Approximating average and worst-case quality measure values for d-dimensional space-filling curves

Sasburg, S.B.

Award date:
2011

technische universiteit eindhoven
Department of Mathematics and Computer Science

Master's Thesis

Approximating average and worst-case quality measure values for d-dimensional space-filling curves

by
Simon Sasburg

Supervisor
Herman Haverkort

Eindhoven, October 19, 2011

Abstract

Space-filling curves can be used in a variety of algorithms and data-structures. In order to help decide which space-filling curve to use, several quality measures exist that express the effectiveness of a space-filling curve for a particular purpose. We improve and extend an algorithm that approximates some worst-case locality and bounding-box quality measures for space-filling curves to make it faster and use less memory. This makes it possible to run the algorithm on space-filling curves in higher dimensions. For this we create generalizations of some known 2D space-filling curves for any number of dimensions. In addition to the locality and bounding-box quality measures we present and implement a method to calculate a perimeter-based quality measure. Other improvements will be made to be able to handle more complex types of space-filling curves. Another algorithm will be introduced to approximate average values for each of the measures used by the worst-case approximation algorithm. Using these algorithms we will evaluate existing and new space-filling curves in relation to each other.

Contents

Chapter 1

Introduction

1.1 Space-filling curves

A space-filling curve is a curve which covers the entire d-dimensional space. To be more precise, a space-filling curve defines a mapping from $\mathbb{R}$ to $\mathbb{R}^d$, and this mapping is continuous and surjective.

The first space-filling curve was discovered by Giuseppe Peano when he was trying to construct a continuous mapping from the unit interval onto the unit square. He discovered the existence of such a curve for $d = 2$ and $d = 3$ in 1890 [10]. Peano's paper contained no graphical representation of his curve or its construction, instead choosing to define the curve entirely with mathematical formula's. David Hilbert later published a variation on Peano's work [7], and defined a new space-filling curve (See Figure 1.1) which we will use as an example here.

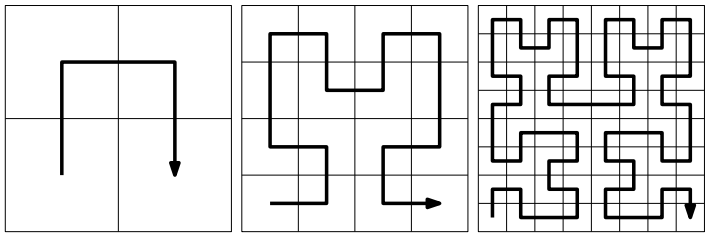


Figure 1.1: First three expansions of the Hilbert curve.

In this figure three expansion steps are shown for the Hilbert curve. The first step (the leftmost image) shows how a unit square is divided into four sections, and the directed curve through these sections represents the order in which the points in these sections appear on the space-filling curve. The order of points inside each section is determined by applying this first step recursively to that section, possibly with some transformations. In the middle image this has been done for each of the subsections to obtain an ordering within each of the sections of the leftmost image. The rightmost image repeats this expansion step. Each iteration of this process makes the directed curve a more precise approximation of its limit, in which the curve fills the entire space.

Space-filling curves can be used to order a set of points P in a d-dimensional space. However, because the mapping defined by the curve is not injective, some care needs to be taken to correctly handle those points which occur on the curve more than once. For example, the point at the center of the unit square would appear in all four sections, and other points on section perimeters can appear in two sections. Such points can easily be handled by only considering their first occurrence on the curve. Doing so yields a surjective mapping from $\mathbb{R}^d$

to $\mathbb{R}$ corresponding to the space-filling curve, which can be used to order P.

1.2 Applications

1.2.1 Bounding-box hierarchies

Bounding-box hierarchies are a type of data structure in which, for example, points are stored, and on which queries can be performed to retrieve these points. These points are stored grouping them into blocks. Each point is stored in one block and each block has at most B points, for some parameter B. For each block a minimal bounding box can be found which contains all the points in that group. These bounding boxes are stored in an index structure. This index structure can use a similar approach to group these bounding boxes into blocks with even bigger bounding boxes stored in an other index structure. This can be repeated to create a hierarchy with any number of levels. For our discussion here we only consider a single level, but similar reasoning holds for each level in such a hierarchy.

Queries on a bounding-box hierarchy, such as finding the point r with the lowest distance to a given point p, can then limit the number of points they need to look at by only looking at blocks for which the bounding box indicates that the block could contain points relevant to the query. To find the closest point to p, you could start looking in the block with the bounding box closest to p. And when you have found a candidate point to return, you can ignore all blocks for which the bounding boxes indicate they can never contain a point that is closer to p than the candidate point.

For large datasets which do not fit in main memory, the performance of such queries then mainly depends on the number of blocks that need to be checked, because the index can be cached in main memory while the blocks need to be loaded from slower external memory (like a hard disk).

