THE MINIMUM EDGE PRODUCT LINEAR ORDERING PROBLEM

PETER LINDSTROM*

Abstract. The minimum p-sum linear ordering problem has been well studied in the graph and sparse matrix ordering literature, e.g. for $p = 1$ (linear arrangement, also known as edge sum) and for $p = \infty$ (bandwidth). In this paper we study a new layout measure that we call *edge product*, which can be thought of as the instance $p = 0$ of the family of p-sum problems. We show that for regular grids, space-filling layouts such as the Hilbert curve are near optimal under this measure. Indeed, like space-filling curves, the minimum-product layout has a "fractal" nature, and can be thought of as an algebraic generalization of space-filling curves to unstructured graphs that does not require a geometric embedding. Whereas space-filling and "graph-filling" layouts have traditionally been defined explicitly in terms of a constructive procedure, the minimum-product measure *implicitly* defines similar layouts, thereby allowing for the possibility of different layout optimization strategies and the ability to quantitatively measure the quality of a layout. We describe an efficient multilevel optimization procedure for producing minimum-product layouts and show that these layouts have excellent locality properties, with applications for example in matrix ordering, data organization, dimensionality reduction, and graph partitioning.

Key words. p-sum problem, matrix ordering, graph layout, linear arrangement, data locality, graph partitioning

AMS subject classifications. 05C78, 05C35, 05C85, 65F10, 65K10

1. Introduction.

Let $\varphi : V \to \{1, 2, \ldots, |V|\}$ be a linear layout, or ordering, of an undirected graph $G(V, E)$ with vertices V and edges E, i.e., φ describes a particular permutation of the vertices. The p-sum of a graph layout φ for $0 < p < \infty$ is defined as:

$$\sigma_p^p = \sum_{ij \in E} |\varphi(i) - \varphi(j)|^p \tag{1.1}$$

The p-sum problem is to find a layout φ for which σ_p^p is minimal. One classic instance of this family of problems is $p = 1$: the *minimum edge sum problem*, named so since it involves finding the minimum sum of edge "lengths" $\sum |\varphi(i) - \varphi(j)|$. To allow the case $p = \infty$, the equivalent p-discrepancy [26] may be used:

$$\sigma_p = \lim_{q \to p} \left(\sum_{ij \in E} |\varphi(i) - \varphi(j)|^q \right)^{1/q} \tag{1.2}$$

Many authors do not distinguish between these two measures, and both are commonly referred to as p-sum [13, 33, 35, 44]. We propose another slight modification that results in yet another equivalent measure on $0 < p \leq \infty$, and which is also well-defined when $p = 0$:

$$\mu_p = \lim_{q \to p} \left(\frac{1}{|E|} \sum_{ij \in E} |\varphi(i) - \varphi(j)|^q \right)^{1/q} \tag{1.3}$$

μ_p is a p-mean, also known as power mean or generalized mean. Using limits the p-mean reduces to the well known geometric mean when $p = 0$:

$$\mu_0 = \exp\left(\frac{1}{|E|} \sum_{ij \in E} \log |\varphi(i) - \varphi(j)| \right) = \left(\prod_{ij \in E} |\varphi(i) - \varphi(j)| \right)^{1/|E|} \tag{1.4}$$

Clearly μ_0 is minimal when $\prod |\varphi(i) - \varphi(j)|$ is, and we refer to finding the layout φ that minimizes μ_0 as the *minimum edge product problem*.

*Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, California, 94550 (pl@llnl.gov).

Minimum p-sum layouts have found use in many applications, including VLSI circuit design [1], sparse matrix ordering [14], graph drawing [30], and others [10]. Minimum 1-sum, also known as minimum linear arrangement (MINLA) and minimum edge sum, has been proposed for cache-friendly layout of meshes to improve data locality; see e.g. [3]. Spectral sequencing, the continuous analogue of the discrete 2-sum problem [44], as well as ∞-sum, also known as bandwidth (MINBW), have been proposed for organizing unstructured meshes to support streaming access with small memory footprints [24]. Spectral graph ordering has also been used successfully for graph partitioning [20, 40] and for clustering [37]. Sparse matrix representations often benefit from orderings with low bandwidth, i.e., that minimize ∞-sum [14].

Since all of these p-sum measures favor orderings with "short" edges that connect related data elements, they have seen widespread use for data organization in applications that require a high degree of locality of reference [1, 14, 16, 24, 26, 29, 34, 42, 57, 58]. Such applications model anticipated access patterns as graphs, where the vertices represent atomic data elements and the edges joining two vertices represent an anticipated likelihood of accessing corresponding data elements in (near) succession [52, 57]. The current computer hardware trend of an increasing number of cores per processor and decreasing memory bandwidth per core puts a premium on data locality to reduce data movement, and is a key motivator for investigating p-sum and related locality measures. In this paper we primarily consider *spatial locality* (nearby elements are accessed together), though *temporal locality* (repeated access to an element or cache line) may also be considered [21].

While p-sum is in general known to be NP-hard [10], optimal layouts of 2D regular grids are known for $p \in \{1, \infty\}$. Such layouts order vertices row by row for $p = \infty$, or partially row by row for $p = 1$ [10] (see Figure 5.4). Hence they result in good cache reuse when accessing the data sequentially in a row by row fashion, as exploited in [24]. However, contrary to [3], such layouts can be rather poor for more general access patterns, such as an orthogonal column by column sweep, or when the cache is too small to hold even a single row. As a result, space-filling layouts [46], which favor no particular direction of ordering, are widely used and are believed to be near optimal for ordering regular grids [15, 36]. Space-filling curves provide a linearization of Euclidean space using a set of deterministic rules, and therefore map well to Cartesian grid graph structures. To use space-filling curves to order non-geometric or unstructured data, a geometric embedding is needed, which usually does not account for the combinatorial structure of the graph or mesh. Nevertheless, space-filling layouts have found some success in applications for general graphs, e.g., in mesh partitioning [9, 23, 48, 60].

In this paper we propose using μ_0 as a measure of locality for evaluating and optimizing layouts of both structured and unstructured graphs and meshes. Whereas p-sum for $1 \leq p \leq \infty$ is a well studied problem [10], we are aware of only one investigation of the case $p < 1$. Mitchison and Durbin [34] conjectured that optimal layouts of grids for $0 < p < 1$ must have a "fractal character" with "fine-scale detail," as is true of our minimum edge product (MINEP) layouts. Somewhat surprisingly, the special case $p = 0$ appears not to have been considered in the literature, perhaps because p-sum as originally defined does not result in a meaningful measure when $p = 0$. The *0-mean* formulation, however, does lead to a well-defined and simple logarithmic measure. We note that a similar yet subtly different measure, $\sum \log(|\varphi(i) - \varphi(j)| + 1)$, was proposed by Wierum [55]. This functional, which is not a p-mean, was motivated by Wierum as a suitable measure of locality in divide-and-conquer operations like searching and sorting. Wierum used his measure only to evaluate the locality properties of space-filling curves on a grid, and no effort was made to optimize layouts for this measure nor to apply it to graphs other than regular grids.

In [57], we derived μ_0 as a cache-oblivious locality measure in an effort to capture and minimize the *edge cut* [27] across blocks of consecutive vertices at multiple scales. Like space-filling curves, we showed that mesh layouts optimized for μ_0 result in few cache misses regardless of cache size. We proposed a rudimentary layout algorithm for minimizing μ_0 based on recursive graph partitioning, and conjectured that lower bounds on μ_0 may exist for judging the absolute quality of a layout. Here we provide both lower and upper bounds for hierarchical layouts of regular grids, describe a more effective layout algorithm, present optimal grid layouts, propose a new space-filling curve with excellent locality properties, and show that our layouts can be used effectively for data organization and for graph and mesh partitioning.

2. Edge product locality measure. Our original motivation for the edge product functional μ_0 was to derive a simple measure of locality for graph orderings in a cache-oblivious sense, i.e. with no knowledge of cache and line size, associativity, replacement policy, etc. Using a simple and generic cache model, we here give a synopsis and somewhat simplified derivation of our cache-oblivious measure presented in [57]. As in [52, 57, 58], we model data elements as vertices in a graph, with an undirected edge indicating a nonzero likelihood that its two vertices be accessed in succession, or more generally a desire to store those two vertices close together in memory. We call such a graph an *affinity graph*.

Consider a cache consisting of a single block that can hold N data elements and is backed by a larger memory consisting of several such blocks. (In practice caches tend to hold more than one block, but that would unnecessarily complicate our derivation.) Let i and j be data elements stored at $\varphi(i)$ and $\varphi(j)$ in blocks $B(i)$ and $B(j)$, respectively, and separated by $\ell_{ij} = |\varphi(i) - \varphi(j)|$ on linear storage. Without loss of generality, assume $\varphi(i) < \varphi(j)$. Suppose i is accessed first, bringing $B(i)$ into the cache. We wish to estimate the probability of a cache miss when j is accessed. Clearly, if $\ell \geq N$, then a cache miss is inevitable, since then $B(i) \neq B(j)$. When $\ell < N$, the likelihood of a cache miss depends on the position of i within $B(i)$, i.e. where the upper boundary of $B(i)$ lies with respect to $\{\varphi(i), \ldots, \varphi(j)\}$. In absence of further information, we will assume that the position of i within $B(i)$ is distributed uniformly.[1] Hence, there are ℓ out of N possible alignments that partition i and j into separate blocks, and the probability of a cache miss occurring when j is accessed is

$$M_N(\ell) = \begin{cases} \frac{\ell}{N} & \text{if } \ell < N \\ 1 & \text{otherwise} \end{cases} \tag{2.1}$$

The observation underlying the cache-oblivious measure μ_0 that we derived in [57] is that most block-based caches employed in current computer architectures are hierarchical and nested, with a roughly geometric progression in size.[2] That is, we may write $N = b^k$ for some base b (usually $b = 2$) and positive integer k. Thus, we wish to estimate the total number of cache misses for all k. We have

$$M(\ell) = \sum_{k=1}^{\infty} M_{b^k}(\ell) = \sum_{k=1}^{\lfloor \log_b \ell \rfloor} 1 + \sum_{k=\lfloor \log_b \ell \rfloor + 1}^{\infty} \frac{\ell}{b^k} = \lfloor \log_b \ell \rfloor + \ell \frac{b^{-\lfloor \log_b \ell \rfloor}}{b-1} \tag{2.2}$$

We note that when ℓ is an exact power of b, $M(\ell)$ simplifies to $\log_b \ell + \frac{1}{b-1}$; otherwise $M(\ell)$ increases monotonically with ℓ. Our primary goal is not to estimate the exact number of

[1] Even in practice, modern operating systems allocate memory blocks with nearly arbitrary alignment.

[2] For instance, the memory hierarchy for the computer used in this paper consists of $2^{16} = 64$ KB L1 cache, $2^{18} = 256$ KB L2 cache, $2^{23} = 8$ MB L3 cache, $2^{34} = 16$ GB of RAM, and $2^{40} = 1$ TB of disk.

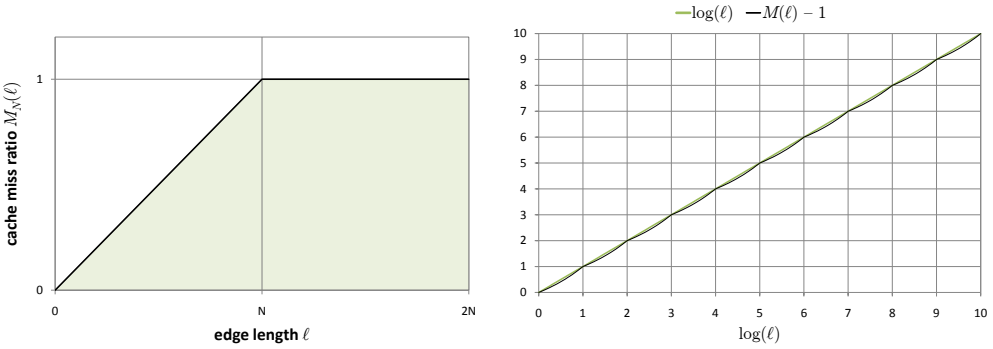


FIG. 2.1. *(a) The cache miss ratio M_N as a function of edge length is linear up to the cache block size N. (b) The* average cache miss ratio *(M) is a measure of the expected number of cache misses across a memory hierarchy. This plot shows that the M associated with accessing two data elements ij is strongly correlated with the logarithm of the distance $x = \ell_{ij}$ between the elements.*

cache misses incurred, but rather to assign a relative "cost" as a function of edge length ℓ. We may thus ignore the value of b (since it affects only the slope of M) and the constant term independent of ℓ, and arrive at the approximation

$$M(\ell) \simeq \log \ell. \tag{2.3}$$

The quality of this approximation is quite good, as evidenced by Figure 2.1(b).

Finally, if we consider all edges E of an affinity graph, the *average cache miss ratio* is

$$M = \frac{1}{|E|} \sum_{ij \in E} M(\ell_{ij}) = \frac{1}{|E|} \sum_{ij \in E} \log |\varphi(i) - \varphi(j)| = \log \mu_0 \tag{2.4}$$

In other words, $\mu_0 = \exp(M)$. As a result, low values of μ_0 imply good cache utilization, not only for a particular size cache but across the whole memory hierarchy. As we shall see, this expected behavior is observed also in practice, with layouts optimized for μ_0 having excellent locality properties.

3. Layouts of regular grids. It is well-known from image processing [51], databases [4], visualization [32], and matrix computations [6], among other fields, that discrete space-filling curve orderings of regular grids have excellent locality properties (see also [15, 18, 25, 36, 38, 46, 54]). It is therefore natural to ask whether space-filling curves result in low values of μ_0. Conversely, do optimal layouts of regular grids with respect to μ_0 exhibit space-filling qualities? In this section we answer both questions in the affirmative, and further show that *any* dyadic grid layout constructed by binary recursive partitioning results in a μ_0 that is bounded both from above and below.

3.1. Bounds for hierarchical grid layouts. We here derive lower and upper bounds on μ_0 for *hierarchical* layouts of 2D grids. In a hierarchical layout, the grid is recursively partitioned into smaller rectilinear subgrids, or *tiles*, such that the vertices V in a subgrid occupy a contiguous range $\{\varphi(i) : i \in V\} = \{p, p+1, \ldots, p+|V|-1\}$ in the layout. We note that Hasan et al. [17] have recently derived upper bounds on μ_0 for the optimal and worst-case (nonhierarchical) layouts of bounded-degree planar graphs, the latter being a trivial $O(|V|)$ bound. Our contribution is a *lower* bound on the best hierarchical layout of a

dyadic grid, as well as a tighter upper bound on the worst such layout. We show that μ_0 for any hierarchical layout of a grid of arbitrary size is $O(1)$, which suggests that hierarchical layouts, however constructed, have good locality properties. Indeed, many space-filling curves, such as the Hilbert curve and Morton curve, are hierarchical layouts. We here limit the discussion to grids of size $2^k \times 2^k$, which we further partition into four subgrids of size $2^{k-1} \times 2^{k-1}$, and so on.

