (and hence, so can an r-separator of a 2D graph), subject to a constraint on the size of the main memory. We will discuss the issue further in Section 1.2.

1.1 Computation Model

We will work with the *external memory* (EM) computation model of [3], which is the de facto model for studying I/O-efficient algorithms nowadays. In this model, the machine is equipped with M words of (internal) memory, and a disk that has been formatted into *blocks*, each of which has B words. The values of M and B satisfy $M \geq 2B$. An *I/O* either reads a block of data from

the disk into memory, or writes B words of memory into a disk block. The *cost* of an algorithm is measured in the number of I/Os performed. Denote by $sort(n) = \Theta((n/B)\log_{M/B}(n/B))$ the I/O complexity of sorting n elements [3].

1.2 Our Results

Let $G = (V, E)$ be a d-dimensional grid graph. As mentioned, the existence of r-separators of G is already known [21, 23, 28]. Our construction, however, uses ideas different from those of [21, 23, 28]. Interestingly, as a side product, our proof presents a new type of r-separators that can be obtained by a recursive binary orthogonal partitioning of $\mathbb{N}^d$. To formalize this, we introduce:

Definition 2 *Let $G = (V, E)$ be a d-dimensional grid graph. An **orthogonal partitioning** of G is a pair $(S, \mathcal{G})$ made by a subset S of V and a set $\mathcal{G}$ of subgraphs of G, such that $(S, \mathcal{G})$ satisfies either of the conditions below:*

1. *$(S, \mathcal{G}) = (\emptyset, \{G\})$.*
2. *$(S, \mathcal{G}) = (S_0 \cup S_1 \cup S_2, \mathcal{G}_1 \cup \mathcal{G}_2)$ where:*
   - *(a) S_0 is the set of vertices on some plane π satisfying:*
     - *π is perpendicular to one of the d dimensions;*
     - *V has vertices on both sides of π.*
   - *(b) $(S_1, \mathcal{G}_1)$ and $(S_2, \mathcal{G}_2)$ are orthogonal partitionings of G_1 and G_2 respectively, where G_1 and G_2 are the subgraphs of G induced by the vertices on the two sides of π, respectively.*

Note that since, in the second bullet, G_1 and G_2 have at least one less vertex than G, the recursive definition is well defined (see Figure 2 for an illustration). It is worth pointing out that, every vertex of V appears either in S, or exactly one of the subgraphs in $\mathcal{G}$.

Consider any r-separator S of G, and the set $\mathcal{G}$ of subgraphs induced by S. We call S an *orthogonal r-separator* of G if $(S, \mathcal{G})$ is an orthogonal partitioning. The first main result of the paper is:

Theorem 1 *Let $G = (V, E)$ be a d-dimensional grid graph where d is a fixed constant. G has an orthogonal r-separator for any integer $r \in [1, |V|]$.*

The above is not subsumed by the existence results of [21, 23, 28] because the vertex separators in [21, 23, 28] are not orthogonal. Our proof of the theorem is constructive, and can be implemented efficiently to obtain our second main result:

Theorem 2 *Let $G = (V, E)$ be a d-dimensional grid graph where d is a fixed constant. For any values of M, B satisfying $M \geq 2B$, there is an algorithm that computes in $O(sort(|V|))$ I/Os an M-separator S of G, as well as the $O(|V|/M)$ subgraphs induced by S.*

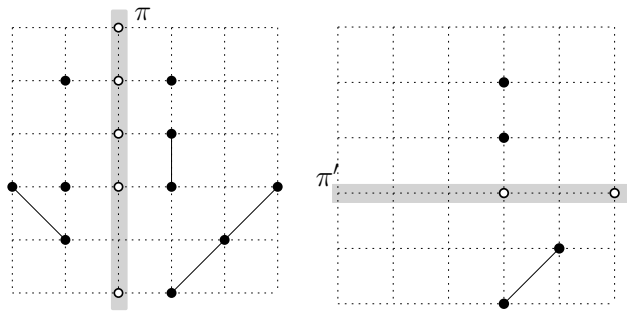


Figure 2: (a) shows a plane π on the grid graph G of Figure 1a; let S be the set of white vertices, and G_1 (resp., G_2) the subgraph induced by the black vertices on the left (resp., right) of π. $(S, \{G_1, G_2\})$ forms an orthogonal partitioning of G. (b) shows another plane π' on G_2; let S' be the set of white vertices on π', and G_3 (resp., G_4) the subgraph induced by the black vertices above (resp., below) of π'. $(S', \{G_3, G_4\})$ forms an orthogonal partitioning of G_2. Furthermore, $(S \cup S', \{G_1, G_3, G_4\})$ is also an orthogonal partitioning of G.

It is notable that our algorithm in Theorem 2 works for all M, B satisfying $M \geq 2B$. When $d = 2$, an M-separator can also be computed in $O(sort(n))$ I/Os using the planar-separator algorithm of [21]. However, the algorithm of [21] requires the *tall-cache assumption* of $M \geq B^2$ (when this assumption is not true, the I/O cost of the algorithm is substantially larger). This difference is what permits us to strengthen a number of existing results on 2D grid graphs, as will be explained later. Remember, also, that the algorithm of [21] cannot be applied to grid graphs of $d \geq 3$.

Next, we will explain some new results made possible by our new algorithm.

1.2.1 Application 1: New Results on Grid Graphs

Single Source Shortest Path and Breadth First Search on 2D Grid Graphs. As mentioned, an M-separator of 2D grid graphs can be obtained using the planar-graph algorithm of [21]. This is a key step behind the state-of-the-art algorithms for solving the *single source shortest path* (SSSP) and the *breadth first search* (BFS) on 2D grid graphs I/O-efficiently. However, since the algorithm of [21] is efficient only under the tall-cache assumption $M \geq B^2$, the same assumption is inherited by the SSSP and BFS algorithms as well. Our Theorem 2 remedies this defect by removing the tall-cache assumption altogether.

Specifically, for SSSP, we will prove:

Corollary 1 *The single source shortest path (SSSP) problem on a 2D grid graph $G = (V, E)$ can be solved in $O(|V|/\sqrt{M} + sort(|V|))$ I/Os.*

Previously, the state of the art was an algorithm in [15] that matches the

performance guarantee of Corollary 1 under the tall cache assumption. For $M = o(B^2)$, however, the I/O-complexity of [15] becomes $O((|V|/\sqrt{M}) \cdot \log_M |V|)$, which we strictly improve. It is worth mentioning that, on a general undirected graph $G = (V, E)$, the fastest SSSP algorithm [18] in EM to our knowledge requires $O(|V| + \frac{|E|}{B} \log_2 \frac{|E|}{B})$ I/Os, which is much worse than the bound in Corollary 1.

For BFS, we will prove:

Corollary 2 *We can perform breadth first search (BFS) on a 2D grid graph $G = (V, E)$ in $O(|V|/\sqrt{M} + sort(|V|))$ I/Os.*

The corollary nicely bridges the previous state of the art, which runs either the SSSP algorithm of [15], or the best BFS algorithm [22] for general graphs. When applied to a 2D grid graph $G = (V, E)$, the algorithm of [22] performs $O(|V|/\sqrt{B} + sort(|V|))$ I/Os. Corollary 2 improves the winner of those two algorithms when M is between $\omega(B)$ and $o(B^2)$.

For fairness, it should be pointed out that the work of [21] focused on studying the smallest memory size needed to achieve $O(sort(n))$ in computing vertex separators for planar graphs. A topic, which was not explored in [21] but is relevant to us, is the explicit I/O complexity of the algorithm in [21] when M is in the range from $2B$ to B^2. It appears that the techniques of [21] could be adapted to compute an M-separator on 2D grid graphs in $O(|V|/\sqrt{M} + sort(|V|))$ I/Os for all $M \geq 2B$. If so, then Corollaries 1 and 2 can already be achieved with the current state of the art [15]. We include our own proofs for the two corollaries anyway because (i) the proofs are short, and make the claims official; (ii) they indicate that, for $M = o(B^2)$, the performance bottleneck is not on computing an M-separator (our algorithm finds an M-separator in $O(sort(n))$ I/Os); and (iii) they explain the details unique to our M-separator when it comes to integration with the existing SSSP/BFS algorithms.

Finding Connected Components on d-Dimensional Grid Graphs. It has been stated [30, 33] that the *connected components* (CCs) of a 2D grid graph $G = (V, E)$ can be computed in $O(sort(|V|))$ I/Os. This is based on the belief that a 2D grid graph has the property of being *sparse under edge contractions*. Specifically, an *edge contraction* removes an edge between vertices v_1, v_2 from G, combines v_1, v_2 into a single vertex v, replaces every edge adjacent to v_1 or v_2 with an edge adjacent to v, and finally removes duplicate edges thus produced (see Figure 3); all these steps then create a new graph. The aforementioned property says that, if one performs any sequence of edge contractions to obtain a resulting graph $G' = (V', E')$, G' must still be sparse, namely, $|E'| = O(|V'|)$. Surprisingly, the belief — perhaps too intuitive — seemed to have been taken for granted, such that no proof has ever been documented.