R-Trees [12] are an example of a bounding-box hierarchy. In an R-Tree each leaf node makes up a block, and the non-leaf nodes will make up an index structure containing bounding boxes. One way to divide the points into blocks is by ordering the points with a space-filling curve and putting the first B points into the first block, the next B points into the second block, and so on.

Query performance is based on the number of blocks that need to be checked, and the number of blocks that need to be checked is at most the number of blocks for which the bounding box intersects with the circle with p as the center point and a radius equal to the distance between p and r. Therefore, the ordering given by the space-filling curve used should map points close to each other into the same block, resulting in smaller bounding boxes, which reduces the likelihood of each bounding box intersecting with the circle.

1.2.2 Partial differential equations

Partial Differential Equations (PDE's [4]) are a method to formulate problems involving functions of multiple variables. For example, they can be used to model systems describing the propagation of sound or heat, or fluid dynamics. Typically such systems are simulated by dividing the space into a grid of cells, storing a set of interesting values for each cell. Passage of time is simulated by time steps; in each step new values are calculated for all cells. The new values of a cell will only depend on their own old values and on the old values of their neighbors. This calculation can be parallelized among multiple machines (or processors/threads) by assigning each machine a group of cells for which it will calculate (and store) the new values. In such a setup, communication can become a bottleneck. To prevent that from happening we would like to minimize the amount of communication needed each time step. The amount of

communication that is needed for a machine M corresponds the number of cells assigned to another machine N ($N \neq M$) that neighbor a cell on M.

When using a space-filling curve to divide the cells among machines, this roughly corresponds to the perimeter (for $d = 2$) or surface (for $d = 3$) of the region filled by a curve segment.

Chapter 2

How to describe a space-filling curve

Space-filling curves can be described in many different ways. Peano used a purely algebraic approach in his paper [10]. Another approach is to describe a recursive refinement procedure as we have seen previously for Hilbert's curve (Figure 2.1). Such a refinement consists of a number of rules where each rule describes a refinement step. Hilbert's curve only has one rule as it has only one type of refinement step. For each rule we need to know the entry and exit points of the curve in the area of the refinement. Each rule divides an area into an ordered sequence of sections, where the ordering within each section will be determined by the (possibly recursive) application of a rule. So for each section we need to know which rule it will use, and how it is transformed. Each section can be rotated, mirrored, reversed, and scaled. For each curve we also define a unit region of area 1 and indicate what rule should be used to refine it (as a convention this will be the first rule in the curve definition). Often this unit region is just the unit square and the rule normally used by curve is defined on squares, and thus can be used to refine that unit region. Some curve examples can be seen in Figures 1.1, 2.2 and 2.3.

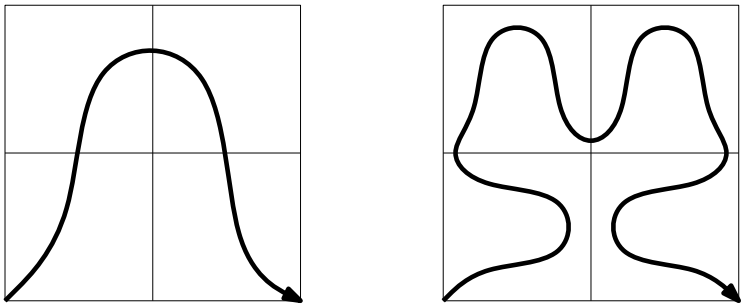


Figure 2.1: Hilbert curve

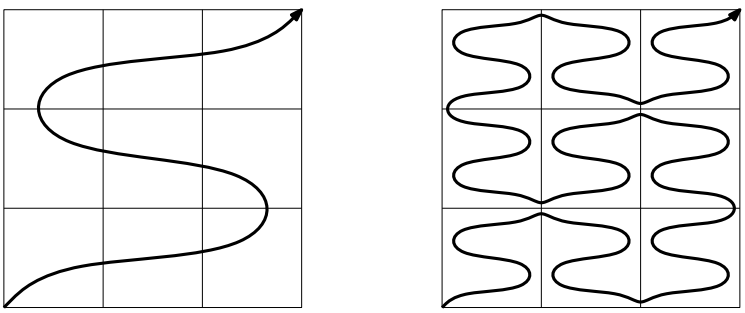


Figure 2.2: Peano curve

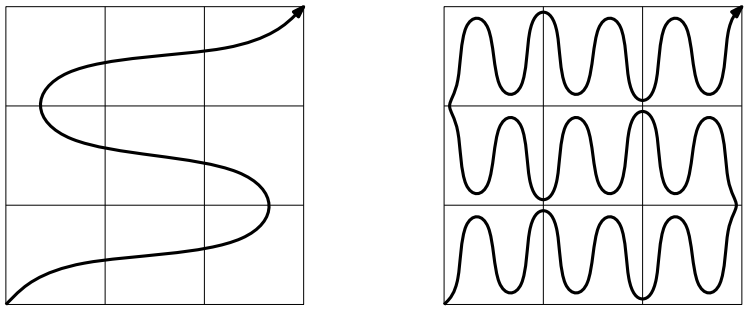


Figure 2.3: Coil curve

Such a set of rules define an *scanning order* $\prec$ of the points in the plane. We assume the set of points to be ordered is first scaled to fit within the unit region for simplicity, but it is also possible to extend the ordering in the unit region to the full plane.