Our basic approach is to derive a recurrence for μ_0 by considering best- and worst-case orderings of the four subgrids in the hierarchy, and by computing lower respectively upper bounds on the lengths of edges that connect the subgrids. Though our bounds are not tight, they are finite in the limit as $k \to \infty$, and their values may be useful in estimating the quality of a given hierarchical layout. We note that μ_0 for nonhierarchical layouts need not converge. For instance, a lexicographic (row-by-row) ordering of an $n \times n$ grid yields $\mu_0 = \sqrt{n}$.

3.1.1. Lower bound. A lower bound can be determined by considering the distances between tiles in the 1D layout, with each tile being a dyadic grid. In particular, we will focus on finding lower bounds on the lengths of edges that connect tiles in a 4×4 grid of tiles of size $2^k \times 2^k$. We refer to these edges as *cut edges* and the vertices they connect as *cut vertices*.

Let $V_k = 4^k$ and $E_k = 2^{k+1}(2^k - 1)$ denote the number of vertices and edges in a $2^k \times 2^k$ tile, and let $\varphi(I)$ denote the linear position of tile I among all tiles on the same level in the hierarchy. Suppose two adjacent tiles are not consecutive, i.e. $\ell_{IJ} = |\varphi(I) - \varphi(J)| > 1$. Then each of the 2^k cut edges that join I and J must have length at least $(\ell_{IJ} - 1)V_k$. Consequently, these edges contribute at least

$$2^k \log_2\big((\ell_{IJ} - 1)V_k\big) = 2^k\big(2k + \log_2(\ell_{IJ} - 1)\big) \tag{3.1}$$

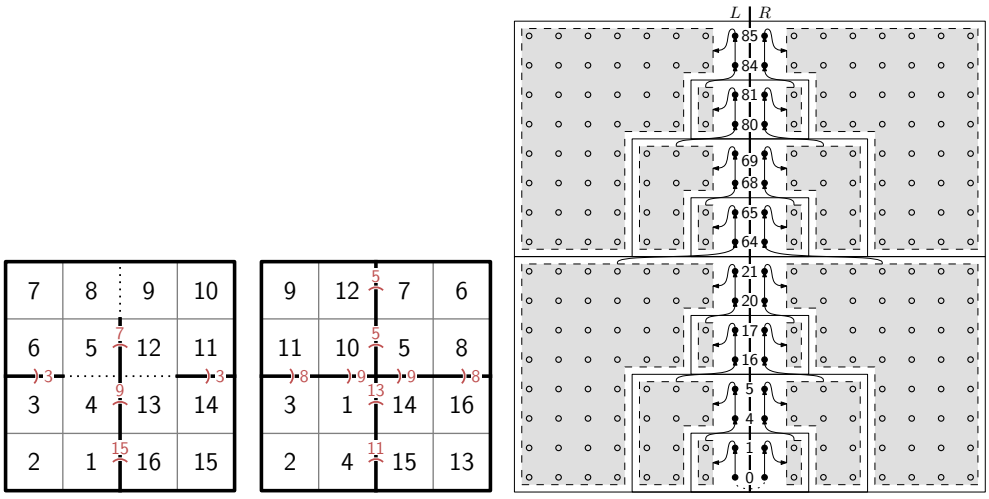
to μ_0, where, for reasons that will become clear, we have used the base-2 logarithm in the μ_0 computation. Note in particular that this contribution to the lower bound is independent of the ordering of vertices within each tile. Moreover, the contribution can be factored into a tile size component $2k$ and a size-independent component that depends only on the relative ordering of tiles, given by ℓ_{IJ}. For the sake of computing a lower bound, this single optimal ordering of tiles can then be used on all levels of the hierarchical layout.

To find the optimal set of lengths ℓ_{IJ} in a 4×4 grid, we simply resort to exhaustive enumeration of all $4!^5 \simeq 8$ million possible hierarchical layouts. The best such layout is shown in Figure 3.1(a), resulting in distances $\{3, 3, 7, 9, 15\}$ between the nonconsecutive tiles (red edges). In other words, the contribution of nonconsecutive tiles is

$$a(k) = 2^k(10k + 2\log_2 2 + \log_2 6 + \log_2 8 + \log_2 14) = 2^k(10k + \log_2 21 + 7) \tag{3.2}$$

The optimal layout shows that there are three pairs of contiguous tiles—the minimum number possible—that reside in different quadrants, and whose adjoining edges complete the set of cut edges on this level in the hierarchy. Applying the bound in Equation 3.1 to these tiles is not useful, as it gives a trivial lower bound of zero for the length of their cut edges. Instead, we must consider the ordering of vertices within such tiles, and examine how close the vertices could possibly be in the linear layout.

Consider two adjacent consecutive tiles: one on the left and one on the right. Assuming the left tile is ordered before the right tile, to minimize μ_0 we clearly must place the cut vertices L_k in the left tile as late as possible, and then place the corresponding cut vertices R_k in the right tile as early as possible. Note that due to the hierarchical partitioning of vertices into subtiles, the cut vertices in L_k cannot in general be placed consecutively (and similarly for R_k), since the vertices in a subtile occupy a contiguous range, but not all vertices in a subtile are (simultaneously) cut vertices. Instead, we are left with 2^k noncontiguous "slots"



(a) μ_0-minimal layout (b) μ_0-maximal layout (c) slots for cut vertices

FIG. 3.1. *(a, b) Hierarchical orderings of a 4×4 grid that minimize respectively maximize μ_0. The red cut edges join nonconsecutive tiles. Dashed lines indicate boundaries between consecutive tiles in different quadrants. (c) The hierarchical ordering constrains the positions of cut vertices (solid) on a partition boundary. The numbering corresponds to the relative vertex positions p_i used in the derivation of the lower bound on μ_0.*

$P = \{p_i\}_{i=1}^{2^k}$ for the cut vertices, where p_i measures the linear distance from the last vertex in L_k and, symmetrically, from the first vertex in R_k.

First, it is easy to see that the slot positions are given by

$$p_i = \begin{cases} 0 & \text{if } i = 1 \\ w_i^2 + p_{i-w_i} & \text{otherwise} \end{cases} \qquad w_i = \begin{cases} 0 & \text{if } i = 1 \\ 2^{\lfloor \log_2(i-1) \rfloor} & \text{otherwise} \end{cases} \tag{3.3}$$

where w_i is the largest power of two less than or equal to $i - 1$. In other words, the i^{th} cut vertex must be preceded by all w_i^2 vertices in the square tile that contains the first w_i cut vertices in L_k (see Figure 3.1(c)). Note that the hierarchical ordering fully constrains the slot numbers p_i, under the assumption that any remaining degrees of freedom for ordering vertices within a tile are chosen so as to minimize the slot numbers. The assignment of slot numbers to rows is also partially constrained by the hierarchical layout. Without loss of generality we may order the cut vertices in L_k row by row, so that the cut vertex on row i is assigned slot p_i. What remains is the assignment of vertices in R_k to the available slots P so as to minimize μ_0 over the set of cut edges.

Let q_i denote the slot assignment for the cut vertex on row i in R_k. It is easy to show that the assignment that minimizes μ_0 is $p_i = q_i$, resulting in cut edges of length $\ell_i = p_i + q_i + 1 = 2p_i + 1$. We proceed using proof by contradiction. Consider two cut edges on rows i and j, and assume $p_i < p_j$ and $q_i > q_j$. Then

$$(p_i + q_i + 1)(p_j + q_j + 1) - (p_i + q_j + 1)(p_j + q_i + 1) = (p_j - p_i)(q_i - q_j) > 0 \tag{3.4}$$

Thus, the contribution to the edge product is reduced by swapping the slots q_i and q_j such that $q_i < q_j$. As a result, $p_i < p_j$ implies $q_i < q_j$, and since p_i and q_i are drawn from the same set, we must have $p_i = q_i$. As stated above, the cut edges are thus of length $\ell_i = 2p_i + 1$.

Although $\sum_i \log_2 \ell_i$ is a tight bound, we would prefer a simpler, closed-form expression that does not involve summations and the somewhat nontrivial computation associated with

ℓ_i. We proceed by making use of the fact that $\ell_i > 2w_i^2$:

$$\begin{aligned}\sum_{i=1}^{2^k} \log_2 \ell_i &= \sum_{i=2}^{2^k} \log_2 \ell_i = \sum_{i=2}^{2^k} \log_2(2p_i + 1) > \sum_{i=2}^{2^k} \log_2(2w_i^2) \\ &= \sum_{i=2}^{2^k} (1 + 2\log_2 w_i) = 2^k - 1 + 2\sum_{i=2}^{2^k} \log_2 2^{\lfloor \log_2(i-1) \rfloor} \\ &= 2^k - 1 + 2\sum_{i=2}^{2^k} \lfloor \log_2(i-1) \rfloor = 2^k - 1 + 2\sum_{j=1}^{k-1} j2^j \\ &= 2^k(2k-3) + 3\end{aligned} \tag{3.5}$$

We now have a lower bound $c(k) = 2^k(2k-3) + 3 < \sum_{i=1}^{2^k} \log_2 \ell_i$ on the cut edge contribution to μ_0 for a pair of consecutive tiles. Let $\hat{\alpha}_k = \log_2 \alpha_k$ denote the logarithm of the lower bound α_k on μ_0. Combining Equation 3.2 and Equation 3.5, we obtain the recurrence

$$\begin{aligned}\hat{\alpha}_2 &= \frac{1}{24} \log_2(3^9 \times 5 \times 7) \\ \hat{\alpha}_k &= \frac{1}{E_k}\Big(4E_{k-1}\hat{\alpha}_{k-1} + a(k-2) + 3c(k-2)\Big)\end{aligned} \tag{3.6}$$

where $\hat{\alpha}_2$ is given by Figure 3.1(a). Solving this recurrence,[3] we obtain

$$\alpha_k = 4^{\frac{1}{64}(33+32^{k-4}-\frac{64k+7}{2^{k-1}})} 3^{\frac{1}{16}(3+\frac{1}{2^{k-1}})} 5^{\frac{1}{32}(1+\frac{1}{2^{k-1}})} 7^{\frac{1}{16}(1-\frac{1}{2^{k-1}})} > 4^{\frac{7}{8}-\frac{k}{2^{k-1}}} \tag{3.7}$$

One may easily verify that this solves the recurrence for $k \geq 2$, and that the weaker bound $4^{\frac{7}{8}-\frac{k}{2^{k-1}}}$ holds also for $k = 1$. Clearly, as k, and thus the grid size, approaches infinity, α_k approaches $4^{\frac{7}{8}} \simeq 3.364$.

3.1.2. Upper bound. A trivial upper bound β on μ_0 for *any* hierarchical layout may be obtained using the fact that the length of a cut edge on level k is bounded by the number of vertices V_k on that level:

$$\beta_k = \exp\left(\frac{1}{E_k}\Big(4E_{k-1}\log\beta_{k-1} + 2^{k+1}\log V_k\Big)\right) < 4^{2-\frac{k}{2^{k-1}}} < 16 \tag{3.8}$$

A tighter bound can be found using a strategy similar to the one employed for the lower bound, i.e. by considering the "worst" layout of a 4×4 grid, as shown in Figure 3.1(b). However, instead of using the lower bound $(\ell - 1)V_k$ on cut edge lengths, we use the upper bound $(\ell + 1)V_k$, where $\ell \in \{5, 5, 8, 8, 9, 9, 11, 13\}$. We thus have as cut edge contribution

$$\begin{aligned}b(k) &= 2^k(16k + 2\log_2 6 + 2\log_2 9 + 2\log_2 10 + \log_2 12 + \log_2 14) \\ &= 2^k(16k + \log_2 382725 + 7)\end{aligned} \tag{3.9}$$

This results in the recurrence

$$\begin{aligned}\hat{\beta}_2 &= \frac{1}{24} \log_2(2^{14} \times 3^8 \times 5^2 \times 11 \times 13) \\ \hat{\beta}_k &= \frac{1}{E_k}\Big(4E_{k-1}\hat{\beta}_{k-1} + b(k-2)\Big)\end{aligned} \tag{3.10}$$

[3] We used *Mathematica's* `RSolve`.

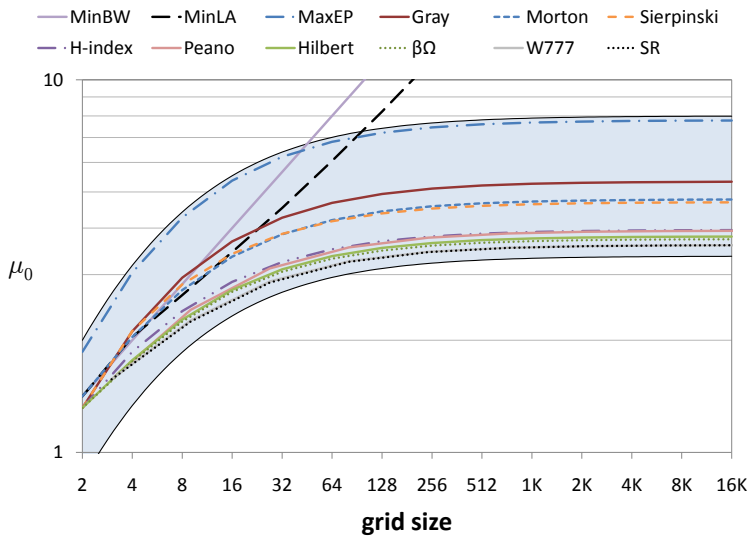


FIG. 3.2. *Upper and lower bound (shaded region) on μ_0 for hierarchical layouts of 2D Cartesian grids as a function of the number of grid points per coordinate axis. Shown also are plots for several (hierarchical) space-filling curves, and two non-hierarchical layouts (*MINBW *and* MINLA*).* MAXEP *corresponds to recursive application of the production rule implied by Figure 3.1(b).*

Using the same technique as above, one may show that $\beta_k < 4^{\frac{3}{2}-\frac{k}{2^k-1}} < 8$.