We will formally *disprove* this belief:

Theorem 3 *There exists a 2D grid graph that is not sparse under edge contractions.*

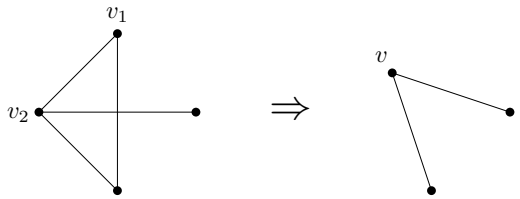


Figure 3: Contracting the edge between v_1, v_2 from the graph on the left produces the graph on the right

With the belief invalidated, the best existing deterministic algorithm for computing the CCs of a 2D grid graph requires an I/O complexity that is the minimum of $O(sort(|V|) \cdot \log\log B)$ [24] and $O(sort(|V|) \cdot \log(|V|/|M|))$ [16]. Equipped with Theorem 2, we will improve this result by proving:

Corollary 3 *The connected components of a d-dimensional grid graph $G = (V, E)$ where $d = O(1)$ can be computed in $O(sort(|V|))$ I/Os for all constant $d \geq 2$.*

Note that the above corollary applies not only to $d = 2$, but also to any constant $d \geq 2$.

1.2.2 Application 2: Density-Based Clustering

Density-based clustering is an important class of problems in data mining (see textbooks [14, 29]), where DBSCAN [11] is a well-known representative. The input of the DBSCAN problem consists of:

- A constant integer $minPts \geq 1$,
- A real number $\epsilon > 0$, and
- A set P of n points in $\mathbb{R}^d$, where $\mathbb{R}$ denotes the set of real values, and the dimensionality d is a constant integer at least 2.

Denote by $dist(p_1, p_2)$ the distance between two points p_1 and p_2, according to a certain distance metric. A point $p \in P$ is a *core point* if $|\{q \in P | dist(p, q) \leq \epsilon\}| \geq minPts$; otherwise, p is a *non-core point*. Define a *neighbor core graph* G as follows: (i) each vertex of G corresponds to a distinct core point, and (ii) there is an edge between two vertices (a.k.a, core points) p_1, p_2 if and only if $dist(p_1, p_2) \leq \epsilon$. Then, the clusters of P are uniquely determined in two steps:

1. Take each connected component of G as a cluster. After this step, each cluster contains only core points.
2. For each non-core point $p \in P$, consider every core point q satisfying $dist(p, q) \leq \epsilon$; assign p to the (only) cluster that contains q. This may add p to $minPts = O(1)$ clusters.

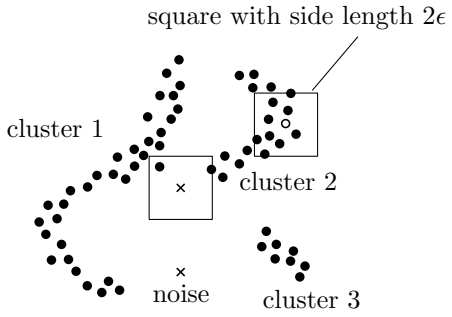


Figure 4: The square on the right illustrates the value of ϵ (all the points in the square are within L_∞ distance ϵ from the white point) and $minPts = 4$. All the circle points are core points, while the two cross points are non-core points. One non-core point is assigned to both Cluster 1 and Cluster 2, while the other non-core point is classified as noise.

The clusters after Step 2 constitute the final clusters on P. It is possible that some non-core points are not assigned to any clusters; those points are classified as *noise*. The goal of the *DBSCAN problem* is to compute the DBSCAN clusters on the input set P with respect to the parameters ϵ and $minPts$.

Figure 4 illustrates an example where the distance metric is the L_∞ norm. Note that there can be $\Omega(n^2)$ edges in G (for simplicity, no edges are given in the example, but the square as shown should make it easy to imagine which edges are present). Thus, one should not hope to solve the problem efficiently by materializing all the edges.

We will prove:

Theorem 4 *For any fixed-dimensionality d, the DBSCAN problem under the L_∞ norm can be solved in*

- $O(sort(n))$ *I/Os for $d = 2$ and 3;*
- $O((n/B)\ \log^{d-2}_{M/B}(n/B))$ *for any constant $d \geq 4$.*

Our proof relies on the proposed separator algorithm in Theorem 2, and manifests on the usefulness of d-dimensional grid graphs in algorithm design.

It is worth mentioning the DBSCAN problem is known to be hard under the L_2 norm: it demands $\Omega(n^{4/3})$ time to solve for $d \geq 3$ [13], unless Hopcroft's problem[1] [10] could be solved in $o(n^{4/3})$ time, which is commonly believed to be impossible [9, 10]. Consequently, the L_2 norm is unlikely to admit an EM algorithm with near linear I/O complexity (otherwise, one could obtain an efficient RAM algorithm by setting M and B to constants). Theorem 4, therefore, separates the L_∞ norm (and hence, also the L_1 norm) from the L_2 norm, subject to the above hardness assumption on Hopcroft's problem.

[1]Let S_{pt} be a set of points, and S_{line} be a set of lines, all in $\mathbb{R}^2$. Hopcroft's problem is to determine whether there is a point in S_{pt} that lies on some line of S_{line}.

1.3 Paper Organization

The rest of the paper is organized as follows. Section 2 gives a constructive proof to show the existence of a new class of r-separators. Section 3 describes an algorithm for computing an M-separator in $O(sort(n))$ I/Os. Section 4 presents our algorithm for solving the DBSCAN problem under the L_∞ norm in EM, and as a side product, also an algorithm for finding the CCs of a d-dimensional grid graph. Section 5 proves the other results on grid graphs mentioned in Section 1.2.1. Finally, Section 6 concludes the paper with some open questions.

2 Orthogonal Separators

This section is devoted to establishing Theorem 1. We will explain our proof in four steps, each of which is presented in a different subsection.

Let $G = (V, E)$ be a d-dimensional grid graph, and $(S, \mathcal{G})$ be an orthogonal partitioning of G. Consider any subgraph $G' \in \mathcal{G}$. A vertex v in G' is a *boundary vertex* of G' if v is adjacent in G to at least one vertex in S. Define the *minimum bounding box* of G' — denoted as $MBB(G')$ — as the smallest d-dimensional axis-parallel rectangle that contains all the vertices of G'. The fact that G is a grid graph implies that all boundary vertices of G' must be on the boundary faces of $MBB(G')$.

2.1 A Binary Partitioning Lemma

Recall that an r-separator can be *multi-way* because it may induce any number $h = O(|V|/r)$ of subgraphs. Let us first set $h = 2$, and prove the existence of a binary orthogonal separator:

Lemma 1 *Let $G = (V, E)$ be a d-dimensional grid graph satisfying*

$$|V| \geq 2^d \cdot (2d + 1)^{d+1}.$$

There exists an orthogonal partitioning $(S, \{G_1, G_2\})$ of G such that:

- $|S| \leq (2d + 1)^{1/d} \cdot |V|^{1-1/d}$
- *G_1 and G_2 each have at least $|V|/(4d + 2)$ vertices.*

Proof: Given a point $p \in \mathbb{N}^d$, denote by $p[i]$ its coordinate on dimension $i \in [1, d]$. Given a vertex $v \in V$, an integer x, and a dimension i, we say that v is *on the left of x on dimension i* if $v[i] < x$, and similarly, *on the right of x on dimension i* if $v[i] > x$. We define the *V-occupancy of x on dimension i* as the number of vertices $v \in V$ satisfying $v[i] = x$.

To prove Lemma 1, our strategy is to identify an integer x and a dimension i such that (i) the V-occupancy of x on dimension i is at most $(2d+1)^{1/d} \cdot |V|^{1-1/d}$, and (ii) there are at least $|V|/(4d+2)$ points on the left and right of x on dimension i, respectively. Choosing (i) S as the set of vertices $v \in V$ with $v[i] = x$, and (ii)

G_1 (resp., G_2) as the graph induced by the vertices on the left (resp., right) of x on dimension i will satisfy the lemma — in this case, we say that a *split* is performed using a plane perpendicular to dimension i. We will prove that such a pair of x and i definitely exists.

For each $j \in [1, d]$, define y_j to be the largest integer y such that V has at most $|V|/(2d+1)$ vertices on the left of y on dimension j, and similarly, z_j to be the smallest integer z such that V has at most $|V|/(2d+1)$ vertices on the right of z on dimension j. It must hold that $y_j \leq z_j$.

Consider the axis-parallel box whose projection on dimension $j \in [1, d]$ is $[y_j, z_j]$. By definition of y_j, z_j, the box must contain at least

$$|V| \left(1 - \frac{2d}{2d+1}\right) = |V| \cdot \frac{1}{2d+1}$$

vertices. This implies that the box must contain at least $|V|/(2d+1)$ points in $\mathbb{N}^d$, that is:

$$\prod_{j=1}^{d} (z_j - y_j + 1) \quad \geq \quad \frac{|V|}{2d+1}$$

Therefore, there is at least one j satisfying

$$z_j - y_j + 1 \quad \geq \quad \left(\frac{|V|}{2d+1}\right)^{1/d}.$$

Set i to this j. Since the box can contain at most $|V|$ vertices, there is one integer $x \in [y_i, z_i]$ such that the V-occupancy of x on dimension i is at most

$$\frac{|V|}{|V|^{1/d}/(2d+1)^{1/d}} \quad = \quad (2d+1)^{1/d} \cdot |V|^{1-1/d}.$$

We now argue that there must be at least $|V|/(4d+2)$ vertices on the left of x on dimension i. For this purpose, we distinguish two cases:

- $x = y_i$: By definition of y_i and x, the number of vertices on the left of x on dimension i must be at least

  $$\frac{|V|}{2d+1} - (2d+1)^{1/d} \cdot |V|^{1-1/d}$$

  which is at least $|V|/(4d+2)$ for $|V| \geq 2^d(2d+1)^{d+1}$.