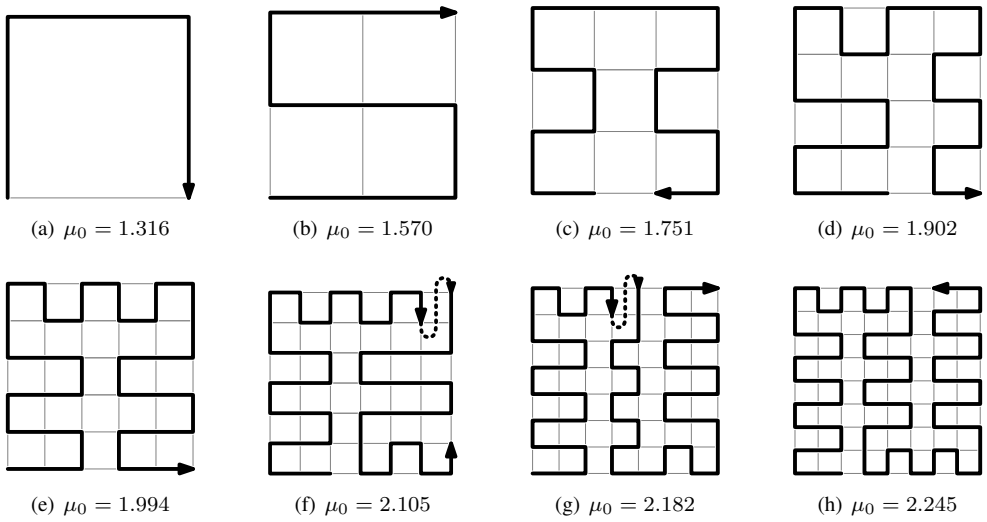
To summarize, a hierarchical layout of a $2^k \times 2^k$ grid satisfies $\mu_0 \in (4^{\frac{7}{8}-\frac{k}{2^k-1}}, 4^{\frac{3}{2}-\frac{k}{2^k-1}})$. This is a quite remarkable result—it implies that one may entirely ignore the local ordering of grid vertices and just recursively partition the grid into quadrants, with the only constraint being that the vertices in each quadrant are stored consecutively, and still obtain a layout whose μ_0 is $O(1)$. In fact, this is the approach taken by Tchiboukdjian et al. [52] in their FASTCOL geometric recursive partitioning method. Moreover, this property of hierarchical layouts hints at a strategy for constructing a layout that minimizes the edge product, which we will exploit in Section 4.

Our bounds are plotted in Figure 3.2, which also shows μ_0 plots for several space-filling curves. (These curves are illustrated in Figure 5.4 and Figure 5.5.) Note that the bounds are guaranteed to hold only for grids of size $2^k \times 2^k$, though the approach taken here can easily be applied, for instance, to derive bounds for triadic grids, on which layouts such as the Peano curve are defined. Figure 3.2 suggests that our bounds are reasonably tight, as evidenced by their proximity to known instances of space-filling curves.

3.2. Optimal grid orderings. It is natural to ask what orderings minimize μ_0 for various classes of graphs. In this section we focus on graphs corresponding to 2D Cartesian grids. We have seen above that hierarchical layouts of $2^k \times 2^k$ grids result in a low, bounded μ_0. We here examine square grids of more general dimensions, $n \times n$, for small values of n.

To find the optimal orderings of small grids, we developed a simple branch-and-bound method that exhaustively enumerates all permutations of the vertices in a grid, and quickly discards collections of permutations that are known to exceed the optimum found so far. This approach allowed us to determine the optimal layouts of grids up to 5×5 vertices. These optimal grids are shown in Figure 3.3.

For larger grids, we used a more sophisticated multilevel optimization strategy, as outlined in detail in Section 4. Because this method is not guaranteed to find the global optimum,



(a) $\mu_0 = 1.316$ (b) $\mu_0 = 1.570$ (c) $\mu_0 = 1.751$ (d) $\mu_0 = 1.902$

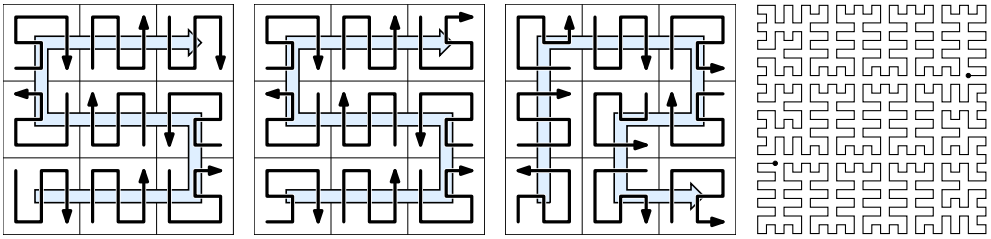
(e) $\mu_0 = 1.994$ (f) $\mu_0 = 2.105$ (g) $\mu_0 = 2.182$ (h) $\mu_0 = 2.245$

FIG. 3.3. *Optimal layouts of 2D grids of dimensions $n \times n$ for $2 \leq n \leq 9$. The optimality of cases $2 \leq n \leq 5$ has been verified exhaustively, while the remaining grids are conjectured optimal based on a large number of computer experiments.*

we made an extensive number of runs starting from different initial configurations, which nevertheless frequently resulted in the same layout. We also attempted to further optimize these layouts by hand, e.g. to enforce symmetry or Hamiltonicity, but were not able to improve the layouts found via computation. Due to the large number of runs explored, we conjecture that the layouts for $6 \leq n \leq 9$ are also globally optimal.

Beyond $n = 9$ it is difficult to find optimal layouts reliably, due to the exponentially increasing search space. However, based on the best layouts we have been able to find, we make the following general observations:

1. *The μ_0-optimal layouts are Hamiltonian—or very nearly Hamiltonian—paths*. This is in stark contrast with $n \times n$ regular grid layouts that minimize μ_1 (MINLA) and μ_∞ (MINBW), which have $\Theta(n)$ pairs of consecutive vertices that do not share an edge. This property is also reflected in the μ_0 plots of space-filling curves in Figure 3.2: Hamiltonian curves like the H-index [38], β-Ω [54], and Peano, Hilbert, and Wunderlich [46] all yield lower μ_0 than non-Hamiltonian curves, such as Gray code [12], Morton (aka. Lebesgue), and Sierpiński [46].
2. *μ_0 favors layouts that are (roughly) hierarchical*. That is, the grids can generally be partitioned recursively into smaller subgrids whose vertices are ordered consecutively. Somewhat surprisingly, the optimal ordering of an 8×8 grid (Figure 3.3(g)) is neither dyadic nor Hamiltonian. The dyadic β-Ω layout is, however, at $\mu_0 = 2.244$ quite close to optimal (in fact, it is the optimal hierarchical layout of an 8×8 grid). This property, which we further explored above, confirms that space-filling and other hierarchical layouts have good locality. In particular, unlike MINLA and MINBW, hierarchical layouts traverse all dimensions of the domain at roughly the same rate, without a clear dominant direction.
3. *Subgrids of size 3×3 are generally favored over 2×2 subgrids*. As evidenced by Figure 3.3, as $n \geq 6$, the predominant ordering pattern consists of S-like or R-like 3×3 templates (see also Figure 3.4), even in the case $n = 2^3 = 8$, which does not evenly divide 3. In fact, the optimal 9×9 grid, which is a triadic hierarchical layout,



(a) Endpoint S rule (b) Interior S rule (c) Interior R rule (d) 27×27 SR curve

FIG. 3.4. *Production rules for the SR space-filling curve.*

yields $\mu_0 = 2.245$, which only slightly exceeds $\mu_0 = 2.244$ for the 8×8 dyadic β-Ω layout, in spite of being a larger grid with nearly 30% more edges.

This last observation suggests the existence of a new space-filling triadic layout with improved locality properties, which we discuss in the following section.

3.3. The SR space-filling curve. A striking result of our study of $n \times n$ grids is that the best known layout of a 9×9 grid is a hierarchical curve based upon a combination of the Wunderlich serpentine (S) and meandering (R) curves (see [46, 56]). That is, one obtains the 9×9 curve by productions that turn the 3×3 Peano S curve into a 3×3 array of 3×3 curves with S and R shapes (see Figure 3.4(a)). Note that we cannot recursively apply this one single rule, since (1) the endpoints of the 9×9 curve are not at grid corners, and therefore do not not admit a Hamiltonian path, and (2) a separate rule is needed to refine the R-shaped template. To address both of these issues, we constrained the entrance and exit points to the corners of a 9×9 grid, and exhaustively computed the optimal hierarchical layouts (with respect to μ_0) consisting only of S and R templates. The resulting productions are shown in Figure 3.4(b) and Figure 3.4(c), with Figure 3.4(d) showing one more level of refinement of the curve. Because this hierarchical layout is made up of S- and R-shaped templates, we call it the *SR space-filling curve*.

Note that one may substitute the top-level endpoint rule with the interior S rule with little loss in locality, which both simplifies the construction and produces a curve that begins and ends at grid corners. This may be beneficial for designing hybrid dyadic-triadic layouts of grids whose dimensions are not powers of three. For instance, the optimal 6×6 grid (Figure 3.3(e)), which consists only of S and R templates, could using these rules be refined into an 18×18 grid. Similarly, each 2×2 template in the Hilbert and β-Ω curves can at any level of refinement be substituted with an R-shaped 3×3 template.

4. Optimization algorithm. We now turn our attention to the optimization problem of finding the permutation of nodes that minimizes the edge product functional for general undirected graphs. As is common in graph ordering, we employ a multilevel approach inspired by algebraic multigrid (AMG) methods [41], and in particular we base our algorithm on the weighted contraction framework of Safro et al. [45], with specializations for the case $p = 0$. The main idea is to perform a sequence of coarsening steps, to find a near optimal arrangement of the coarsest graph, and to then incrementally refine the graph by reinserting nodes and to optimize its layout on each level. The process of graph coarsening followed by refinement is called a *V cycle*, and usually a sequence of several V cycles is performed.

The main difference between Safro et al.'s algorithm and those based on unweighted edge contraction (e.g. [29]) is that each node removed in the coarsening process is not necessarily grouped with a single coarse node. Rather, such nodes are partitioned into fractional units

that may be assigned to multiple neighbors in the next coarser graph. How big a fraction that gets aggregated with a neighbor depends on how closely the two nodes are *bonded*, which in turn depends on the current layout of the graph. Safro et al. show that, by avoiding binary decisions of whether or not to group two nodes, this continuous, weighted edge contraction approach to aggregation generally leads to improved layouts for several ordering problems, including 1-sum, 2-sum, and bandwidth minimization. We here describe our variation on Safro et al.'s algorithm in detail.

Let $G = (V, E)$ be an undirected graph with nodes V and edges E. Each edge $ij \in E$ may have an associated positive weight w_{ij} (for unweighted graphs, this weight is uniformly equal to one). We use the convention that $w_{ij} = 0$ if $ij \notin E$. The *neighbors* of a node i are the set $\{j : ij \in E\}$. With each node i we associate a *length* ℓ_i, which represents the number of finest-level nodes aggregated with i. Thus, nodes in the uncoarsened graph have $\ell_i = 1$.

4.1. Initial node ordering. As the combinatorial space of all orderings is potentially huge, we do not expect that a deterministic algorithm be able to navigate around local minima and find an optimal solution in finite time. Rather, we explore different parts of the solution space through multiple invocations of the algorithm, each seeded with a random permutation of the nodes. The coarsening process is weakly dependent on this initial ordering of nodes, which allows different aggregations to be considered.

4.2. Coarsening. Each V cycle $k = 1, \ldots$ begins by coarsening the graph. The coarsening process attempts to group nodes strongly bonded together, with bonds β_{ij} associated with the edges of the graph. These bonds depend on the edge weights w_{ij} (if any) and the ordering φ found in the previous V cycle, so that any prior ordering is not completely undone in subsequent V cycles. Formally, $\beta_{ij} = w_{ij}|\varphi(i) - \varphi(j)|^{-\alpha(k)}$, where $\alpha(k) = \frac{2k}{k+1}$. Thus, in the first V cycle $\alpha(1) = 1$, and in the limit α approaches two. This weighting scheme, which favors adjacent nodes with large weights and that are close in the current linear layout, is the $p = 0$ analogue to the $p = 1$ specialization employed by Safro et al. [43] for the minimum linear arrangement problem.

We then begin coarsening by partitioning the graph nodes V into a set of coarse nodes C that are retained on the next level and a remaining set of fine nodes $F = V \backslash C$. Initially all nodes on the current level are assigned to F, and we progressively move nodes over to C until $|F| \simeq |C|$. The next node to be added to C is the one most strongly bonded to nodes in F relative to nodes in C, i.e. the node $i \in F$ with maximal

$$\beta_i = \sum_{j \in F} \beta_{ij} - \sum_{j \in C} \beta_{ij}. \tag{4.1}$$

Nodes i with large β_i have a significant influence on its neighbors in F, and constitute good representatives for those nodes in the coarsened graph. Our implementation uses a dynamic priority queue to order the node migrations. Thus, once i is moved from F to C, we recompute β_j for each neighbor $j \in F$ by subtracting off $2\beta_{ij}$ per Equation 4.1 and update the priority queue. Note that in this process β_j may eventually become negative, i.e. when j is more closely bonded to C than F. We terminate the node migration when the maximal bond is negative, at which point F and C contain roughly equal numbers of nodes. We note that our coarsening step slightly differs from the one in [45] in the definition of migration priorities and migration criterion.

4.3. Aggregation. Following this partitioning step, we form aggregations by assigning parts of nodes in F to the surviving nodes in C. This step is the same as in [45], and essentially amounts to forming an AMG interpolation matrix P, whose $|V|$ rows correspond to nodes in the uncoarsened graph and whose $|C|$ columns are associated with aggregates V' in the

coarsened graph. The elements of P are thus fractions of nodes in V that make up aggregates in V'. For a given coarse node $j \in C$, let $j' \in V'$ denote its corresponding column in P. We then define the elements of P as

$$p_{ij'} = \begin{cases} 1 & \text{if } i \in C \text{ and } i = j \\ \frac{\beta_{ij}}{\sum_{k \in C} \beta_{ik}} & \text{if } i \in F \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

As in [45], we exclude very small bonds from the above computation by thresholding them to avoid introducing a large number of weak connections in the coarse graph that do not greatly influence the final layout. We do this primarily to limit memory requirements and computation time. Given P, we compute node lengths and edge weights and bonds for the coarsened graph as follows:

$$\ell_u = \sum_{i \in V} p_{iu} \ell_i \qquad w_{uv} = \sum_{ij \in E} p_{iu} p_{jv} w_{ij} \qquad \beta_{uv} = \sum_{ij \in E} p_{iu} p_{jv} \beta_{ij} \tag{4.3}$$

Since the sum of each row in P is one, the total node lengths and edge weights and bonds are thus preserved.

4.4. Node placement. Coarsening proceeds until fewer edges than nodes remain, or until the number of nodes is less than or equal to the width of the *optimization window* (see below). Once the coarsening terminates, an initial node placement is computed. Since coarse nodes are represented as line segments with non-unit length, we place these nodes so that their end points abut and center them on (possibly non-integer) positions x, such that for two consecutive nodes i and j we have

$$x_j = x_i + \tfrac{1}{2}(\ell_i + \ell_j). \tag{4.4}$$

The initial placement of coarsest-level nodes is arbitrary; given some permutation of nodes, we compute positions according to Equation 4.4.

4.5. Relaxation. As in [45], we relax the layout by performing per-node optimal placement. In this step, we freeze the positions of all nodes but one and find the optimal position of that node with respect to the others. We here momentarily allow the placement of nodes so that they may overlap, accepting any value for the position x_i of node i. Following the relaxation step, we sort the nodes on their positions (with no particular rules for breaking ties) to obtain a permutation, and then adjust the positions to avoid overlaps according to Equation 4.4.

For a node i under consideration, we seek to compute

$$x_i = \operatorname*{argmin}_x \prod_{j\,:\,ij \in E} w_{ij} |x - x_j|, \tag{4.5}$$

which is a continuous optimization problem involving a single variable x. Clearly all of the neighbor positions x_j are minima of this function. In fact, one can show that for any p-mean functional μ_p with $p < 1$, these are the only minima. For $0 < p < 1$, each term $|x_i - x_j|^p$ is quasiconvex (i.e. its second derivative is negative everywhere except at x_j, where it is minimal and not differentiable). Similarly, the log terms $\log |x_i - x_j|$ in μ_0 are also quasiconvex. When $p < 0$, $|x_i - x_j|^p$ is quasiconcave, i.e. x_j maximizes this term. The outer negative exponent $1/p$ in μ_p then turn local maxima into minima. Figure 4.1 illustrates this property for several choices of p.

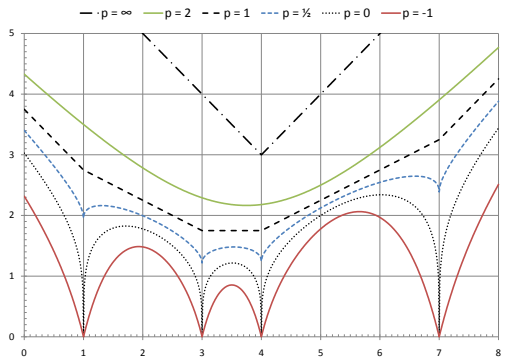


FIG. 4.1. *μ_p as a function of the continuous position x_i of a node i with neighbors positioned at* $\{1, 3, 4, 7\}$.

The above observation implies that to place i we need consider only the positions of its neighbors. As we have seen, the tendency of the geometric mean measure μ_0 to prefer Hamiltonian paths (i.e. sequences for which each pair of consecutive nodes is joined by an edge) may be explained in part by the fact that the optimal placement of a node i is right at (or just before or after, when correcting for overlaps) one of its neighbors, *regardless of the relative positions of i's neighbors*. In contrast, the optimal placement for $p = 1$ is the (weighted) median neighbor position; for $p = 2$ the optimum is the (weighted) mean; and for $p = \infty$ the optimum is the midrange. For these functionals, roughly half of the neighbors will on average appear before a given node, while the other half will appear after, leading to orderings that are essentially one-dimensional "sweeps" over the graph (see, e.g., the spectral mesh layouts in [24]).

In choosing which neighbor j to place i at, we exclude any zero factors from the product in Equation 4.5. This approach is essentially equivalent to choosing the neighbor for which the magnitude of the derivative of μ_0 with respect to the position x of i is (in the limit) minimal. Our choice is motivated by the fact that any overlapping nodes will ultimately be separated, thus $\mu_0(x) \neq 0$ will depend largely on the magnitude of its derivative.

As in the method of Safro et al., we perform two separate passes of relaxation: compatible and Gauss-Seidel. The first involves placement only of fine nodes while keeping the coarse nodes fixed; in Gauss-Seidel relaxation both fine and coarse nodes are optimized. After each sweep over the nodes to be optimized, we adjust positions to avoid overlaps.

4.6. Windowed optimization. Following the rather global reordering implied by relaxation, we fine-tune the local ordering using windowed optimization. The optimization window determines the number of consecutive nodes n for which we compute an optimal arrangement, i.e. by considering all $n!$ permutations. We thus sweep this window in overlapping steps over the nodes in their current layout, freeze all remaining $|V| - n$ nodes in the graph, and explicitly evaluate the (weighted) edge product functional for each n-node permutation. We keep the best such permutation and advance the window one node at a time. Note that evaluating each n-node permutation involves locally reordering nodes into non-overlapping segments.

Since solving this problem on coarse levels is relatively much cheaper than on fine levels, we can afford to increase the width of the optimization window as we move up to coarse levels, such that the amount of work $O(n!(|V| + |E|))$ per level is roughly constant; the most important ordering decisions are often made on coarse levels. As a result, the coarse-level problems are solved nearly globally optimally, whereas on fine levels this windowed optimization locally fine-tunes the ordering and untangles the graph where poor tie breaking

decisions were made in the prior relaxation steps. Usually we specify n for the finest level and widen the window appropriately on the coarser levels.

Although potentially more expensive to perform than solving a constrained quadratic program, as in [45], we found a finest-level optimization window on the order of a handful of nodes to be sufficient. Moreover, this combinatorial optimization problem is straightforward to solve exactly, with no additional numerical issues or false minima to consider.

4.7. Refinement. Once a coarse-level layout has been established, the graph is refined to its next finer level. This refinement can essentially be thought of as a completion of the migration process from F to C. First, the (shortened) coarse nodes C are placed according to their (segment midpoint) positions in the next coarser graph, which due to conservation of total node length results in gaps between coarse nodes that may be filled with fine nodes. The nodes of F are placed one at a time using the compatible relaxation strategy above (i.e. keeping all coarse and previously placed fine nodes fixed). As in the case of coarsening, the order in which fine nodes are placed is dictated by the connectivity with already placed nodes. Finally, the layout is corrected for overlaps, resulting in an initial ordering of nodes for the current level. We then proceed with relaxation and windowed optimization until finally the finest level has been ordered, thus completing the V cycle. We then recompute the bonds (Section 4.2) before proceeding with the next V cycle.

4.8. Optimization schedule. Our general approach is to begin optimization with a small optimization window of, say, four nodes to obtain a rough ordering, and to periodically between V cycles increment the width of the window for fine-tuning. We evaluate the cost of each finest-level layout computed and maintain the best layout found so far. Since the first-cycle ordering largely dictates the global structure of the layout, and since our method employs no stochastic optimization, it is likely that any given invocation of our algorithm will yield a suboptimal solution. Hence, as is common, we recommend executing multiple runs with different random seeds (for the initial bond computation) to explore different "search directions" in the vast combinatorial optimization space.

5. Results. In this section we demonstrate many applications of using μ_0 as a quality measure, as well as desirable properties of graph and mesh layouts optimized for this measure. In particular, we investigate applications in dimensionality reduction, cache-friendly data organization, and graph and mesh partitioning. We will furthermore show that our new ordering method outlined in Section 4 produces layouts that are superior to those generated by OpenCCL [59]; our previous heuristic ordering method for the same measure.

5.1. Spatial locality and connectedness. Linear graph ordering finds applications in dimensionality reduction, where the goal is to map a discrete set of nodes of a graph embedded in d dimensions, say $\mathcal{D} = \mathbb{R}^d$, to a lower-dimensional space $\mathcal{R} = \mathbb{R}^r$, $r < d$. We will focus on the case $\mathcal{R} = \{1, \ldots, |V|\}$. Usually the goal in dimensionality reduction is to preserve relative distances to the extent possible. For graphs without a geometric embedding, or for spaces without a well-defined metric, a related goal is to preserve *locality*, in the sense that adjacent nodes in $\mathcal{D}$ are also adjacent (i.e. consecutive) in $\mathcal{R}$, and vice versa. Note that the number of consecutive pairs $|V| - 1$ is usually smaller than the number of edges $|E|$ in the graph, making preservation of adjacency impossible in general. In particular, although constructing a sequence of nodes such that consecutive pairs are shared by an edge—the Hamiltonian path problem—is often possible, it is generally not possible to map each graph edge to a consecutive vertex pair. Moreover, although orderings that are Hamiltonian are generally preferable, they do not necessarily imply good locality in the sense of preserving distances in $\mathcal{D}$. For instance, a Hilbert curve traversal of a grid generally pre-

serves distances better than a raster-snake (row-by-row, back-and-forth) traversal, in spite of both being Hamiltonian paths [15].

To capture locality in both directions, we define the *edge connectedness* and *cell connectedness* of a mesh $M = (V, E, C)$, where C is a set of *cells*, e.g. polygons in 2D and polyhedra in 3D. Edge connectedness measures the number of consecutive vertex pairs that are also edges in the mesh with respect to the maximum possible (i.e. $|V| - 1$). Note that an equivalent definition is the number of mesh edges of "length" one:

$$\begin{aligned} econ &= \frac{1}{|V|-1}\big|\{i \in V : ij \in E,\, j = \varphi^{-1}(\varphi(i)+1)\}\big| \\ &= \frac{1}{|V|-1}\big|\{ij \in E : |\varphi(i) - \varphi(j)| = 1\}\big|. \end{aligned} \tag{5.1}$$

That is, $econ$ is the (normalized) number of edges that are connected in $\mathcal{R}$. Thus $econ = 1$ if and only if φ is a Hamiltonian path.

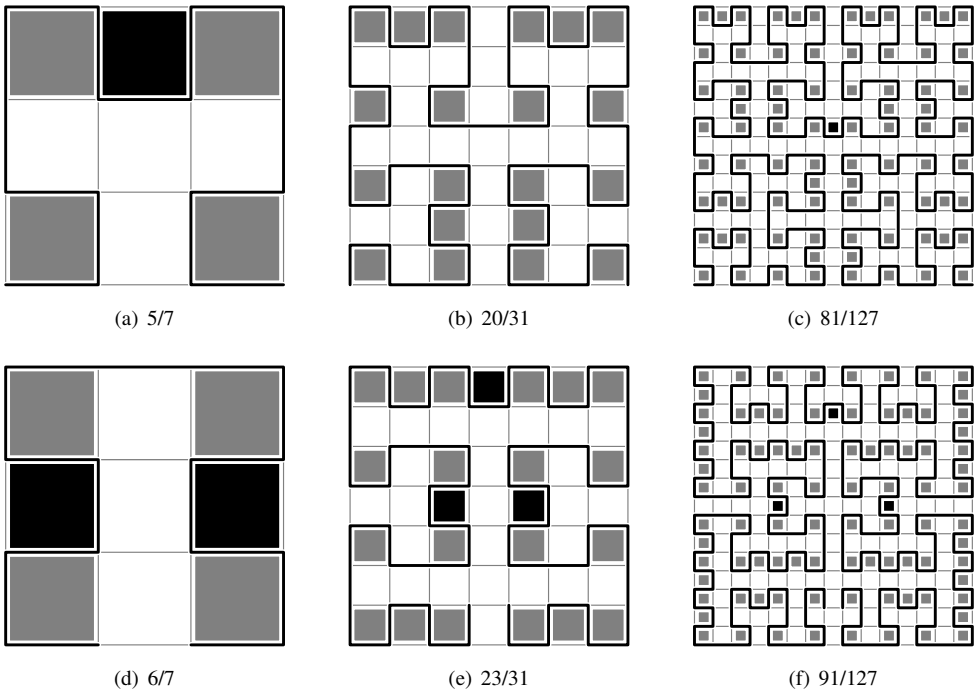
Edge connectedness in essence measures how many nodes in $\mathcal{R}$ have neighbors (immediate predecessors and successors) that are also neighbors in $\mathcal{D}$ (share an edge). To derive an analogue in the opposite direction, we first must define which sets of vertices to consider as neighbors and what it would mean to preserve adjacency. For instance, on a 2D Cartesian grid it is impossible to place a node and its four neighbors in consecutive positions without breaking Hamiltonicity. And as we have seen, measuring the number of adjacent vertices in $\mathcal{D}$ that are adjacent in $\mathcal{R}$ is equivalent to edge connectedness. Instead, we define *cell connectedness* as the number of cells whose vertices form a contiguous sequence in $\mathcal{R}$. Since all polygonal and many polyhedral cells admit Hamiltonian paths,[4] this is a reasonable complementary measure of locality for meshes. In particular, this measure accounts for the desired property of locally traversing all d dimensions without an explicit reference to a geometric embedding.

We define cell connectedness with respect to an upper bound on the maximum number of connected cells C_{max}. Two consecutive connected cells with n vertices each may share at most an m-vertex face in 3D and an edge in 2D, and may "overlap" in $\mathcal{R}$ by at most m vertices. Thus for meshes with uniform cell type $C_{max} = \left\lfloor \frac{|V|-m}{n-m} \right\rfloor$ and

$$ccon = \frac{1}{C_{max}}\big|\{c \in C : \max_{i,j \in c} |\varphi(i) - \varphi(j)| = n - 1\}\big|. \tag{5.2}$$

We note that edge connectedness is simply an instance of cell connectedness, for which the mesh edges constitute the cells and $m = 1, n = 2$. Furthermore, whereas edge connectedness incorporates only locality in one dimension (along edges), cell connectedness accounts for locality in all d dimensions spanned by the cells. In other words, cell connectedness puts a premium on locally traversing all dimensions at roughly the same rate (which many recursive space-filling curves do) and penalizes traversals biased along a single, dominant direction (as lexicographic ordering does). This generally makes cell connectedness a better discriminator of space-filling curves, many of which are Hamiltonian and thus already have maximum unit edge connectedness. Furthermore, to achieve a high cell connectedness, cell vertex sequences should overlap on shared faces, such that the traversal visits adjacent cells and their vertices in order. An ideal traversal is thus Hamiltonian over both the primal and dual graph.

[4] All pyramids (including tetrahedra), prisms (including hexahedra), and Platonic solids (including octahedra) are Hamiltonian.



(a) 5/7 (b) 20/31 (c) 81/127

(d) 6/7 (e) 23/31 (f) 91/127

FIG. 5.1. *Connected cells (shaded) are cells whose vertices appear in contiguous locations in the linear ordering, and thus exhibit high locality. The number of connected cells out of the maximum possible is listed for Hilbert (top) and β-Ω (bottom) layouts of grids of size $2^k \times 2^k$. Dark shaded cells connect subgrids of size $2^{k-1} \times 2^{k-1}$, and exist only when k is even in Hilbert layouts.*

5.1.1. Regular grids. Figure 5.1 demonstrates cell connectedness for two layouts of a 2D grid. For space-filling curves like these, it is fairly easy to derive the cell connectedness in closed form, since for each of the three pairs of consecutive quadrants at each level of refinement one has to examine only the connectedness of the cell that "bridges" the quadrants (e.g. the dark shaded cells), i.e. where the curve exits one quadrant and enters the next. Table 5.1 lists the edge and cell connectedness for several well-known layouts of 2D infinite grids. These layouts are illustrated in Figure 5.4 and Figure 5.5. The triadic Z_3 curve is a generalization of Morton's dyadic curve using a radix of three. The triadic *W* curves correspond to variants of Wunderlich's space-filling curve. We use three octal digits to encode the orientation of the nine subgrids [46]. The *WR* curve is Wunderlich's "meandering" curve based on R-like productions (see Figure 5.5(d)).