- $x > y_i$: By definition of y_i, there are at least $|V|/(2d+1)$ vertices whose coordinates on dimension i are at most y_i. All those vertices are on the left of x on dimension i.

A symmetric argument shows that at least $|V|/(4d+2)$ vertices are on the right of x on dimension i. This finishes the proof of Lemma 1. □

2.2 A Multi-Way Partitioning Lemma

In this subsection, we establish a multi-way version of the previous lemma:

Lemma 2 *Let $G = (V, E)$ be a d-dimensional grid graph. For any positive integer r satisfying*

$$2^d \cdot (2d+1)^{d+1} \leq r \tag{1}$$

G has an orthogonal partitioning $(S, \mathcal{G})$ such that $|S| = O(|V|/r^{1/d})$ and $\mathcal{G}$ has $O(|V|/r)$ subgraphs, each of which has at most r vertices.

Proof: Motivated by [12], we perform the binary split enabled by Lemma 1 recursively until every subgraph has at most r vertices. This defines an orthogonal partitioning $(S, \mathcal{G})$ as follows. At the beginning, $S = \emptyset$ and $\mathcal{G} = \{G\}$. Every time Lemma 1 performs a split on a subgraph $G' \in \mathcal{G}$, it outputs an orthogonal partitioning $(S', \{G_1, G_2\})$ of G'; we update $(S, \mathcal{G})$ by (i) adding all the vertices of S' into S, (ii) deleting G' from $\mathcal{G}$, and (iii) adding G_1, G_2 to $\mathcal{G}$.

Focus now on the final $(S, \mathcal{G})$. Each subgraph in $\mathcal{G}$ has at least $(r+1)/(4d+2) = \Omega(r)$ vertices because each application of Lemma 1 is on a subgraph of at least $r+1$ vertices. It thus follows that the number of subgraphs in $\mathcal{G}$ is $O(|V|/r)$.

It remains to show $|S| = O(|V|/r^{1/d})$. For this purpose, define function $f(n)$ which gives the maximum possible $|S|$ when the original graph has $n = |V|$ vertices. If $\frac{r}{4d+2} \leq n \leq r$, $f(n) = 0$. Otherwise, Lemma 1 indicates

$$f(n) \quad \leq \quad (2d+1)^{1/d} \cdot n^{1-1/d} + \max_{\alpha \in [\frac{1}{4d+2}, \frac{4d+1}{4d+2}]} f(\alpha n) + f((1-\alpha)n).$$

It is not difficult to verify (by the substitution method [7]) that $f(n) = O(n/r^{1/d})$ for $n > r$. □

Note that the lemma does not necessarily yield an r-separator because the set S produced may not satisfy Condition 2(b) in Definition 1.

2.3 Binary Partitioning with Colors

We say that a d-dimensional grid graph $G = (V, E)$ is *r-colored* if

- $|V| \leq r$;
- Every vertex in V is colored either black or white;
- There are at least $8d^2 \cdot r^{1-1/d}$ black vertices, all of which are on the boundary faces of $MBB(G)$.

Next, we prove a variant of Lemma 1, which concentrates on splitting only the black vertices evenly (recall that Lemma 1 aims at an asymptotically even split of all the vertices):

Lemma 3 *Let $G = (V, E)$ be an r-colored d-dimensional grid graph with b black vertices. There is an orthogonal partitioning $(S, \{G_1, G_2\})$ of G satisfying:*

- $|S| \leq r^{1-1/d}$.
- *G_1 and G_2 each have at least $\frac{b}{8d^2}$ black vertices.*

Proof: We will adopt the strategy in Section 2.1 but with extra care. Since $MBB(G)$ has $2d$ faces, one of them contains at least $b/(2d)$ black vertices. Fix R to be this face, which is a $(d-1)$-dimensional rectangle. Assume, without loss of generality, that R is orthogonal to dimension d.

For each $j \in [1, d-1]$, define y_j to be the largest integer y such that R has at most $\frac{b}{2d} \cdot \frac{1}{2d}$ black vertices on the left of y on dimension j, and similarly, z_j to be the smallest integer z such that R has at most $\frac{b}{2d} \cdot \frac{1}{2d}$ black vertices on the right of z on dimension j. It must hold that $y_j \leq z_j$.

Consider the axis-parallel box in R whose projection on dimension $j \in [1, d-1]$ is $[y_j, z_j]$. By definition of y_j, z_j, the box must contain at least

$$\frac{b}{2d}\left(1 - \frac{2(d-1)}{2d}\right) = \frac{b}{2d} \cdot \frac{1}{d}$$

black vertices. Therefore, there is at least one dimension $j \in [1, d-1]$ on which the projection of the box covers at least

$$\left(\frac{b}{2d^2}\right)^{1/(d-1)}$$

coordinates. Set i to this j. Since the box can contain at most $|V| \leq r$ vertices, there is one integer $x \in [y_i, z_i]$ such that the V-occupancy of x on dimension i is at most

$$\frac{r}{b^{1/(d-1)}} \cdot (2d^2)^{1/(d-1)} \leq r^{1-1/d} \leq \frac{b}{8d^2} \tag{2}$$

where both inequalities used $b \geq 8d^2 \cdot r^{1-1/d}$ (which is true because G is r-colored).

We perform a split perpendicular to dimension i at x; namely, choose S as the set of vertices $v \in V$ with $v[i] = x$, and set G_1 (resp., G_2) to be the subgraph induced by the vertices on the left (resp., right) of x on dimension i. To show that G_1 has at least $\frac{b}{8d^2}$ black vertices, we distinguish two cases:

- $x = y_i$: By the definitions of y_i and x, the number of black vertices on the left of x on dimension i must be at least
$$\frac{b}{4d^2} - \frac{r}{b^{1/(d-1)}} \cdot (2d^2)^{1/(d-1)} \geq \frac{b}{4d^2} - \frac{b}{8d^2} = \frac{b}{8d^2} \tag{3}$$
where the first inequality is due to (2).
- $x > y_i$: The definitions of y_i and x imply at least $\frac{b}{4d^2}$ black vertices on the left of x on dimension i.

A symmetric argument shows that G_2 must have at least $\frac{b}{8d^2}$ black vertices. This completes the proof of Lemma 3. □

2.4 Existence of Orthogonal Separators (Proof of Theorem 1)

We are now ready to prove Theorem 1. It suffices to do so for $r \geq 2^d \cdot (2d+1)^{d+1}$, because an orthogonal $(2^d \cdot (2d+1)^{d+1})$-separator is also a valid orthogonal r-separator for any $r < 2^d \cdot (2d+1)^{d+1}$ when $d = O(1)$. The following discussion concentrates on $r \geq 2^d \cdot (2d+1)^{d+1}$.

Let $G = (V, E)$ be the input d-dimensional grid graph. First, apply Lemma 2 on G to obtain an orthogonal partitioning $(S, \mathcal{G})$. The lemma ensures that $|S| = O(|V|/r^{1/d})$ and that each of the $O(|V|/r)$ subgraphs in $\mathcal{G}$ has at most r vertices. We say that a subgraph in $\mathcal{G}$ is *bad* if it has more than

$$8d^2 \cdot 3^{d-1} \cdot r^{1-1/d}$$

boundary vertices. We refer to each bad subgraph in $\mathcal{G}$ at this moment as a *raw bad subgraph* (the content of $\mathcal{G}$ may change later).

Motivated by [12], we deploy Lemma 3 to get rid of all the bad subgraphs with an *elimination procedure*. As long as $\mathcal{G}$ still has at least one bad subgraph, the procedure removes an arbitrary bad subgraph G_{bad} from $\mathcal{G}$, and executes the following steps on it:

1. Color all the boundary vertices of G_{bad} black, and the other vertices white. G_{bad} thus becomes r-colored (by definition of bad subgraph).
2. Apply Lemma 3 to find an orthogonal partitioning $(S', \{G_1, G_2\})$ of G_{bad}.
3. Add all the vertices in S' to S. Delete G_{bad} from $\mathcal{G}$, and add G_1, G_2 to $\mathcal{G}$. Note that $(S, \mathcal{G})$ still remains as an orthogonal partitioning of G.

When $\mathcal{G}$ has no more bad subgraphs, we return the set S of the current $(S, \mathcal{G})$.

Next, we show that the final S obtained is an orthogonal r-separator, namely: (i) $|S| = O(|V|/r^{1/d})$, (ii) the final $\mathcal{G}$ has $O(|V|/r)$ subgraphs, and (iii) every subgraph in $\mathcal{G}$ has $O(r^{1-1/d})$ boundary vertices. The elimination procedure already guarantees (iii); the rest of the section will focus on proving (i) and (ii).

Denote by $(S_{before}, \mathcal{G}_{before})$ the content of $(S, \mathcal{G})$ before the elimination procedure, while still using $(S, \mathcal{G})$ to denote the orthogonal partitioning at the end. $\mathcal{G}_{before}$ has $O(|V|/r)$ subgraphs. Some of those subgraphs are also in $\mathcal{G}$. Every "new" subgraph in $\mathcal{G}$ but not in $\mathcal{G}_{before}$ must be created during the elimination procedure. We can think of the subgraph creation during the elimination procedure as a forest. Each tree in the forest is rooted at a raw bad subgraph; and every node in the tree corresponds to a subgraph created in the elimination procedure. Every internal node of a tree has two child nodes, corresponding to the splitting of a subgraph G_{bad} into G_1, G_2 at Step 2. Each leaf of a tree is a subgraph in $\mathcal{G}$. The next lemma bounds the size of each tree:

Lemma 4 *Let G_{raw} be a raw bad subgraph with b boundary vertices. The elimination procedure generates $O(b/r^{1-1/d})$ leaf subgraphs in the tree rooted at G_{raw}.*

Proof: Let G_{bad} be a bad subgraph that is split with Lemma 3 in the elimination procedure. Define $f(n)$ as the maximum number of leaf subgraphs in the subtree rooted at G_{bad}, when G_{bad} has n boundary vertices.

If $n \leq 8d^2 \cdot 3^{d-1} \cdot r^{1-1/d}$, $f(n) = 1$. Now assume $n > 8d^2 \cdot 3^{d-1} \cdot r^{1-1/d}$. An application of Lemma 3 on G_{bad} generates G_1 and G_2. Let us analyze the number of boundary vertices that G_1 can have. Every boundary vertex of G_1 may be (i) inherited from G_{bad}, or (ii) newly created during the split performed by Lemma 3. The second bullet of Lemma 3 shows that there can be at most $\alpha \cdot n$ vertices of type (i), for some $\alpha \in [\frac{1}{8d^2}, 1 - \frac{1}{8d^2}]$. As for (ii), note that every vertex of this type must be adjacent to some vertex of the vertex set in the first bullet of Lemma 3. Since G_{bad} is r-colored, the number of vertices of type (ii) must be at most $3^{d-1} \cdot r^{1-1/d}$ (the vertex set in the first bullet of Lemma 3 has size $r^{1-1/d}$, while each vertex in that set has at most 3^{d-1} neighbors in G_1). After extending the analysis to G_2, we obtain the following recurrence:

$$f(n) \leq \max_{\alpha \in [\frac{1}{8d^2}, 1-\frac{1}{8d^2}]} \Big(f\big(\alpha n + 3^{d-1} \cdot r^{1-1/d}\big) + f\big((1-\alpha)n + 3^{d-1} \cdot r^{1-1/d}\big) - 1 \Big).$$

It is not difficult to verify (with the substitution method [7]) that $f(n) = O(n/r^{1-1/d})$. The lemma then follows by setting $n = b$. □

Suppose that there are h' raw bad subgraphs. Let b_i ($1 \leq i \leq h'$) be the number of boundary vertices of the i-th raw bad subgraph. From Lemma 2 and by the fact that each vertex in a d-dimensional grid graph has degree $O(1)$, we know

$$\sum_{i=1}^{h'} b_i \quad = \quad O(|V|/r^{1/d}).$$

Combining this with Lemma 4 shows that the elimination procedure introduces at most

$$O\left(\frac{|V|}{r^{1/d}} \cdot \frac{1}{r^{1-1/d}}\right) = O(|V|/r)$$

new subgraphs. Therefore, in total there are $h' + O(|V|/r) = O(|V|/r)$ subgraphs in $\mathcal{G}$ at the end of the elimination procedure.

The above analysis also indicates that the elimination procedure can apply Lemma 3 no more than $O(|V|/r)$ times, each of which adds $O(r^{1-1/d})$ vertices into S. Therefore, the final separator S has size at most

$$|S_{before}| + O\left(\frac{|V|}{r} \cdot r^{1-1/d}\right) \quad = \quad O(|V|/r^{1/d}).$$

This concludes the whole proof of Theorem 1.

3 I/O-Efficient Separator Computation

This section will prove Theorem 2 by giving an algorithm to construct an M-separator. Our proof is essentially an efficient implementation of the strategy explained in Section 2 for finding an orthogonal M-separator. Recall that the strategy involves two phases: (i) Lemma 2, and (ii) the elimination procedure in Section 2.4. The second phase, as far as algorithm design is concerned, is trivial. Every subgraph produced by the first phase has — by definition of M-separator — $O(M)$ edges, which can therefore be loaded into memory so that the algorithm in Section 2.4 runs with no extra I/Os. In other words, the second phase can be accomplished in only $O(|V|/B)$ I/Os.

Henceforth, we will focus exclusively on the first phase, assuming

$$M \geq 2^d \cdot (4d+2)^{2d} \cdot B. \tag{4}$$

Note that this assumption is made without loss of generality as long as d is a constant. It is folklore that, in general, any algorithm assuming $M \geq cB$ for any constant $c > 2$ can be adapted to work under $M \geq 2B$, with only a constant blowup in the I/O cost.

The construction algorithm of Lemma 2 recursively applies binary splits to the input graph until all the obtained subgraphs have at most M vertices. This process can be imagined as a *split tree*, where $G = (V, E)$ is the parent of $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ if the splitting of G spawns G_1 and G_2. The split is *balanced* in the sense that both $|V_1|$ and $|V_2|$ are at least $|V|/(4d+2)$. Hence, the split tree has a height of $O(\log(|V|/M))$.

3.1 One Split

In this subsection, we describe an algorithm that performs a single split on a d-dimensional grid graph $G = (V, E)$ with $|V| > M$ using *sublinear* I/Os, assuming certain preprocessing has been done. This algorithm will play an essential role in our final solution.

Recall that, given a coordinate x on dimension $i \in [1, d]$, the V-occupancy of x is the number of vertices $v \in V$ with $v[i] = x$. We now extend this concept to an interval $\sigma = [x_1, x_2]$ on dimension i: the *average V-occupancy* of σ equals

$$\frac{|\{v \in V \mid v[i] \in \sigma\}|}{x_2 - x_1 + 1}.$$

Preprocessing Assumed. Prior to invoking the algorithm below, each dimension $i \in [1, d]$ should have been partitioned into at most s disjoint intervals — called *slabs* — where

$$s = (M/B)^{1/d}. \tag{5}$$

A slab σ of dimension i is said to *cover* a vertex $v \in V$ if $v[i] \in \sigma$. A slab $\sigma = [x_1, x_2]$ is called a *singleton* slab if it contains only a single coordinate, i.e.,

$x_1 = x_2$. We call σ *heavy* if it covers more than $|V|/(4d+2)$ vertices. Our algorithm demands an important *heavy-singleton property*:

> If a slab $\sigma = [x_1, x_2]$ of any dimension is heavy, then σ must be a singleton slab.

All the slabs naturally define a d-dimensional *histogram* H. Specifically, H is a d-dimensional grid with at most s^d cells, each of which is a d-dimensional rectangle whose projection on dimension $i \in [1, d]$ is a slab on that dimension. For each cell ϕ of H, the following information should already be available:

- A *vertex count*, equal to the number of vertices $v \in V$ that ϕ contains (i.e, the point v falls in ϕ). Denote by $\phi(V)$ the set of these vertices.
- d *vertex lists*, where the i-th ($1 \leq i \leq d$) one sorts all the vertices of $\phi(V)$ by dimension i. This means that a vertex $v \in \phi(V)$ is duplicated d times. We store with each copy of v all its $O(1)$ adjacent edges.

All the vertex counts are kept in memory. The sorted vertex lists in all the cells, on the other hand, occupy $O(s^d + |V|/B) = O(|V|/B)$ blocks on disk.

Given a slab σ on any dimension, we denote by $\sigma(V)$ the set of vertices covered by σ. The vertex counts in H allow us to obtain $|\sigma(V)|$ precisely, and hence, the average V-occupancy of σ precisely, without any I/Os. Define

$$K = \max_{\text{non-singleton } \sigma} |\sigma(V)| \tag{6}$$

Note that the maximum ranges over all *non-singleton* slabs of *all* dimensions.

As in Section 2.1, our aim is to find a dimension i and a coordinate x such that (i) the V-occupancy of x is at most $(2d+1)^{1/d}|V|^{1-1/d}$, and (ii) at least $|V|/(4d+2)$ vertices are on the left and right of x on dimension i, respectively. Our algorithm will perform $O((M/B)^{1-1/d} + K/B)$ I/Os.

Algorithm. Suppose that the slabs on dimension i are numbered from left to right, i.e., the leftmost one is numbered 1, the next 2, and so on. For dimension $j \in [1, d]$, let y_j be the largest integer y such that at most $|V|/(2d+1)$ points are covered by the slabs on this dimension whose numbers are *less than* y, and similarly, let z_j be the smallest integer z such that at most $|V|/(2d+1)$ points are covered by the slabs on this dimension whose numbers are *greater than* z. It must hold that $y_j \leq z_j$.

Let R be the d-dimensional rectangle whose projection on dimension each $j \in [1, d]$ is the union of the slabs numbered $y_j, y_j + 1, ..., z_j$. As R contains at least $|V|/(2d+1)$ vertices, its projection on at least one dimension covers at least $(|V|/(2d+1))^{1/d}$ coordinates. Fix i to be this dimension. Note that the projection of R on dimension i (i.e., an interval on the dimension) has an average V-occupancy of at most $(2d+1)^{1/d}|V|^{1-1/d}$. Therefore, at least one of the slabs numbered $y_i, y_i + 1, ..., z_i$ on dimension i has an average V-occupancy at most $(2d+1)^{1/d}|V|^{1-1/d}$. Let σ be this slab.

It thus follows that at least one coordinate x within σ has V-occupancy of at most $(2d+1)^{1/d}|V|^{1-1/d}$. If σ is a singleton slab, then x is the (only) coordinate

contained in σ. Otherwise, to find such an x, we scan the vertices of $\sigma(V)$ in ascending order of their coordinates on dimension i. This can be achieved by merging the vertex lists of all the at most s^{d-1} cells in σ — more specifically, the lists sorted by dimension i. The merge takes

$$O(s^{d-1} + |\sigma(V)|/B) = O((M/B)^{1-1/d} + K/B)$$

I/Os, by keeping a memory block as the reading buffer for each cell in σ.