Table 5.1 also includes the Euclidean locality measure

$$WL_2 = \max_{i,j \in V} \frac{\|C(i) - C(j)\|_2^2}{|\varphi(i) - \varphi(j)|} \tag{5.3}$$

suggested by Gotsman and Lindenbaum [15], where $C(i)$ denotes the 2D integer coordinates of vertex i, and the worst-case bounding box square perimeter ratio of Haverkort et al. [18]

$$WBP = \frac{1}{4} \max_{i,j \in V} \frac{\max\limits_{k \in V_{ij}} C(k)_1 - \min\limits_{k \in V_{ij}} C(k)_1 + \max\limits_{k \in V_{ij}} C(k)_2 - \min\limits_{k \in V_{ij}} C(k)_2}{|\varphi(i) - \varphi(l)| + 1} \tag{5.4}$$

where $V_{ij} = \{k \in V : \varphi(i) \le \varphi(k) \le \varphi(j)\}$. We see that low values of μ_0, WL_2,

(a) $2^k \times 2^k$ grids

layout	μ_0	WL_2	WBP	$econ$	$ccon$
β-Ω	$\mathbf{3.723}$	5	$\mathbf{2\frac{1}{4}}$	$\mathbf{1}$	$\mathbf{\frac{29}{40} = 0.725}$
Hilbert	3.785	6	$2\frac{2}{5}$	$\mathbf{1}$	$\frac{19}{30} = 0.633...$
H-index	3.940	$\mathbf{4}$	3	$\mathbf{1}$	$\frac{2}{3} = 0.666...$
Sierpiński	4.677	$\mathbf{4}$	3	$\frac{2}{3}$	$\frac{1}{3} = 0.333...$
Morton	4.759	∞	∞	$\frac{1}{2}$	$\frac{1}{2} = 0.5$
Gray code	5.311	∞	∞	$\frac{3}{4}$	$\frac{1}{2} = 0.5$
MINLA	43.692	∞	∞	$\mathbf{1}^*$	$0 = 0.0$
MINBW	64.000	∞	∞	$\mathbf{1}^*$	$0 = 0.0$

(b) $3^k \times 3^k$ grids

layout	μ_0	WL_2	WBP	$econ$	$ccon$
SR	$\mathbf{3.591}$	$\mathbf{5}$	$\mathbf{2\frac{9}{20}}$	$\mathbf{1}$	$\mathbf{\frac{2}{3} = 0.666...}$
W777	3.593	$6\frac{2}{3}$	$2\frac{2}{3}$	$\mathbf{1}$	$\frac{3}{5} = 0.6$
W252	3.839	$6\frac{2}{3}$	$2\frac{10}{11}$	$\mathbf{1}$	$\frac{4}{9} = 0.444...$
WR	3.914	$6\frac{2}{3}$	$2\frac{2}{3}$	$\mathbf{1}$	$\frac{17}{36} = 0.4722...$
W000	3.928	8	$2\frac{13}{18}$	$\mathbf{1}$	$\frac{1}{2} = 0.5$
Z_3	4.748	∞	∞	$\frac{2}{3}$	$0 = 0.0$
MINLA	55.219	∞	∞	$\mathbf{1}^*$	$0 = 0.0$
MINBW	81.000	∞	∞	$\mathbf{1}^*$	$0 = 0.0$

TABLE 5.1

Locality measures for several layouts of (a) dyadic and (b) triadic 2D Cartesian grids. The μ_0 values were computed for grids of size $2^{12} = 4,096$ and $3^8 = 6,561$ squared, respectively, while the remaining measures were computed in closed form for infinite grids. Boldface numbers indicate the best locality among all layouts.

layout	bunny				torso				steven		
	μ_0	AL_1	$econ$	$ccon$	μ_0	AL_1	$econ$	$ccon$	μ_0	$econ$	$ccon$
MINEP	$\mathbf{4.393}$	$\mathbf{0.721}$	$\mathbf{0.961}$	$\mathbf{0.551}$	$\mathbf{18.933}$	$\mathbf{0.457}$	$\mathbf{0.978}$	$\mathbf{0.265}$	$\mathbf{7.769}$	$\mathbf{0.959}$	$\mathbf{0.468}$
OPENCCL	5.286	0.747	0.886	0.383	25.133	0.478	0.928	0.164	9.126	0.841	0.189
Hilbert	7.552	0.904	0.737	0.234	37.193	0.500	0.788	0.059	15.605	0.581	0.027
Morton	7.581	0.910	0.666	0.207	43.254	0.549	0.633	0.024	20.468	0.433	0.004
FASTCOL	11.000	1.000	0.571	0.111	55.775	0.599	0.491	0.019	34.004	0.367	0.000
MINLA	15.195	0.940	0.510	0.009	76.283	0.493	0.613	0.006	14.197	0.845	0.003
ONMETIS	37.566	0.864	0.193	0.007	124.106	0.576	0.222	0.003	162.691	0.017	0.000
MINBW	72.966	1.532	0.060	0.001	835.630	0.970	0.076	0.000	138.561	0.011	0.000

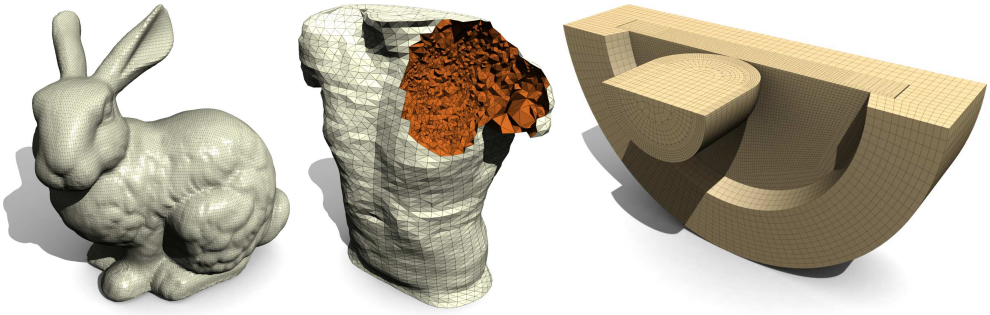
TABLE 5.2

Statistics for unstructured meshes (cf. Table 5.1). AL_1 denotes the average locality (lower is better).

and WBP generally imply high values for edge and cell connectedness. In particular, for layouts like MINBW and MINLA, which both have unit edge connectedness,[5] the complete lack of cell connectedness results in a large increase in μ_0 compared to other layouts. We further note that the best known layouts with respect to μ_0 also exhibit the highest edge and cell connectedness, and that our new SR space-filling curve has the lowest WL_2 and WBP values among $3^k \times 3^k$ grids. In fact, the best μ_0 curves for dyadic and triadic grids have $WL_2 = 5$ in the limit. These results suggest that μ_0 is a suitable indicator of locality, while at $O(|E|)$ being far easier to compute than the $O(|V|^2)$ complexity implied by both WL_2 and WBP. Moreover, μ_0 does not require a geometric embedding, and unlike the other measures is therefore able to assess the locality of any graph layout.

5.1.2. Unstructured meshes. The edge and cell connectedness measures are well defined for any mesh composed of cells of the same type. In our evaluation, we also include unstructured meshes composed of triangles, tetrahedra, and hexahedra, as shown in Figure 5.2, which are representative of problems arising in computer graphics and finite element analysis. Unless otherwise stated, we used the primal graphs of these meshes. These meshes all have a geometric embedding, which allows us to compare our layouts with those produced both by geometry-based methods, like 3D space-filling curves and Tchiboukdjian et al.'s FASTCOL

[5] Although these layouts are not Hamiltonian, they have unit edge connectedness in the limit as the grid size approaches infinity.



(a) bunny
34,834 : 104,288 : 69,451

(b) torso
168,930 : 1,254,712 : 1,082,723

(c) steven
96,030 : 273,506 : 81,832

FIG. 5.2. *Unstructured (a) triangular, (b) tetrahedral, and (c) hexahedral meshes used in our evaluation. The captions list the number of vertices, edges, and cells, respectively.*

method [52], and purely connectivity-based methods, such as the ONMETIS nested disection method [28] and our OPENCCL heuristic for minimizing μ_0 [59]. The MINBW and MINLA layouts were produced by making appropriate modifications to our multilevel layout algorithm for the corresponding layout functionals, as detailed further in [45]. We found these layouts comparable in quality to those produced by Safro et al.'s implementation.

In the left column of Figure 5.3 we show for each mesh and layout the geometric mean (μ_0) and corresponding edge and cell connectedness values. Table 5.2 lists these values in addition to the *average locality* AL_1 in the sense of Gotsman and Lindenbaum [15], replacing Euclidean distance with graph distance and substituting mean for maximum:

$$AL_1 = \frac{1}{(|V|-1)|V|} \sum_{i,j \in V} \frac{D(i,j)^d}{|\varphi(i) - \varphi(j)|} \tag{5.5}$$

Here $D(i,j)$ denotes the shortest path between i and j (in number of edges), and d is the intrinsic dimensionality of the mesh. On Cartesian grids, $D(i,j)$ is simply the L_1 Manhattan distance. We note that for unstructured graphs whose layouts are not Hamiltonian, the WL_2 measure is near arbitrary, and therefore we report the average locality instead. We have omitted the AL_1 values for the steven mesh, which is made up of several disconnected components. By interleaving the components in the 1D layout, as the geometry-based methods do, one is able to further reduce AL_1 by artificially increasing the denominator $|\varphi(i) - \varphi(j)|$. This interleaving puts nodes from different components in consecutive locations. However, such non-neighboring nodes are not penalized, because their graph distance is not well defined (since no path connects the nodes). Thus, while detrimental to locality, such interleaving serves only to reduce AL_1.

Evidently μ_0 is a good predictor of locality and connectedness for unstructured meshes as well. For instance, the Kendall τ rank correlation coefficient between μ_0 and edge connectedness for the three meshes is -1.00, -0.93, and -0.86, while for cell connectedness it is -1.00, -1.00, and -0.82.[6] The layouts produced by our new multilevel algorithm presented in Section 4 (here labeled MINEP) consistently scored highest on both connectedness measures, and also yielded the lowest AL_1 values. Moreover, our new algorithm significantly outperformed OPENCCL (and the other layouts) in terms of μ_0.

[6] A good layout has low μ_0 and high connectedness, and hence the correlation coefficient between the two quantities is negative.

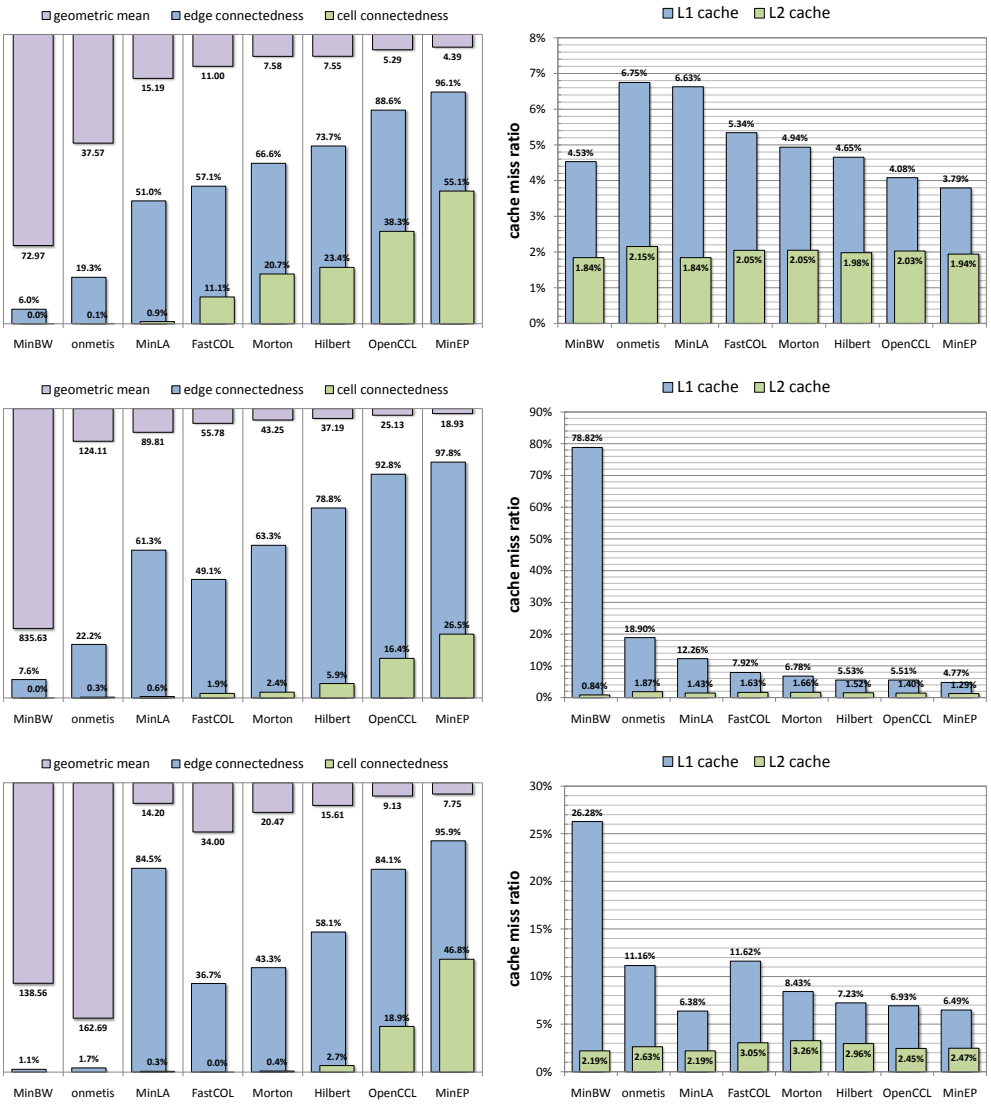
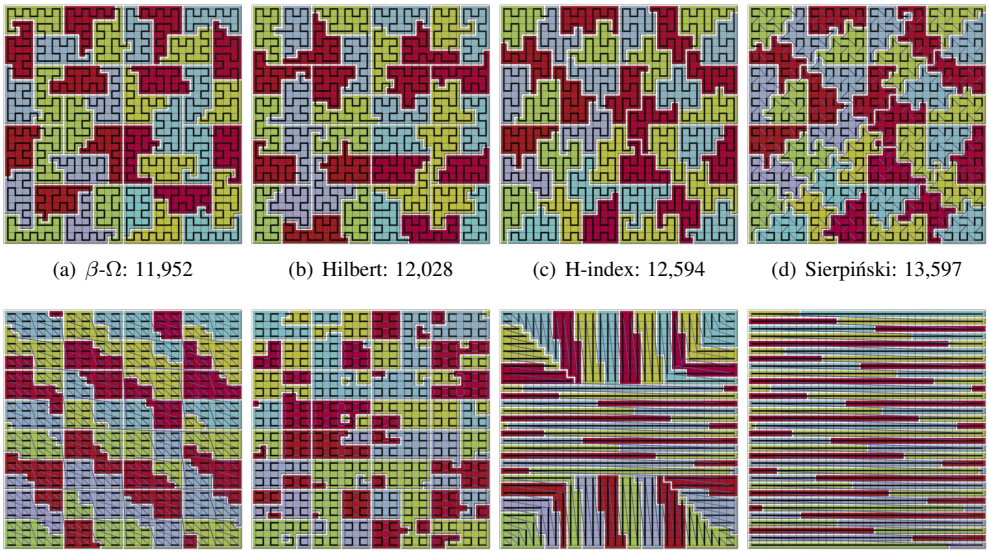