To prove the algorithm's correctness, we first argue that at least $|V|/(4d+2)$ vertices are on the left of x on dimension i. Because of $|V| > M$ and (4), it holds that

$$(2d+1)^{1/d}|V|^{1-1/d} \quad \leq \quad \frac{|V|}{4d+2}.$$

This implies that σ — the slab which x comes from — cannot be heavy. In other words, σ contains no more than $\frac{|V|}{4d+2}$ vertices. Therefore, by definition of y_i, there must be at least

$$\frac{|V|}{2d+1} - \frac{|V|}{4d+2} \quad = \quad \frac{|V|}{4d+2}$$

vertices in the slabs of dimension i whose numbers are less than y_i. All those vertices are on the left of x on dimension i. A symmetric argument shows that at least $|V|/(4d+2)$ vertices are on the right of x on dimension i.

3.2 $2^{\Omega(\log(M/B))}$ Splits

Let $G = (V, E)$ be a d-dimensional grid graph with $|V| > M$ that is stored as follows. First, V is duplicated in d lists, where the i-th one sorts all the vertices $v \in V$ by dimension i. Second, each copy of v stores all the $O(1)$ edges adjacent to v.

In this section, we present an algorithm that achieves the following purpose in $O(|V|/B)$ I/Os: recursively split G using the *one-split algorithm* of Section 3.1 such that, in each resulting subgraph, the number of vertices is at most

$$\max\left\{M, O\left(\frac{|V|}{2^{\Omega(\log(M/B))}}\right)\right\}$$

but at least $M/(4d+2)$.

Our algorithm is inspired by an algorithm of Agarwal et al. [1] for bulkloading the kd-tree I/O-efficiently (but the two algorithms differ considerably in details). Recall that our one-split algorithm has sub-linear cost as long as the histogram is available. The histogram, on the other hand, requires linear cost to prepare, because of which we cannot afford to compute from scratch the histogram for the next split. A crucial observation is that we do not need to do so *from scratch*. This is because a split only affects a small part of the histogram, such that the histograms for the next two splits can be generated from the old histogram incrementally with sub-linear cost.

Constructing the Initial Histogram. Define

$$t = \frac{1}{2} \cdot (M/B)^{1/d}.$$

Partition each dimension $i \in [1, d]$ into disjoint intervals (a.k.a., *slabs*) $\sigma = [x_1, x_2]$ satisfying two conditions:

- σ covers no more than $|V|/t$ vertices, unless σ is singleton (i.e., $x_1 = x_2$).
- The right endpoint x_2 of σ cannot be increased any further without violating the above condition, unless σ is the rightmost slab on this dimension. Equivalently, this means that, if σ is not the rightmost slab, there must be more than $|V|/t$ vertices $v \in V$ satisfying $v[i] \in [x_1, x_2 + 1]$.

These conditions can be understood intuitively as follows. To create a slab of dimension i starting at coordinate x_1, one should set its right endpoint x_2 ($\geq x_1$) as large as possible, provided that the slab still covers at most $|V|/t$ points. But such an x_2 does not exist if x_1 itself already has a V-occupation of more than $|V|/t$; in this case, create a singleton slab containing only x_1. It is easy to obtain these slabs in $O(|V|/B)$ I/Os from the vertex list of V sorted by dimension i.

Proposition 1 *Each dimension has less than* $2t$ *slabs.*

Proof: The union of any two consecutive slabs must cover more than $|V|/t$ vertices. Consider the following pairs of consecutive slabs: (1st, 2nd), (3rd, 4th), ..., leaving out possibly the rightmost slab. A vertex is covered by the union of at most one such pair. Therefore, there can be at most

$$\left\lfloor \frac{|V|}{\lfloor |V|/t \rfloor + 1} \right\rfloor \leq t - 1$$

pairs, making the number of slabs at most $2(t-1) + 1 = 2t - 1$. □

Construct the histogram H on G as defined in Section 3.1. This can be accomplished in $O(|V|/B)$ I/Os. To understand this, observe that, by Proposition 1, the total number of cells in the histogram is at most $(2t)^d \leq M/B$, which allows us to allocate one memory block to each cell. Using these blocks as writing buffers, we can create all the cells' vertex lists on a dimension by scanning V only once.

Recursive One-Splits. We invoke the one-split algorithm on G (notice that all its preprocessing requirements have been fulfilled), which returns a coordinate x and dimension i. The I/O cost is $O((M/B)^{1-1/d} + |V|/(tB))$ I/Os, because the value of K in (6) is at most $|V|/t$ (every non-singleton slab covers at most $|V|/t$ vertices).

The pair x and i defines a separator S, which consists of all the vertices $v \in V$ with $v[i] = x$. Removing S splits G into $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Let σ be the slab on dimension i containing x. Extracting S from σ requires $O(1 + |S|/B + |\sigma(V)|/B) = O(|S|/B + |V|/(tB))$ I/Os.

We will then recursively apply the one-split algorithm on G_1 and G_2, respectively, before that, however, we need to prepare their histograms H_1, H_2. If σ is singleton, H_1 and H_2 can be obtained trivially with no I/Os: H_1 (or H_2, resp.) includes all the cells of H on the left (or right, resp.) of x on dimension i.

If σ is non-singleton, each cell ϕ in σ needs to be split (at x on dimension i) into ϕ_1, ϕ_2, whose information is not readily available yet. We can produce the information of all such ϕ_1, ϕ_2 by inspecting each ϕ as follows:

1. Assign the vertices in ϕ — if not in S — to ϕ_1 or ϕ_2.

2. Prepare the d sorted lists of ϕ_1 and ϕ_2 by splitting the corresponding lists of ϕ.

As there are $O((M/B)^{1-1/d})$ cells in σ, the above steps finish in $O((M/B)^{1-1/d} + |V|/(tB))$ I/Os.

If $|V_1| > M$ (or $|V_2| > M$), we now apply the one-split algorithm on G_1 (or G_2, resp.) — descending one level from G in the split tree — which is recursively processed in the same manner. The recursion ends after we have moved

$$\ell \quad = \quad \left\lfloor (\log_{4d+2} t) - 1 \right\rfloor \tag{7}$$

levels down in the split tree from G. It can be verified that $\ell \geq 1$ (applying (4)) and $2^\ell = O(t)$.

Correctness. Recall that the one-split algorithm requires the heavy-singleton property to hold. We now prove that this property is always satisfied during the recursion. Let $G' = (V', E')$ be a graph processed by the one-split algorithm. Since G' is at most ℓ levels down in the split tree from G, we know (by the fact that each split is balanced) that

$$|V'| \quad \geq \quad \frac{|V|}{(4d+2)^\ell}$$

which together with (7) shows

$$\frac{|V'|}{4d+2} \geq \frac{|V|}{t}.$$

Therefore, a heavy slab σ' of any dimension for G' must contain more than $|V|/t$ vertices. On the other hand, σ' must be within a slab σ defined for G, which thus also needs to cover more than $|V|/t$ vertices. By our construction, σ must be singleton, and therefore, so must σ'.

Finally, it is worth pointing out that each split will generate $O((M/B)^{1-1/d})$ cells, and hence, demands the storage of this many extra vertex counts in memory. This is fine because the total number of vertex counts after $2^\ell = O(t)$ splits is $O((M/B)^{1-1/d} \cdot t) = O(M/B)$.

Bounding the Total Cost. The one-split algorithm is invoked at most 2^ℓ times in total. By the above analysis, the overall I/O cost is

$$\begin{aligned} & O\left(\frac{|S|}{B} + \left(\left(\frac{M}{B}\right)^{1-1/d} + \frac{|V|}{tB}\right) \cdot 2^\ell\right) \\ = \; & O\left(\frac{|S|}{B} + \left(\left(\frac{M}{B}\right)^{1-1/d} + \frac{|V|}{tB}\right) \cdot t\right) \\ = \; & O\left(\frac{|S|}{B} + \frac{M}{B} + \frac{|V|}{B}\right) = O\left(\frac{|V|}{B}\right) \end{aligned}$$

utilizing two facts: (i) every vertex v contributes to the $|S|/B$ term at most once — once included in a separator, v is removed from further consideration in the rest of the recursion, and (ii) a non-singleton slab of any histogram throughout the recursion is within a non-singleton slab of H (the histogram of G), and hence, covers no more than $|V|/t$ vertices.

3.3 The Overall Algorithm

We are ready to describe how to compute an M-separator on a d-dimensional grid graph $G = (V, E)$ in $O(sort(|V|))$ I/Os which, according to the discussion at the beginning of Section 3, will complete the proof of Theorem 2.

First, sort the vertices of V d times, each by a different dimension, thus generating d sorted lists of V. We store with each copy of v all its $O(1)$ edges. The production of these lists takes $O(sort(|V|))$ I/Os.

We now invoke the algorithm of Section 3.2 on G. For each subgraph $G' = (V', E')$ thus obtained, we materialize it into d sorted lists, where the i-th one sorts V' by dimension i, ensuring that each copy of a vertex is stored along with its $O(1)$ edges. This can be done in $O(|V'|/B)$ I/Os as follows. Recall that the algorithm maintains a histogram of at most M/B cells. By allocating a memory block as the writing buffer for each cell, we can generate the sorted list of V' on a dimension by one synchronous scan of the corresponding vertex lists of all cells for the same dimension. The cost is $O(M/B + |V'|/B) = O(|V'|/B)$ because $|V'| \geq M/(4d + 2)$.