FIG. 5.3. *Geometric mean edge length and connectedness (left) and observed cache misses in matrix-vector products (right) for (from top to bottom) the bunny, torso, and steven meshes.*

5.2. Spatial locality and perimetry. The cell connectedness measure attempts to capture locality in terms of how uniformly and quickly a 1D layout traverses all d dimensions associated with a cell. It does so, however, at the finest granularity possible—a single cell—and applies only to graphs that represent a cell complex. Ideally locality is expressed also at coarser levels of granularity. Intuitively, a contiguous subsequence of the layout exhibits locality when its preimage traces out a compact d-dimensional volume with a roughly uniform extent along each coordinate axis. In other words, we wish for such subsets to minimize the *perimeter-to-volume* ratio. In (infinite) Cartesian grids, the perimeter of a simply connected subset of vertices S is the same as the number of cut edges between S and $\neg S$. Thus, the *edge cut* of a given partition of $\mathcal{R}$ is an indicator of locality. Recall that the edge product measure



(a) β-Ω: 11,952 (b) Hilbert: 12,028 (c) H-index: 12,594 (d) Sierpiński: 13,597

(e) Morton: 13,248 (f) Gray code: 15,008 (g) MINLA: 18,948 (h) MINBW: 27,776

FIG. 5.4. *Dyadic space-filling curves for* 32×32 *grids. The captions include the number of edges cut between blocks of* 27 *contiguous vertices. These block boundaries are shown in white.*

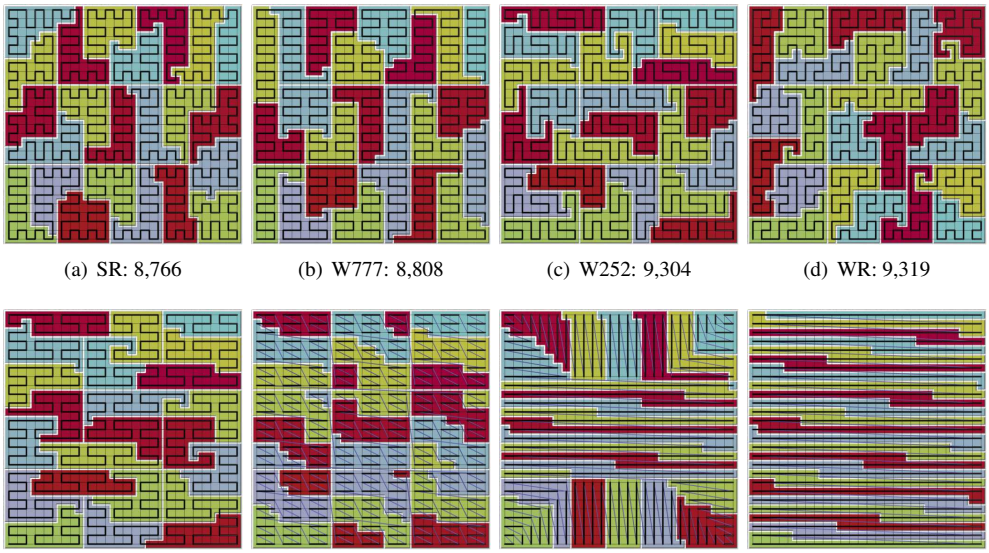
μ_0 was designed exactly to minimize the edge cut for hierarchical, uniform partitions. We will return to the importance of edge cuts below in the context of graph partitioning.

To appreciate the quality of the space-filling curve layouts discussed above, we show in Figure 5.4 and Figure 5.5 the edge cut and resulting perimeter shapes associated with a given choice of block size N. Notice the more compact shapes produced by the best curves (β-Ω and SR) compared to the thin strands associated with lexicographic ordering (MINBW) that primarily extend in only a single dimension.

Figure 5.6 and Figure 5.7 plot the *normalized mean perimeter* as a function of block size N (i.e. interval length in $\mathcal{R}$) for regular grids and unstructured meshes, respectively. We normalize the perimeter (i.e. edge cut) by the "ideal" perimeter $2dN^{\frac{d-1}{d}}$ of a hypercube on a d-dimensional Cartesian grid. For our unstructured meshes, we use the equivalent normalization term $2\frac{|E|}{|V|}N^{\frac{d-1}{d}}$, which yields a unit normalized perimeter when $N = 1$.

We observe that our MINEP layouts, and those regular grid layouts with minimal μ_0, generally result in the smallest perimeter. Indeed, the relative ordering of the perimeter curves is well correlated with the μ_0 values of the corresponding layouts. The main exceptions to this general rule are the MINBW and MINLA layouts, which at high block sizes cut across the other curves. We showed in [57] that MINLA layouts minimize the edge cut when averaged over *all* block sizes (not just over a geometric sequence). In other words, MINLA minimizes the area under the curve in a linear plot (note the logarithmic horizontal axes), and hence the MINLA curves necessarily yield low perimeter values over at least a portion of the range. For data sets partitioned into multiple scales (e.g. domain-decomposed computational meshes), these results hint at a hybrid layout strategy: use μ_0 to order nodes *within* each subdomain at fine scales, and μ_1 to establish an order *across* subdomains at coarse scales.

As observed also in [18], the perimeter plots for the space-filling layouts of Cartesian grids exhibit regular fluctuations whose period is a power of 2 or 3 (depending on the grid type). Interestingly, the phase of these periodic signals varies among the curves.



(a) SR: 8,766 (b) W777: 8,808 (c) W252: 9,304 (d) WR: 9,319

(e) W000: 9,748 (f) Z_3: 9,936 (g) MINLA: 13,372 (h) MINBW: 19,656

FIG. 5.5. *Triadic space-filling curves for* 27×27 *grids. The captions include the number of edges cut between blocks of* 32 *contiguous vertices.*

5.3. Spatial locality and fragmentation. We have seen how the perimeter captures the compactness of contiguous node sets in $\mathcal{R}$. We may turn this around and ask how compact a connected subregion of $\mathcal{D}$ is in $\mathcal{R}$. In other words, given a finite neighborhood $S \subseteq V$ in $\mathcal{D}$ connected by edges, what is the "1D perimeter"of S in $\mathcal{R}$? Equivalently, how many contiguous *fragments* is S partitioned into? A low fragmentation is desired in many applications, as nodes within a fragment are likely to be fetched together (e.g. as a disk block), and each fragment may incur access overhead (e.g. a disk seek operation). Moon et al. [36] studied the fragmentation of several space-filling curves (they refer to fragments as "clusters"), and concluded that the number of fragments of such curves is linear in the $(d-1)$-dimensional perimeter of S. They also found the Hilbert curve to yield the lowest fragmentation.

We here expand on the study by Moon et al. by including several additional space-filling curves, as well as unstructured meshes. Like Moon et al., we also use subregions formed by hypercubes to form connected regions on Cartesian grids. To generalize this approach to unstructured meshes, we parameterize the region by two nodes i and j:

$$R_{ij} = \{k \in V : D(i,k)^2 + D(j,k)^2 \leq D(i,j)^2\} \tag{5.6}$$

where $D(i,j)$ denotes graph distance. In the continuous setting (using Euclidean distance in place of D), R_{ij} is a Euclidean d-ball with ij as diameter. Thus R_{ij} approaches a compact sphere-like region on our unstructured graphs.

Figure 5.6 and Figure 5.7 plot the *normalized mean fragmentation* as a function of the diameter $D(i,j)$. For the unstructured mesh plots, we normalized the fragmentation by $D(i,j)$ in 2D (for the bunny) and by $D(i,j)^2$ in 3D (for torso and steven). For computational tractability, we sampled one million random (i,j) pairs to avoid having to consider the fragmentation over all possible node pairs. Evidently μ_0 is also a good predictor of fragmentation, and the fragmentation curves largely follow the same order as the perimeter curves. In particular, our MINEP layouts consistently yield the lowest fragmentation.

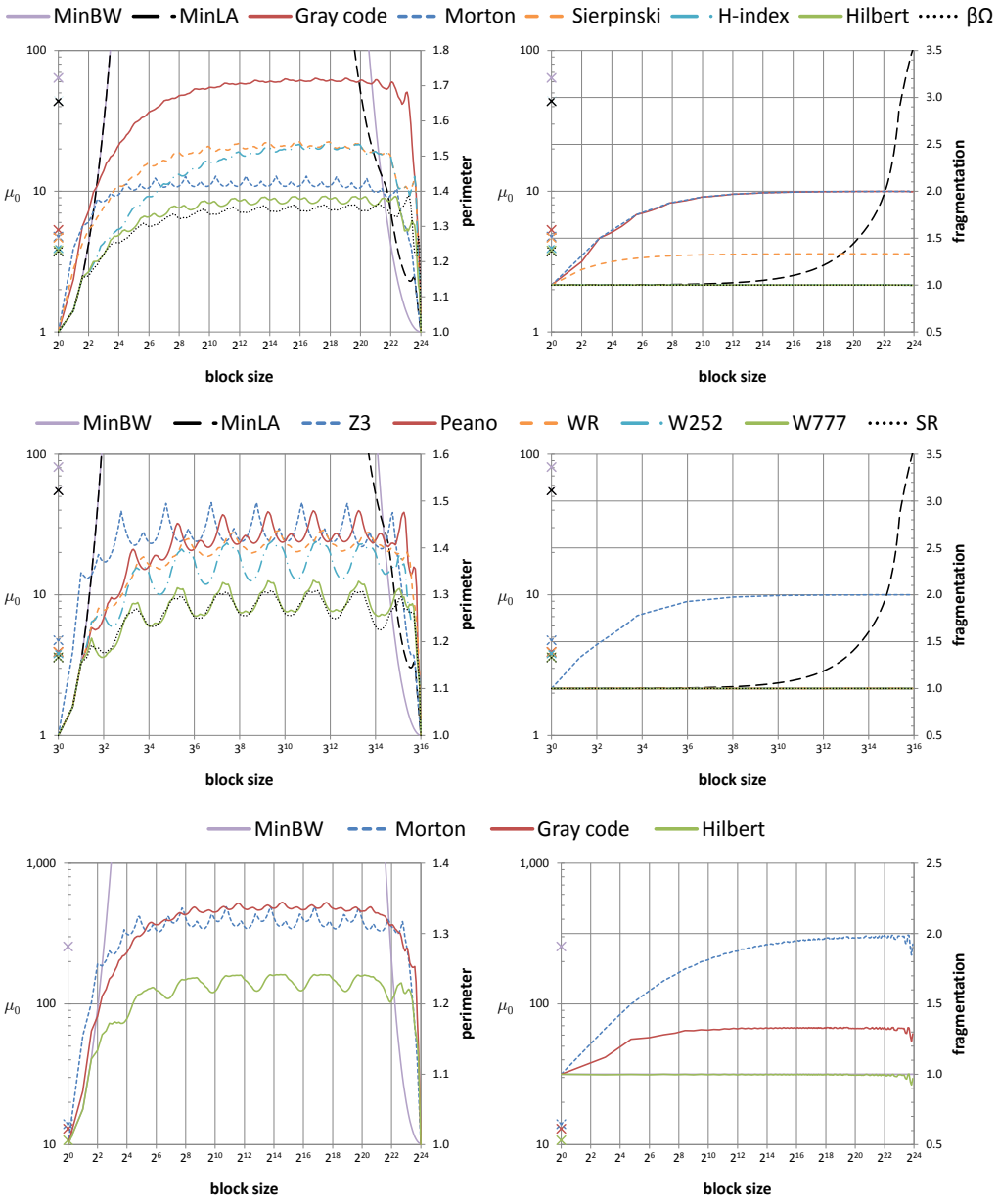


FIG. 5.6. *Normalized perimeter (left) and fragmentation (right) for (from top to bottom) grids of size* $2^{12}\times 2^{12}$, $3^8 \times 3^8$, *and* $2^8 \times 2^8 \times 2^8$. μ_0 *is shown as crosses on the left axes. Many fragmentation curves overlap at* $f = 1$.

For Hamiltonian path orderings of $n \times n$ grids, the number of fragments in an $m \times m$ region S equals half the number of path edges cut by S, because each fragment must enter and exit S exactly once [5]. As a consequence, assuming the path has an equal number of uniformly distributed horizontal and vertical edges, by counting the number of edges cut by all $(n-m+1)^2$ translations of the four sides of S, one can show that the expected normalized fragmentation is approximately $f \simeq 1 + \frac{1}{n} - \frac{1}{n-m+1} \simeq 1$ for large n. This explains the overlapping curves (including MINBW) with unit fragmentation in Figure 5.6.