Finally, if $|V'| > M$, we recursively apply the algorithm of Section 3.2 on G', noticing that the preprocessing requirements of the algorithm have been fulfilled on G'.

Now we prove that the total cost of the whole algorithm is $O(sort(|V|))$. One application of the algorithm of Section 3.2 on a graph $G' = (V', E')$ costs $O(|V'|/B)$ I/Os, or equivalently, charging $O(1/B)$ I/Os on each vertex of V'. A vertex can be charged $O(\log_{M/B}(|V|/M))$ times, adding up to $O(sort(|V|))$ I/Os overall for all vertices.

4 Density-Based Clustering

In Section 4.1, we will describe an algorithm to solve the DBSCAN problem under the L_∞ norm with the I/O complexities stated in Theorem 4. Our algorithm demonstrates an elegant application of d-dimensional grid graphs. The application requires Corollary 3, which we prove in Section 4.2.

4.1 Proof of Theorem 4

Recall that the input to the DBSCAN problem involves a constant integer $minPts \geq 1$, a real number $\epsilon > 0$, and a set P of n points in $\mathbb{R}^d$. Also recall that a point $p \in P$ is a core point if P has at least $minPts$ points within distance ϵ from p (counting p itself). Our DBSCAN algorithm under the L_∞ norm includes three main steps: (i) core point identification, (ii) core point clustering, and (iii) non-core point assignment. Our discussion will focus on the case where $B > minPts$ (recall that $minPts = O(1)$).

Core Point Identification. We impose an arbitrary grid $\mathbb{G}$ on $\mathbb{R}^d$, where each cell is an axis-parallel d-dimensional square with side length ϵ. Assign each point $p \in P$ to the cell of $\mathbb{G}$ which covers p. If p happens to lie on the boundaries of multiple cells, assign p to an arbitrary one of them. For each cell ϕ of $\mathbb{G}$, denote by $\phi(P)$ the set of points assigned to ϕ. If $\phi(P)$ is not empty, ϕ is a *non-empty* cell. Obviously, there can be at most n non-empty cells; we can find all of them in $O(sort(n))$ I/Os.

We say that a non-empty cell ϕ is *sparse* if $|\phi(P)| \leq B$; otherwise, ϕ is *dense*. Also, another cell ϕ' is a *neighbor* of ϕ if the minimum L_∞ distance between the boundaries of the two cells is at most ϵ. Note that a cell has less than $3^d = O(1)$ neighbors.

The non-empty neighbors of all non-empty cells can be produced in $O(sort(n))$ I/Os as follows. For each non-empty cell ϕ, generate $3^d - 1$ pairs (ϕ, ϕ'), one for each of all its neighbors ϕ', regardless of whether ϕ' is empty. Put all such pairs together, and join them with the list of non-empty cells to eliminate all such pairs (ϕ, ϕ') where ϕ' is empty. The non-empty neighbors of each non-empty cell can then be easily derived from the remaining pairs.

Define the *neighbor point set* of a non-empty cell ϕ — denoted as N_ϕ — to be the set that unions the point sets $\phi'(P)$ of all non-empty neighbors ϕ' of ϕ. Since we already have the non-empty neighbors of all non-empty cells, it is easy to create the N_ϕ of all ϕ in $O(sort(n))$ I/Os. While doing so, we also ensure that the points of N_ϕ are sorted by which $\phi'(P)$ they come from. Note that as each point can belong to $O(1)$ neighbor point sets, all the neighbor point sets can be stored in $O(n/B)$ blocks in total.

Observe that the points in dense cells must be core points. For each sparse cell ϕ, we load $\phi(P)$ in memory and scan through N_ϕ to decide the label (i.e., core or non-core) for each point in ϕ. Clearly, after arranging $\phi(P)$ (resp., N_ϕ) of all the sparse cells ϕ to be stored together in $O(sort(n))$ I/Os, this can be done in $O(n/B)$ I/Os. Therefore, the total I/O cost for core point identification

is bounded by $O(sort(n))$.

Core Point Clustering. Let us first mention a relevant result on the *maxima/minima problem*. Let P be a set of n distinct points in $\mathbb{R}^d$. A point $p_1 \in P$ *dominates* another $p_2 \in P$ if $p_1[i] \geq p_2[i]$ for all dimensions $i \in [1, d]$ — recall that $p[i]$ denotes the coordinate of p on dimension i. The *maxima set* of P is the set of points $p \in P$ such that p is not dominated by any point in P. Conversely, the *minima set* of P is the set of points $p \in P$ such that p does not dominate any point in P. A point in the maxima or minima set is called a *maximal* or *minimal* point of P, respectively. In EM, both the maxima and minima sets of P can be found in $O(sort(n))$ I/Os for $d = 2, 3$, and $O((n/B)\log^{d-2}_{M/B}(n/B))$ I/Os for $d \geq 4$ [27].

Next, we show how to compute the clusters on the core points I/O efficiently. Denote by P_{core} the set of core points of P and by $\phi(P_{core})$ the set of core points assigned to cell ϕ of $\mathbb{G}$. A cell ϕ is called a *core cell* if $\phi(P_{core})$ is non-empty. Let $N_\phi(P_{core})$ be the set of core points in N_ϕ. The $\phi(P_{core})$ and $N_\phi(P_{core})$ of all the core cells ϕ can be extracted from the results of the previous step in $O(sort(n))$ I/Os. Meanwhile, we also ensure that the points of $N_\phi(P_{core})$ are sorted by which cell they come from.

It is also clear that two core points assigned to the same cell ϕ must belong to the same cluster. As a result, it allows us to "sparsify" P_{core} by computing the primitive clusters at the cell level. For this purpose, we define a graph $G = (V, E)$ as follows:

- Each vertex V corresponds to a distinct core cell of $\mathbb{G}$.
- Two different vertices (a.k.a. core cells) $\phi_1, \phi_2 \in V$ are connected by an edge if and only if there exists a point $p_1 \in \phi_1(P_{core})$ and a point $p_2 \in \phi_2(P_{core})$ such that $dist(p_1, p_2) \leq \epsilon$.

We will explain later how to generate G efficiently, but a crucial observation at the moment is that G is a d-dimensional grid graph. To see this, embed the grid $\mathbb{G}$ naturally in a space $\mathbb{N}^d$ with one-one mapping between the cells of $\mathbb{G}$ and the points of $\mathbb{N}^d$. It is easy to verify that there can be an edge between two core cells ϕ_1 and ϕ_2 *only if* their coordinates differ by at most 1 on every dimension.

Thus, we can compute the clusters on core points by computing the CCs (connected components) of G. Corollary 3, which will be proved in Section 4.2, permits us to achieve the purpose in $O(sort(n))$ I/Os. For each CC, collect the union of $\phi(P_{core})$ for each vertex (i.e., core cell) ϕ therein. The union corresponds to a cluster on the core points.

We now discuss the generation of G. Given a core cell ϕ, we elaborate on how to obtain its edges in G. This is easy if ϕ is sparse, in which case we can achieve the purpose by simply loading the entire $\phi(P_{core})$ in memory and scanning through $N_\phi(P_{core})$. The I/O cost of doing so for all the sparse cells is therefore $O(n/B)$.

Consider instead ϕ to be a dense cell. A core cell ϕ' that is a neighbor of ϕ is called a *core neighbor* of ϕ. We examine every core neighbor ϕ' of ϕ, in

ascending order of the appearance of $\phi'(P_{core})$ in $N_\phi(P_{core})$. Let us assume — without loss of generality due to symmetry — that the coordinate of ϕ is at most that of ϕ' on every dimension of $\mathbb{N}^d$. We determine whether there is an edge in G between ϕ and ϕ' by solving three d-dimensional maxima/minima problems, each on no more than $|\phi(P_{core})| + |\phi'(P_{core})|$ points:

1. Find the maxima set Σ_1 of $\phi(P_{core})$, and the minima set Σ_2 of $\phi'(P_{core})$.
2. Construct a set Π of points as follows: (i) add all points of Σ_1 to Π, and (ii) for each point $p \in \Sigma_2$, decrease its coordinate by ϵ on every dimension, and add the resulting point to Π.
3. If Π contains two points with the same coordinates, declare *yes* (i.e., there is an edge between ϕ and ϕ'), and finish. This implies the existence of $p_1 \in \Sigma_1$ and $p_2 \in \Sigma_2$ with $p_1[i] + \epsilon = p_2[i]$ for all $i \in [1, d]$.
4. Find the minima set Σ_3 of Π.
5. If any point of Σ_1 is absent from Σ_3, declare *yes*; otherwise, declare *no*.

To see the correctness, suppose first that there should be an edge. Then, there must be a maximal point p_1 of $\phi(P_{core})$ and a minimal point p_2 of $\phi'(P_{core})$ that have L_∞ distance at most ϵ. Let p_2' be the point shifted from p_2 after decreasing its coordinate by ϵ on all dimensions; p_2' either is dominated by p_1 or coincides with p_1. It follows that p_1 will not appear in Σ_3 if the execution comes to Step 5, prompting the algorithm to output yes. Similarly, one can show that if there should not be an edge, the algorithm definitely reports no.

For $d = 2, 3$, running the above algorithm for all dense cells ϕ and their core neighbors ϕ' entails I/O cost (applying the aforementioned result of [27] on the minima/maxima problem)

$$\sum_{\text{dense } \phi, \text{ core neighbor } \phi'} O(sort(|\phi(P_{core})| + |\phi'(P_{core})|)) = O(sort(n))$$

using the fact that each cell ϕ is a neighbor of less than $3^d = O(1)$ dense cells. In the same fashion using the $d \geq 4$ result of [27], we can bound the I/O cost by $O((n/B)\log_{M/B}^{d-2}(n/B))$.

Non-Core Point Assignment. For each non-core point $p \in P$, we first find all the core points q such that $dist(p, q) \leq \epsilon$; for each q, assign p to the cluster that contains q. If a non-core point is assigned to no clusters, it is classified as noise. The assignment process can be implemented by loading $\phi(P)$ in memory and scanning through N_ϕ for each non-empty sparse cell ϕ. The I/O cost is bounded by $O(sort(n))$.

The total cost of our algorithm is therefore $O(sort(n))$, subject to Corollary 3. The next subsection will prove the correctness of this corollary, which will complete the whole proof of Theorem 4.

4.2 Proof of Corollary 3

Given a d-dimensional grid graph $G = (V, E)$, apply Theorem 2 to compute an M-separator S, as well as its induced subgraphs $G_1 = (V_1, E_1)$, ..., $G_h = (V_h, E_h)$ where $h = O(|V|/M)$. For each $i \in [1, h]$, define G_i^+ as an *extended subgraph* whose

- Vertices include (i) those in V_i and (ii) the separator vertices (i.e., vertices in S) that are adjacent to any boundary vertices of G_i. There are $O(M^{1-1/d})$ such separator vertices, i.e., the same order as the number of boundary vertices.
- Edges include (i) those in E_i, and (ii) the edges between the boundary vertices of G_i and separator vertices. G_i^+ has $O(M)$ edges in total.

All these graphs can be generated in $O(sort(|V|))$ I/Os.

Construct a graph $G' = (V', E')$ with $V' = S$ as follows. First, E' includes all the edges in E among the separator vertices of S. $O(|S|) = O(|V|/M^{1/d})$ edges are added this way. Second, we add to E' additional edges that reflect the connectivity of the separator vertices within each extended subgraph. Specifically, for each $i \in [1, h]$, load into memory G_i^+ and compute its CCs. If a CC contains $x \geq 2$ separator vertices, add to E' $x - 1$ edges that form a tree connecting those vertices. The total number of edges inserted to E' in the second step is $O((|V|/M) \cdot M^{1-1/d}) = O(|V|/M^{1/d})$. Both steps can be done in $O(|V|/B)$ I/Os.

We apply the algorithm[2] of [24] to find the CCs of G' in

$$\begin{aligned} O\left(\frac{|E'|}{B}\log_{M/B}\frac{|E'|}{B}\cdot\log\log B\right) &= O\left(\frac{|V|}{BM^{1/d}}\log_{M/B}\frac{|V|}{B}\cdot\log\log B\right) \\ &= O(sort(|V|)) \end{aligned}$$

I/Os because $M^{1/d} \geq B^{1/d} = \omega(\log\log B)$. Label the vertices of V' (i.e., S) so that vertices in a CC receive the same unique label.

Finally, for $i \in [1, h]$, load G_i^+ into memory again. For each non-separator vertex v_i, give it the same label as any separator vertex that v_i can reach in G_i^+. If no such separator vertex exists, v_i is in a CC that does not involve any separator vertex; all the vertices in the CC are thus given a new label. Doing so for all i entails $O(|V|/B)$ extra I/Os. This establishes Corollary 3.

5 Results on 2D Grid Graphs

This section will concentrate on $d = 2$. Section 5.1 will demonstrate additional applications of Theorem 2 by revisiting the SSSP (single source shortest path) and BFS (breadth first search) problems and proving Corollaries 1 and 2. Section 5.2 will disprove the "sparsity under edge contraction" belief by establishing Theorem 3.

[2] In general, given an undirected graph $G = (V, E)$, the algorithm of [24] finds the CCs in $O(sort(|V|) + sort(|E|)\log\log(|V|B/|E|))$ I/Os.

5.1 SSSP and BFS

Consider a grid graph $G = (V, E)$ where each edge in E is associated with a non-negative weight. Given two vertices v_1, v_2, a *path* from v_1 to v_2 is a sequence of edges in E that allows us to walk from v_1 to v_2 without leaving the graph. The *length* of a path is the sum of the weights of all its edges. The *shortest path* from v_1 to v_2 is a path from v_1 to v_2 with the smallest length; the length of the path is the *shortest distance* from v_1 to v_2.

In the SSSP problem, besides G, we are also given a source vertex v_{src}, and need to output the shortest paths and distances from v_{src} to all the other vertices in V. In particular, all the shortest paths must be reported space-economically in a *shortest path tree* where (i) each node corresponds to a distinct vertex in V, (ii) v_{src} is the root, and (iii) the shortest path from v_{src} to any other vertex v in G goes through the same sequence of vertices as in the path from v_{src} to v in the tree[3]. The tree should be stored in the disk using the *child adjacency format* where each node is associated with a list of its children.

Consider an M-separator S of G and its $h = O(|V|/M)$ subgraphs $G_1, ..., G_h$. Given a separator vertex $v \in S$, its *adjacent set* is the set of all G_i ($i \in [1, h]$) such that E has an edge between v and at least one vertex in G_i. Arge et al. [4] proved that the SSSP problem can be solved in $O(|V|/\sqrt{M} + sort(|V|))$ I/Os, as long as S fulfills the following *separator-decomposition* requirement:

> S has been divided into $g = O(|V|/M)$ disjoint subsets $S_1, ..., S_g$ such that the vertices in each S_i ($1 \leq i \leq g$) have the same adjacent set.

Our objective is to strengthen the M-separator S in Theorem 2 to satisfy the above requirement in $O(sort(|V|))$ I/Os.

Let S and $G_1, ..., G_h$ be the separator and subgraphs that Theorem 2 returns for G. Recall that our algorithm of Theorem 2 recursively performs binary splits using vertical/horizontal line segments in $\mathbb{N}^2$. If we remove these segments, the remaining portion of the data space consists of disjoint axis-parallel rectangles, which we call *residue rectangles*. It must hold that (i) separator vertices can lie only on these line segments, and (ii) each G_i ($i \leq [1, h]$) is induced by the vertices in a distinct residue rectangle. This property motivates a simple algorithm for dividing S to satisfy the separator-decomposition requirement. First, label the subgraphs arbitrarily from 1 to h. For each vertex $v \in S$, generate a *label list* that sorts in ascending order the labels of the subgraphs in the adjacent set of v. The list has length $O(1)$. We now partition S into disjoint subsets, where the vertices in each subset have the same label list. The aforementioned property implies that there are only $O(|V|/M)$ subsets. The partitioning can be easily done by sorting in $O(sort(|V|))$ I/Os, thus establishing Corollary 1.

The BFS problem is, essentially, an instance of SSSP on a grid graph where all edges have the same weight. In particular, the shortest path tree corresponds to the *BFS tree*. Corollary 1 immediately implies Corollary 2. It is worth mentioning that, in $O(sort(|V|))$ I/Os, one can compute from the BFS tree an

[3]Equivalently, the parent of each node v is the predecessor of v on the shortest path from v_{src} to v.

alternative encoding where every node in the tree keeps a pointer referencing its parent in the tree. In fact, within the same I/O cost, one can even compute a "blocked version" of this encoding such that, for any node v, the path from v to the root v_{src} of the BFS tree (i.e., the reverse of the shortest path from v_{src} to v) is stored in $O(1 + \ell/B)$ blocks, where ℓ is the number of edges of the path (see Theorem 1 of [17]).

5.2 Disproving Edge-Contraction Sparsity

This subsection serves as a proof of Theorem 3. Recall that a graph $G = (V, E)$ is sparse if $|E| \leq c|V|$, for some constant $c > 0$. Given any integer $m \geq 2$, we will design a grid graph that can be edge-contracted into a clique of m vertices. The clique is not sparse when $m > 2c + 1$. Thus, regardless of the choice of c, there is always a grid graph that is not sparse under edge contraction.

Before proceeding, let us point out a basic geometric fact that will be useful in our design. Let $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ be two distinct points in $\mathbb{R}^2$ such that x_1, y_1, x_2, y_2 are all even integers. Let ℓ_1 be the line with slope 1 passing p_1, and ℓ_2 be the line with slope -1 passing p_2. Then, the intersection of ℓ_1 and ℓ_2 must be a point whose coordinates on both dimensions are integers.

Given integers i, j satisfying $i \in [0, m-1]$ and $j \in [0, m-2]$, let $F(i, j)$ be the point $(1000m \cdot i, 100j)$ in $\mathbb{R}^2$. Call these $m(m-1)$ points *cornerstones*.

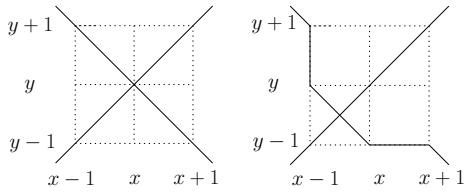
For each pair (i, j) with $i \in [0, m-1]$, $j \in [0, m-2]$ and $i \leq j$ — there are $m(m-1)/2$ such pairs — define a *wedge path* between cornerstones $F(i, j)$ and $F(j+1, i)$ as follows. Shoot a ray with slope 1 emanating upward from $F(i, j)$, and a ray with slope -1 emanating upward from $F(j+1, i)$. Let p be the intersection of the two rays; p must have integer coordinates. The wedge path consists of two segments: the first one connects $F(i, j)$ and p, while the second connects p and $F(j+1, i)$.