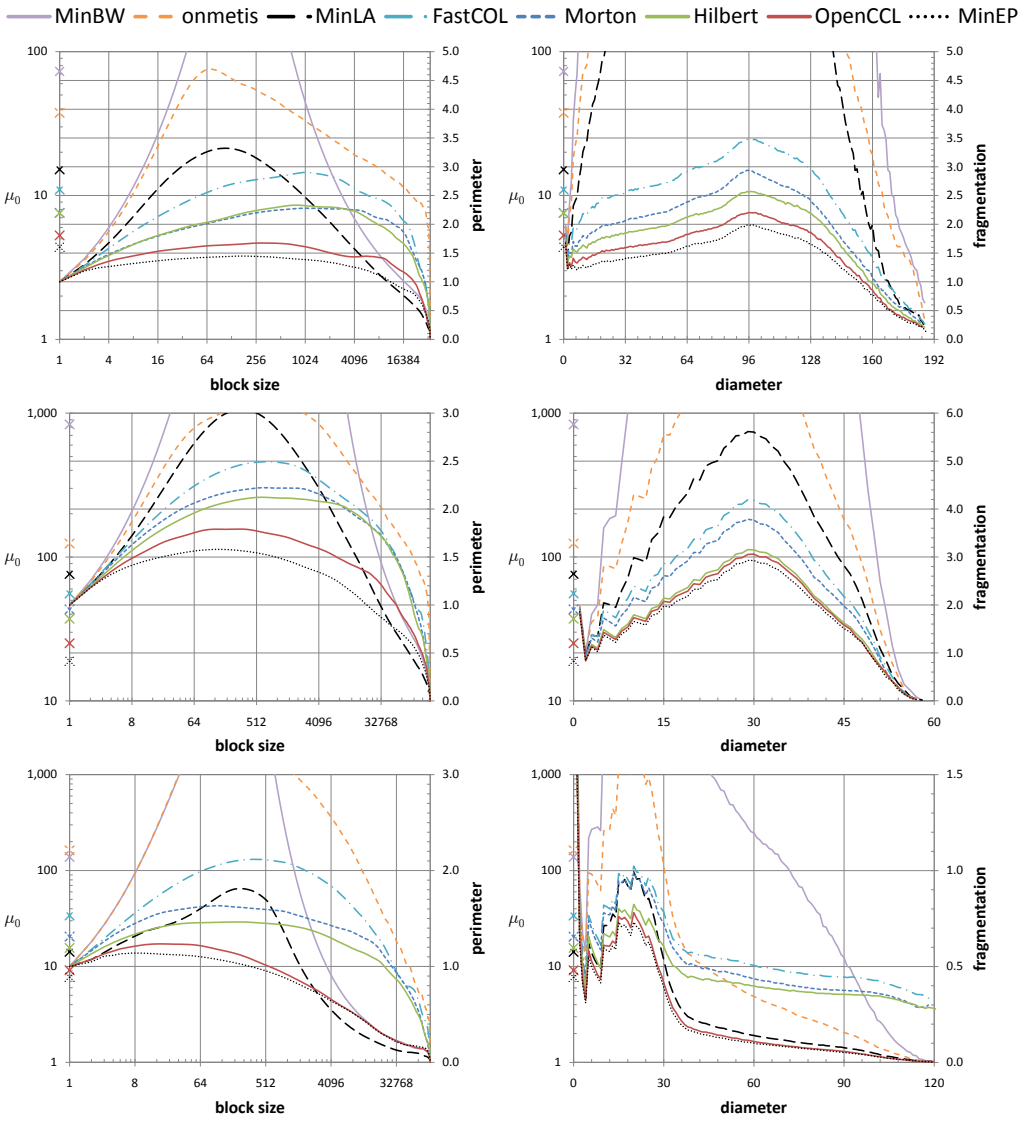


FIG. 5.7. *Normalized perimeter (left) and fragmentation (right) for (from top to bottom) the bunny, torso, and steven mesh. The μ_0 locality measure is shown as crosses on the left axes.*

5.4. Graph partitioning. We have hinted above at the use of MINEP layouts to partition graphs, e.g. by dividing the layout into k equal-sized intervals. Such *implicit graph partitioning*, which finds applications in parallel numerical simulation, has previously been explored using space-filling layouts such as Hilbert and Morton's curves [9, 23, 60]. One downside of such geometric partitioning is that nodes from different components or from different materials that are in close geometric proximity may inadvertently be grouped together. Furthermore, when applied to surfaces embedded in higher dimensions, geometrically close but geodesically distant parts of the mesh may be grouped. Using our algebraic technique, node affinities can be explicitly modeled and respected during partitioning, resulting in higher-quality partitions.

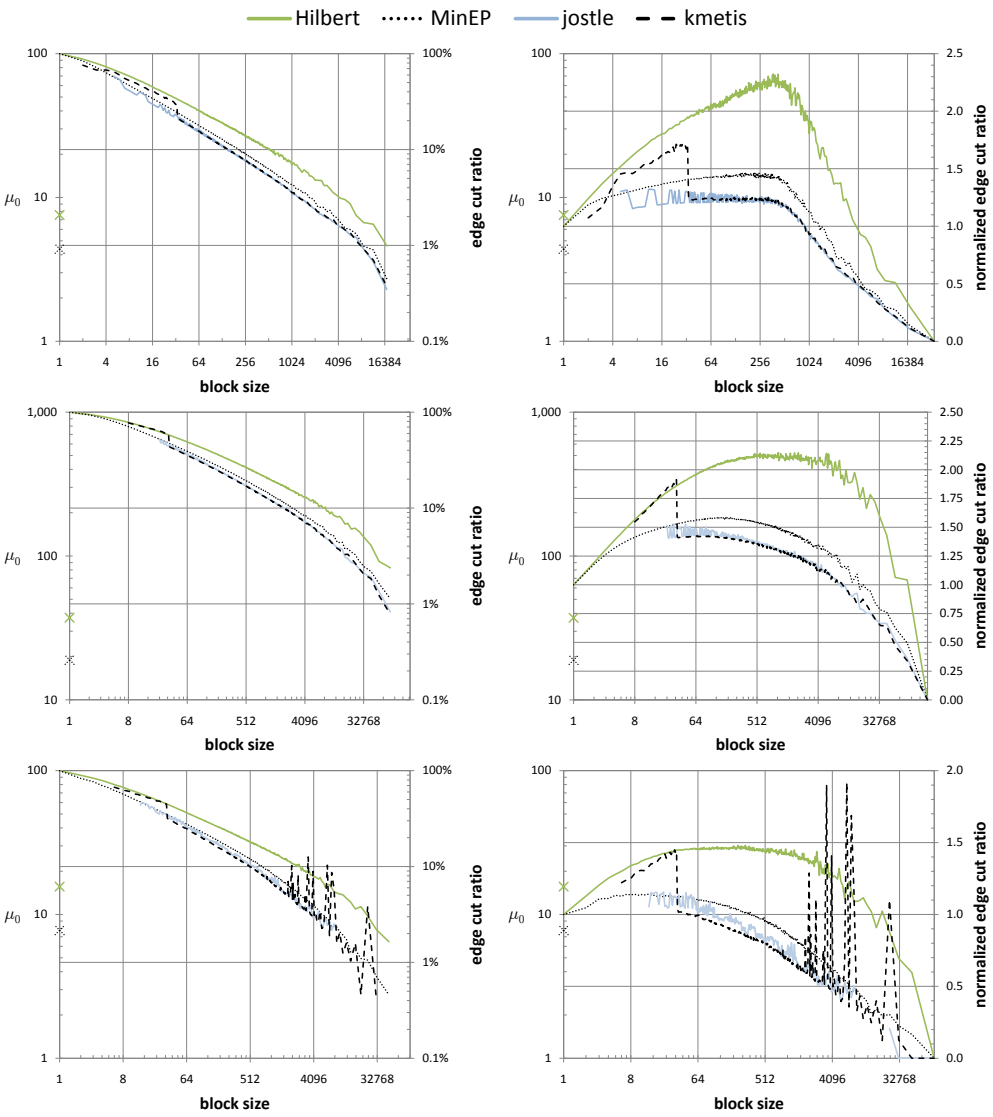


FIG. 5.8. *Graph partitioning. Log-log (left) and normalized (right) plots of edge cut c for (from top to bottom) the primal graphs of the bunny, torso, and steven meshes. The graph partitioners* `jostle` *and* `kmetis` *optimize the partitions for each block size n, whereas Hilbert and* MINEP *partition a single, fixed layout by grouping n consecutive vertices. Given $c(n)$ cut edges, we measure the normalized edge cut as $\hat{c}(n) = n^{1/d}c(n)/|E|$.*

Graph partitioning is usually formulated as a cut minimization problem: group the vertices of the graph into k roughly equal-sized subsets while minimizing the number of edges whose vertices belong to different groups. Because our MINEP layouts attempt to minimize the edge cut at multiple scales, we would expect them to result in good partitions. To test this hypothesis, we partitioned the primal graphs of our benchmark meshes using our layouts, a 3D Hilbert curve, and the state-of-the-art graph partitioners Metis [27] and Jostle [53]. For MINEP and Hilbert, we computed a single layout and then partitioned the graphs into perfectly balanced pieces (modulo rounding) of size $n = |V|/k$. Using the dedicated graph partitioners, we optimized the partitions for each choice of k, which over all values of k required far more computation than computing our layouts.

The resulting edge cuts are plotted in Figure 5.8. These plots show two remarkable results: our MINEP layouts result in partitions that are only a few percent worse than those produced by the graph partitioners. In fact, when many partitions (small blocks) are requested, and for larger subdomains of the steven mesh, MINEP often outperforms `kmetis`. Second, our layouts are far superior to the Hilbert curve for this task. A key advantage of our layouts is that they need be computed only once, after which "instant" partitioning can be achieved for any desired number of subdomains k. Moreover, these subdomains automatically exhibit good locality, which may be of benefit to high-performance computing applications.

5.5. Cache utilization. One common application of graph reordering is the acceleration of linear algebra routines such as sparse matrix-vector multiplication ($SpM \times V$), where the goal is to permute the rows and columns of a sparse matrix to improve the locality of reference to the (often dense) vector and reduce the number of cache misses. $SpM \times V$ is usually implemented using *compressed row storage* (CRS), which stores the sparse matrix as three linear arrays [39]. A matrix-vector multiplication $y = Ax$ is then implemented as follows:

```
1.  for i = 1, ..., |V|
2.      sum = 0
3.      for k = row[i], ..., row[i + 1] − 1
4.          sum = sum + value[k] ∗ x[index[k]]
5.      y[i] = sum
```

This kernel makes some number of compulsory cache misses in the *sequential access* to the matrix coefficients stored in the `value` array and to the column `index` array. We here focus on the number of noncompulsory cache misses due to *random access* to x, as only those are affected by the matrix layout. Our approach is to model the sparse matrix as a graph, with each row and column representing a vertex and each nonzero matrix entry representing an edge, and to order the graph using the strategies already discussed above. A good ordering exploits both spatial coherence (nearby elements in x are accessed together) and temporal coherence (the same element of x is accessed repeatedly over a short period of time) [21].

In our evaluation, we used Laplacian smoothing as the primary application. Each row i of the matrix A has nonzeros in column i and in the columns corresponding to the immediate neighbors of i in the mesh. We used `valgrind`'s `cachegrind` command [50] to simulate a two-level cache and to count noncompulsory cache misses. Figure 5.3 illustrates the number of L1 and L2 cache misses incurred on our unstructured meshes, using an 8 KB L1 cache and a 512 KB L2 cache, both with a line size of 64 bytes (i.e. eight double-precision vector elements). We note that our MINEP layouts result in far fewer L1 cache misses than MINBW, in spite of these bandwidth-reducing layouts being a popular choice for matrix reordering. (Experiments using *reverse Cuthill-McKee* layouts [8] produced similar results.) We attribute this to our layouts' preference for many short edges at the expense of a few long edges. Although the MINBW layout keeps the maximum edge length low, most edges exceed the line size and hence incur more cache misses. The L2 cache is large enough to hold a substantial fraction of x, and here MINBW performs better.

5.6. Temporal locality and triangle mesh rendering. We have so far focused primarily on *spatial locality*, and have paid little attention to a layout's effect on *temporal locality*. Perhaps one of the most ubiquitous uses of temporal locality is the post transform and lighting vertex cache used by off-the-shelf graphics cards (see, e.g., [47]). This cache, which for simplicity is usually implemented as a FIFO queue, holds a few dozen vertices that have undergone coordinate transformations. Each time a triangle is rendered, its three vertices are looked up in the cache and are if present reused, thereby avoiding multiple expensive transformations of each vertex. Because each vertex is incident on six triangles on average, a

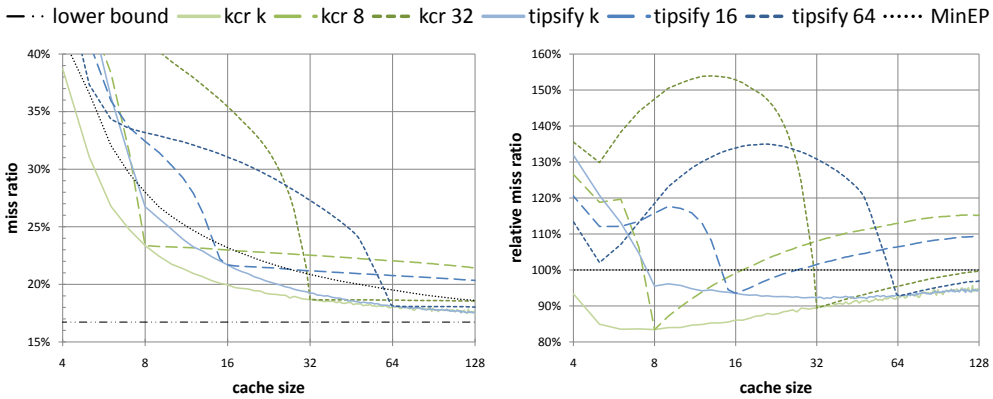


FIG. 5.9. *Cache miss ratio for triangle mesh rendering using a FIFO vertex cache. The* kcr k *and* tipsify k *curves show the best performance obtainable by tailoring the layout to the cache size k, whereas the other curves correspond to the performance of individual layouts optimized for a single cache size (e.g. kcr 8 has beeen optimized for a cache size of 8 vertices). The plot on the right shows cache misses relative to our* MINEP *layout.*

good triangle layout that exploits temporal locality may limit the number of cache misses to the compulsory minimum of $1/6$.

Several cache-aware ordering algorithms have emerged for graphics and related applications, including [2, 7, 11, 22, 24, 31, 47]. Some of these explicitly model the cache size and replacement policy, and produce layouts specifically tailored for a particular cache. By ordering the dual graph using our framework, we may produce a spatially coherent triangle sequence that indirectly exhibits good temporal locality as well, in the sense that adjacent triangles that appear nearby in the layout share and thus reference many of the same vertices.

Figure 5.9 illustrates the cache utilization of our *cache-oblivious* MINEP layouts and several *cache-aware* layouts produced by *k-cache-reorder* [31] and *tipsify* [47] for a simulated FIFO vertex cache. Although not as effective as the best cache-aware layouts, our layouts perform well on average across many different cache sizes. On the contrary, the cache-aware layouts perform well only in a narrow range of cache sizes, and, as Figure 5.9 shows, may yield over 50% more cache misses than MINEP when used with a different-size cache than the one optimized for. Note in particular the sharp knees in the curves for the cache-aware layouts. By contrast, the best cache-aware layout yields only 16% fewer cache misses than our MINEP layout. Because cache sizes differ among graphics cards, and because computers employ a whole hierarchy of caches of varying characteristics, the most reliable performance is obtained using a cache-oblivious data layout such as ours.