The above definition yields $m(m-1)/2$ wedge paths. Two such paths may intersect each other; and the intersection point has integer coordinates — a property that is *not* desired. Next, we will contort some paths a little to ensure the following property: any two resulting paths are either disjoint or intersect only at a point with *fractional* coordinates on both dimensions.

Let P_{intr} be the set of intersection points among the wedge paths. For each point $p = (x, y)$ in P_{intr}, place a square $[x-1, x+1] \times [y-1, y+1]$ centered at p. The constants 1000 and 100 in the definition of $F(i, j)$ ensure that: (i) the squares are disjoint from each other, and (ii) all of them are above the line $y = 100(m-2)$, i.e., higher than all cornerstones.

Focus now on one such square, as shown in Figure 5a, where the two lines illustrate the intersecting wedge paths. We contort one of the two paths as shown in Figure 5b, so that the two paths now intersect at the point $(x - 1/2, y - 1/2)$. Apply the same contortion in all squares.

For each $i \in [0, m-1]$, we add a *vertical path* from cornerstone $F(i, 0)$ through $F(i, m-2)$. These m paths and the $m(m-1)/2$ wedge paths (possibly contorted) give rise to the edges in our grid graph G — notice that every path uses only segments each connecting two points whose coordinates are integers differing by



(a) Before (b) After

Figure 5: Contortion within a square

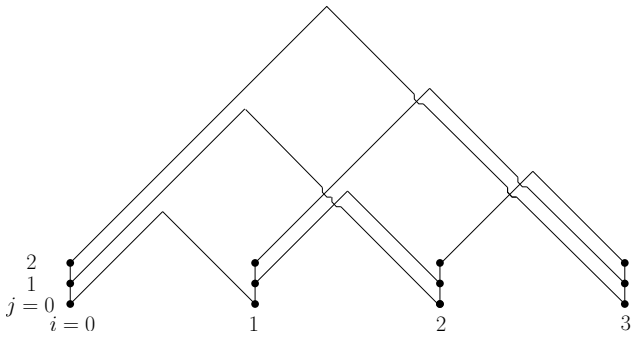


Figure 6: The designed grid graph for $m = 4$ (the black points are the cornerstones; the other vertices are dotted along the curves, but are omitted for clarity)

at most 1 on each dimension. To complete the graph with vertices, we simply place a vertex at every point p of $\mathbb{R}^2$ such that (i) p has integer coordinates on both dimensions, and (ii) p is on one of those $m + m(m-1)/2$ paths. See Figure 6 for such the final G with $m = 4$.

It remains to explain how to perform edge contractions to convert G into a clique of m vertices. First, contract every vertical path into a "super vertex". Between each pair of super vertices, there remains a sequence of edges corresponding to one unique wedge path. The $m(m-1)/2$ edge sequences do not share any vertices except, of course, the super vertices. Contracting each wedge path down to the last edge gives the promised clique.

6 Conclusions

This paper has proved that any d-dimensional grid graph $G = (V, E)$ admits a vertex separator that (i) resembles the well-known multi-way vertex separator of a planar graph, and (ii) can be obtained solely by dividing the space recursively with perpendicular planes, and collecting the vertices on those planes. Furthermore, we have shown that such separators can be computed in $O(sort(|V|))$ I/Os, even if the memory can accommodate only two blocks.

A major application of the above findings is that they lead to an algorithm that performs DBSCAN clustering in d-dimensional space with near-linear I/Os, when the distance metric is the L_∞ norm. Our techniques also lead to improved results on three fundamental problems: CC, SSSP, and BFS. Specifically, the CC problem has been settled in $O(sort(|V|))$ I/Os for any d-dimensional grid graph $G = (V, E)$. Our improvement on SSSP and BFS, however, is less significant, and concerns only small values of M.

We close the paper with some open questions. First, is it possible utilize our separator-computation algorithm to improve the I/O complexity of the DFS algorithm in [16]? Second, does BFS on a 2D grid graph require $\Omega(|V|/\sqrt{M})$ I/Os in the worst case, thus making the result of Corollary 2 optimal? Finally, the practicality of our algorithms also deserves further investigation. We expect that these algorithms, as presented, are useful only when the dimensionality is a small constant. Since in this paper we focused on proving theoretical bounds, we have not discussed any heuristics that may reduce the running cost on "real-world data". The development of such heuristics is a meaningful topic for follow-up engineering research.

Acknowledgements

This work was supported by a direct grant (Project Number: 4055079) from the Chinese University of Hong Kong, and by a Faculty Research Award from Google.

References

[1] P. K. Agarwal, L. Arge, O. Procopiuc, and J. S. Vitter. A framework for index bulk loading and dynamization. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 115–127, 2001. doi:10.1007/3-540-48224-5_10.

[2] P. K. Agarwal, L. Arge, and K. Yi. I/O-efficient batched union-find and its applications to terrain analysis. 7(1):11, 2010. doi:10.1145/1868237.1868249.

[3] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.

[4] L. Arge, G. S. Brodal, and L. Toma. On external-memory mst, SSSP and multi-way planar graph separation. *J. Algorithms*, 53(2):186–206, 2004. doi:10.1016/j.jalgor.2004.04.001.

[5] L. Arge, J. S. Chase, P. N. Halpin, L. Toma, J. S. Vitter, D. Urban, and R. Wickremesinghe. Efficient flow computation on massive grid terrain datasets. *GeoInformatica*, 7(4):283–313, 2003. doi:10.1023/A:1025526421410.

[6] L. Arge, L. Toma, and J. S. Vitter. I/O-efficient algorithms for problems on grid-based terrains. *ACM Journal of Experimental Algorithmics*, 6:1, 2001. doi:10.1145/945394.945395.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[8] K. Deng, X. Zhou, H. T. Shen, Q. Liu, K. Xu, and X. Lin. A multi-resolution surface distance model for k-nn query processing. *The VLDB Journal*, 17(5):1101–1119, 2008. doi:10.1007/s00778-007-0053-2.

[9] J. Erickson. On the relative complexities of some geometric problems. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, pages 85–90, 1995.

[10] J. Erickson. New lower bounds for Hopcroft's problem. *Discrete & Computational Geometry*, 16(4):389–418, 1996. doi:10.1007/BF02712875.

[11] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of ACM Knowledge Discovery and Data Mining (SIGKDD)*, pages 226–231, 1996.

[12] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal of Computing*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

[13] J. Gan and Y. Tao. On the hardness and approximation of euclidean DBSCAN. *ACM Transactions on Database Systems (TODS)*, 42(3):14:1–14:45, 2017. doi:10.1145/3083897.

[14] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2012.

[15] H. J. Haverkort and L. Toma. I/O-efficient algorithms on near-planar graphs. *J. Graph Algorithms Appl.*, 15(4):503–532, 2011.

[16] J. Her and R. S. Ramakrishna. An external-memory depth-first search algorithm for general grid graphs. *Theoretical Computer Science*, 374(1-3):170–180, 2007. doi:10.1016/j.tcs.2006.12.022.

[17] D. A. Hutchinson, A. Maheshwari, and N. Zeh. An external memory data structure for shortest path queries. *Discrete Applied Mathematics*, 126(1):55–82, 2003. doi:10.1016/S0166-218X(02)00217-2.

[18] V. Kumar and E. J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Symposium on Parallel and Distributed Processing*, pages 196–176, 1996. doi:10.1109/SPDP.1996.570330.

[19] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. on Applied Math.*, 36:177–189, 1979. doi:10.1137/0136016.

[20] L. Liu and R. C. Wong. Finding shortest path on land surface. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 433–444, 2011. doi:10.1145/1989323.1989369.

[21] A. Maheshwari and N. Zeh. I/O-efficient planar separators. *SIAM Journal of Computing*, 38(3):767–801, 2008. doi:10.1137/S0097539705446925.

[22] K. Mehlhorn and U. Meyer. External-memory breadth-first search with sublinear I/O. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 723–735, 2002. doi:10.1007/3-540-45749-6_63.

[23] G. L. Miller, S. Teng, and S. A. Vavasis. A unified geometric approach to graph separators. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 538–547, 1991. doi:10.1109/SFCS.1991.185417.

[24] K. Munagala and A. G. Ranade. I/O-complexity of graph algorithms. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 687–694, 1999.

[25] M. H. Nodine, M. T. Goodrich, and J. S. Vitter. Blocking for external graph searching. *Algorithmica*, 16(2):181–214, 1996. doi:10.1007/BF01940646.

[26] C. Shahabi, L. A. Tang, and S. Xing. Indexing land surface for efficient knn query. *PVLDB*, 1(1):1020–1031, 2008.

[27] C. Sheng and Y. Tao. Finding skylines in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 107–116, 2011. doi:10.1145/1989284.1989298.

[28] W. D. Smith and N. C. Wormald. Geometric separator theorems & applications. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 232–243, 1998. doi:10.1109/SFCS.1998.743449.

[29] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson, 2006.

[30] L. Toma and N. Zeh. I/O-efficient algorithms for sparse graphs. In *Algorithms for Memory Hierarchies, Advanced Lectures*, pages 85–109, 2002.

[31] S. Xing, C. Shahabi, and B. Pan. Continuous monitoring of nearest neighbors on land surface. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):1114–1125, 2009.

[32] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of ACM Knowledge Discovery and Data Mining (SIGKDD)*, pages 71–80, 2002. doi:10.1145/775047.775058.

[33] N. Zeh. I/O-efficient graph algorithms. Technical report, Dalhousie University, 2002.