6. Conclusions. We have presented a detailed study of the minimum edge product linear ordering problem, which amounts to finding an optimal permutation of the nodes of an undirected graph. We have shown how the associated locality measure μ_0 and its induced graph layouts may have utility in a large number of applications that put a premium on data locality, from databases to graph partitioning to linear algebra and computer graphics. We conclude with the following summary of our findings:

- Our μ_0 locality measure is a generalized p-mean that complements the set of well-known p-mean ordering functionals that have already been in use for decades, including $p \in \{1, 2, \infty\}$.
- μ_0 is an algebraic measure defined on an affinity graph specified by the user. As such, it requires no underlying geometric structure or embedding, yet allows for the possibility of using geometric proximity as one of many affinity criteria.

- Unlike prior locality measures with complexity $O(|V|^2)$ [15, 18], μ_0 can be evaluated in time $O(|E|)$, and allows for localized updates when only a few nodes are permuted. Moreover, μ_0 correlates well with other locality measures and is a good predictor of layout quality, e.g. in terms of perimeter, fragmentation, and cache use.
- Cartesian grid layouts that minimize μ_0 have a space-filling, fractal character. In general, μ_0 favors layouts that are Hamiltonian and hierarchical.
- We showed that $\mu_0 < 8$ for *any* hierarchical layout of a dyadic grid, whereas μ_0 for a lexicographic ordering is unbounded as the grid size tends to infinity.
- μ_0 favors space-filling layouts known to have good locality properties. In particular, it scores the β-Ω curve (a close cousin to the Hilbert curve) as the best dyadic layout.
- Based on empirical studies of μ_0-optimal grid layouts, we developed the *SR space-filling curve* for triadic grids and showed that it has other locality properties that improve on previously known triadic space-filling curves.
- Unlike many competing layouts based on constructive recipes [2, 7, 11, 19, 22, 27, 31, 47, 49, 52, 58], our MINEP layouts arise from the optimization of an ordering functional that admits a variety of optimization strategies. We presented one effective ordering strategy inspired by algebraic multigrid (AMG), and showed how our layouts improve in quality on prior optimization approaches [57, 58].
- We presented an in-depth study of the performance of many different layouts with respect to several different quality measures and applications. We showed that, in general, our MINEP layouts are superior to the other layouts, both in terms of graph-theoretic measures and in practical applications.
- Some of the prime applications of our layouts include matrix reordering, graph partitioning, efficient mesh rendering, and data organization for spatial queries.

Although our treatment of the problem has been fairly extensive, we see several avenues for continued work in this area. First, our multilevel layout algorithm is quite slow. For more widespread adoption, a more efficient approach is needed that can order millions of nodes within a few seconds. Furthermore, we believe that the layouts it produces can be improved in quality by paying closer attention to the coarsening strategy, which largely dictates the final hierarchical grouping of nodes. Some problems are best expressed not using a graph structure, but rather as a *hypergraph*. How should one extend the notion of edge products to hypergraphs, and how does this influence the ordering strategy?

On the theoretical side, there is considerable interest in finding provably optimal layouts, at least for certain classes of graphs, such as dyadic 2D and 3D Cartesian grids. Are β-Ω curves optimal for all grid sizes? What is the optimal dyadic 3D grid? The answer to this question may be of even greater consequence, considering the much larger size of volumetric grids and the implied increase in data movement. Optimal layouts of other classes of graphs, such as hexagonal grids and binary trees, are also of interest.

We have showed balanced graph partitioning as one potential application. In practice, some imbalance is tolerable and even desired. How should one optimize the 1D partitions under some imbalance constraint? How can dynamic load balancing be achieved effectively using implicit partitioning? Finally, our layouts may find utility in dimensionality reduction. In particular, may μ_0 be used to reduce high-dimensional spaces to dimensions other than one, e.g. for locality-preserving graph layout in two dimensions?

Acknowledgements. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. We wish to thank Marc Tchiboukdjian for providing the FASTCOL layouts, Pedro Sander for sharing his tipsify code, Gang Lin and Thomas Yu for making their k-cache-reorder software available, and Ilya Safro for unstructured MINBW and MINLA layouts.

References.

[1] D. ADOLPHSON AND T. C. HU, *Optimal linear ordering*, SIAM Journal on Applied Mathematics, 25 (1973), pp. 403–423.

[2] J. J. BARTHOLDI III AND P. GOLDSMAN, *Multiresolution indexing of triangulated irregular networks*, IEEE Transactions on Visualization and Computer Graphics, 10 (2004), pp. 484–495.

[3] A. BOGOMJAKOV AND C. GOTSMAN, *Universal rendering sequences for transparent vertex caching of progressive meshes*, Computer Graphics Forum, 21 (2002), pp. 137–148.

[4] C. BÖHM, S. BERCHTOLD, AND D. A. KEIM, *Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases*, ACM Computing Surveys, 33 (2001), pp. 322–373.

[5] E. BUGNION, T. ROOS, R. WATTENHOFER, AND P. WIDMAYER, *Space filling curves versus random walks*, Lecture Notes in Computer Science, 1340 (1997), pp. 199–211.

[6] S. CHATTERJEE, A. R. LEBECK, P. K. PATNALA, AND M. THOTTETHODI, *Recursive array layouts and fast matrix multiplication*, IEEE Transactions on Parallel and Distributed Systems, 13 (2002), pp. 1105–1123.

[7] J. CHHUGANI AND S. KUMAR, *Geometry engine optimization: Cache friendly compressed representation of geometry*, in ACM Symposium on Interactive 3D Graphics and Games, 2007, pp. 9–16.

[8] E. CUTHILL AND J. MCKEE, *Reducing the bandwidth of sparse symmetric matrices*, in 24th National Conference of the ACM, 1969, pp. 157–172.

[9] K. DEVINE AND B. HENDRICKSON, *Tinkertoy parallel programming: A case study with Zoltan*, International Journal of Computational Science and Engineering, 1 (2005), pp. 64–72.

[10] J. DIAZ, J. PETIT, AND M. SERNA, *A survey of graph layout problems*, ACM Computing Surveys, 34 (2002), pp. 313–356.

[11] P. DIAZ-GUTIERREZ, A. BHUSHAN, M. GOPI, AND R. PAJAROLA, *Single strips for fast interactive rendering*, The Visual Computer, 22 (2006), pp. 372–386.

[12] C. FALOUTSOS, *Multiattribute hashing using Gray codes*, ACM SIGMOD Record, 15 (1986), pp. 227–238.

[13] A. GEORGE AND A. POTHEN, *An analysis of spectral envelope reduction via quadratic assignment problems*, SIAM Journal on Matrix Analysis and Applications, 18 (1997), pp. 706–732.

[14] N. E. GIBBS, W. G. POOLE JR., AND P. K. STOCKMEYER, *An algorithm for reducing the bandwidth and profile of a sparse matrix*, SIAM Journal on Numerical Analysis, 13 (1976), pp. 236–250.

[15] C. GOTSMAN AND M. LINDENBAUM, *On the metric properties of discrete space-filling curves*, IEEE Transactions on Image Processing, 5 (1996), pp. 794–797.

[16] L. H. HARPER, *Optimal assignments of numbers to vertices*, SIAM Journal on Applied Mathematics, 12 (1964), pp. 131–135.

[17] M. HASAN, S.-E. YOON, AND K.-Y. CHWA, *Bounds on the geometric mean of arc lengths for bounded-degree planar graphs*, Lecture Notes in Computer Science, 5598 (2009), pp. 153–162.

[18] H. HAVERKORT AND F. VAN WALDERVEEN, *Locality and bounding-box quality of two-dimensional space-filling curves*, Computational Geometry: Theory and Applications, 43 (2010), pp. 131–147.

[19] G. HEBER, R. BISWAS, AND G. R. GAO, *Self-avoiding walks over adaptive unstructured grids*, Concurrency: Practice and Experience, 12 (2000), pp. 85–109.

[20] B. HENDRICKSON AND R. LELAND, *An improved spectral graph partitioning algorithm for mapping parallel computations*, SIAM Journal on Scientific Computing, 16 (1995), pp. 452–269.
[21] D. B. HERAS, V. BLANCO, J. C. CABALEIRO, AND F. F. RIVERA, *Modeling and improving locality for the sparse-matrix-vector product on cache memories*, Future Generation Computer Systems, 18 (2001), pp. 55–67.
[22] H. HOPPE, *Optimization of mesh locality for transparent vertex caching*, in ACM SIGGRAPH, 1999, pp. 269–276.
[23] J. HUNGERSHÖFER AND J.-M. WIERUM, *On the quality of partitions based on space-filling curves*, Lecture Notes In Computer Science, 2331 (2002), pp. 36–45.
[24] M. ISENBURG AND P. LINDSTROM, *Streaming meshes*, in IEEE Visualization, 2005, pp. 231–238.
[25] G. JIN AND J. MELLOR-CRUMMEY, *SFCGen: A framework for efficient generation of multi-dimensional space-filling curves by recursion*, ACM Transactions on Mathematical Software, 31 (2005), pp. 120–148.
[26] M. JUVAN AND B. MOHAR, *Optimal linear labelings and eigenvalues of graphs*, Discrete Applied Mathematics, 36 (1992), pp. 153–168.
[27] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392.
[28] ——, *Metis: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, University of Minnesota, 4.0 ed., 1998.
[29] Y. KOREN AND D. HAREL, *A multi-scale algorithm for the linear arrangement problem*, Lecture Notes In Computer Science, 2573 (2002), pp. 296–309.
[30] ——, *One dimensional layout optimization, with applications to graph drawing by axis separation*, Computational Geometry: Theory and Applications, 32 (2005), pp. 115–138.
[31] G. LIN AND T. P.-Y. YU, *An improved vertex caching scheme for 3D mesh rendering*, IEEE Transactions on Visualization and Computer Graphics, 12 (2006), pp. 640–648.
[32] P. LINDSTROM AND V. PASCUCCI, *Terrain simplification simplified: A general framework for view-dependent out-of-core visualization*, IEEE Transactions on Visualization and Computer Graphics, 8 (2002), pp. 239–254.
[33] R. LIU, H. ZHANG, AND O. VAN KAICK, *Spectral sequencing based on graph distance*, in IEEE Geometric Modeling and Processing, 2006, pp. 632–638.
[34] G. MITCHISON AND R. DURBIN, *Optimal numberings of an $n \times n$ array*, SIAM Journal on Discrete and Algebraic Methods, 7 (1986), pp. 571–582.
[35] B. MOHAR AND S. POLJAK, *Eigenvalues in combinatorial optimization*, in Combinatorial and Graph-Theoretical Problems in Linear Algebra, Springer-Verlag, 1993, pp. 107–151.
[36] B. MOON, H. V. JAGADISH, C. FALOUTSOS, AND J. H. SALTZ, *Analysis of the clustering properties of the Hilbert space-filling curve*, IEEE Transactions on Knowledge and Data Engineering, 13 (2001), pp. 124–141.
[37] A. Y. NG, M. I. JORDAN, AND Y. WEISS, *On spectral clustering: Analysis and an algorithm*, in Fifteenth Annual Conference on Neural Information Processing Systems, 2001, pp. 849–856.
[38] R. NIEDERMEIER, K. REINHARDT, AND P. SANDERS, *Towards optimal locality in mesh-indexings*, Discrete Applied Mathematics, 117 (2002), pp. 211–237.
[39] A. PINAR AND M. T. HEATH, *Improving performance of sparse matrix-vector multiplication*, in ACM/IEEE Conference on Supercomputing, 1999, p. 30.

[40] A. POTHEN, H. D. SIMON, AND K.-P. LIOU, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 430–452.
[41] J. W. RUGE AND K. STÜBEN, *Algebraic multigrid*, in Multigrid Methods, S. F. McCormick, ed., SIAM, 1987, pp. 73–130.
[42] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, second ed., 2003.
[43] I. SAFRO, D. RON, AND A. BRANDT, *Graph minimum linear arrangement by multilevel weighted edge contractions*, Journal of Algorithms, 60 (2006), pp. 24–41.
[44] ——, *A multilevel algorithm for the minimum 2-sum problem*, Journal of Graph Algorithms and Applications, 10 (2006), pp. 237–258.
[45] ——, *Multilevel algorithms for linear ordering problems*, Journal of Experimental Algorithmics, 13 (2009), p. 4.
[46] H. SAGAN, *Space-Filling Curves*, Springer-Verlag, 1994.
[47] P. V. SANDER, D. NEHAB, AND J. BARCZAK, *Fast triangle reordering for vertex locality and reduced overdraw*, ACM Transactions on Graphics, 26 (2007), p. 89.
[48] S. SCHAMBERGER AND J.-M. WIERUM, *Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves*, Lecture Notes in Computer Science, 2763 (2003), pp. 165–179.
[49] ——, *A locality preserving graph ordering approach for implicit partitioning: Graph-filling curves*, in 17th International Conference on Parallel and Distributed Computing Systems, 2004, pp. 51–57.
[50] J. SEWARD, N. NETHERCOTE, AND J. WEIDENDORFER, *Valgrind 3.3—Advanced Debugging and Profiling for GNU/Linux Applications*, 2008.
[51] R. J. STEVENS, A. F. LEHAR, AND F. H. PRESTON, *Manipulation and presentation of multidimensional image data using the Peano scan*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 5 (1983), pp. 520–525.
[52] M. TCHIBOUKDJIAN, V. DANJEAN, AND B. RAFFIN, *Binary mesh partitioning for cache-efficient visualization*, IEEE Transactions on Visualization and Computer Graphics, 16 (2010), pp. 815–828.
[53] C. WALSHAW AND M. CROSS, *Mesh partitioning: A multilevel balancing and refinement algorithm*, SIAM Journal on Scientific Computing, 22 (2000), pp. 63–80.
[54] J.-M. WIERUM, *Definition of a new circular space-filling curve: $\beta\Omega$-indexing*, Tech. Rep. TR-001-02, Paderborn Center for Parallel Computing, 2002.
[55] ——, *Logarithmic path-length in space-filling curves*, in 14th Canadian Conference on Computational Geometry, 2002, pp. 22–26.
[56] W. WUNDERLICH, *Über Peano kurven*, Elemente der Mathematik, 28 (1973), pp. 1–10.
[57] S.-E. YOON AND P. LINDSTROM, *Mesh layouts for block-based caches*, IEEE Transactions on Visualization and Computer Graphics, 12 (2006), pp. 1213–1220.
[58] S.-E. YOON, P. LINDSTROM, V. PASCUCCI, AND D. MANOCHA, *Cache-oblivious mesh layouts*, ACM Transactions on Graphics, 24 (2005), pp. 886–893.
[59] ——, *OpenCCL: Cache-coherent layouts of meshes and graphs*, 2006. http://gamma.cs.unc.edu/COL/OpenCCL/.
[60] G. ZUMBUSCH, *On the quality of space-filling curve induced partitions*, Zeitschrift für Angewandte Mathematik und Mechanik, 81 (2001), pp. 25–28.