The ordering of two points p and q is defined as follows by looking in which subsection of the starting rule the points p and q lie:

- $p \prec q$, if p lies in a subsection before the subsection q lies in
- $p \prec q$, if p and q lie in the same subsection and $p \prec q$ holds according to the rule for that subsection

Rule subsections have some overlap at their boundaries so we resolve this by assuming a point only lies in the first subsection that contains it.

Chapter 3

Quality measures of space-filling curves

Several different quality measures can be considered for space-filling curves, we will consider the locality and bounding-box measures as described in [6], as well as a measure relating the size of the boundary of a curve segment to the size of that curve segment previously investigated in [8].

3.1 Notation

First we will introduce some notation in order to be able to analyze quality measures and space-filling curves in detail. Here we will assume that each rule defining a space-filling curve contains the same number of subregions.

For generality we will use *volume* as a dimension-neutral term for area (2D) and volume (3D), and we will use *boundary* as a dimension-neutral term for perimeter (2D) and surface (3D).

A space-filling curve has a number of rules we will number from 0 to $m-1$, where m is the number of rules. Each rule (numbered r) describes how to divide a unit region of volume 1 into n subregions, and we will write the unit region for the rule as $U(r)$. For each subregion (numbered i) the rule defines a transformation $\tau(r,i)$ and the number of the rule to apply to that subregion $R(r,i)$. Together these define the scanning order inside that subregion.

We will use base-n numbers as a way to identify subregions of unit regions or subregions. For such a number a, we will use a' to denote its first digit and a'' to denote the rest of the digits. For each rule r, we will define $S(r,a)$ as the region of that rule identified by a, as follows: $S(r,a) = \tau(r,a')(S(R(r,a'),a''))$, with $S(r,\emptyset) = U(r)$. For example $S(0,538)$ is subregion 8 of subregion 3 of subregion 5 of rule 0.

We will use $|A|$ as the volume of a region A. The sum of the volumes of the subregions of a rule will be equal to the volume of the unit region of that rule, which is 1: $\sum_{0 \le i < n} |S(r,i)| = |U(r)| = 1$.

N_k is the set of k-digit base-n numbers, and we write $a \prec b$ if $a < b$ and a and b have the same number of digits in base-n notation, and $a \preceq b$ if $a \prec b$ or $a = b$.

When talking about subregions of one of the rules of the curve, $C(r, \prec b)$ will denote the union of all predecessors of subregion b in rule r. More precisely: $C(r, \prec b) := \bigcup_{i \prec b} S(r,i)$. Using this we can define some related operators like this: $C(r, \preceq b) := C(r, \prec b) \cup S(r,b)$, and $C(r, \succ a) := S(r,\emptyset) \setminus C(r, \preceq a)$, and $C(r, \succeq a) := S(r,\emptyset) \setminus C(r, \prec a)$, and $C(r,a,b) := C(r, \prec b) \setminus C(r, \prec a)$.

When we only care about subregions of the entire curve, we can omit the rule number r. In this case we assume rule 0 is used as this rule defines the subregions for the unit region. This leads to these shorthands: $C(\prec a) := C(0, \prec a)$, and $C(\preceq a) := C(0, \preceq a)$, and $C(\succ a) := C(0, \succ a)$, and $C(\succeq a) := C(0, \succeq a)$, and $C(a, b) := C(0, a, b)$. To identify a single subregion we will use $C(a)$ to denote $S(0, a)$.

We will also use $\tau(r, a)$ as a shorthand for $\tau(r, a') \circ \tau(R(r, a'), a'')$, and $\tau(a)$ as a shorthand for $\tau(0, a)$.

3.2 Locality measures

The pairwise locality measure we investigated belong to a class of locality measures originally defined by Gotsman and Lindenbaum [5]. However they only define it on regular grids while in [6] a more general definition of this class is given:

$$WL_r := \lim_{k\to\infty} \sup_{a,b\in N_k} \frac{d_r(C(a), C(b))^d}{|C(a, b)|}$$

where $d_r(A, B)$ is the L_r distance between the center of A and the center of B. In d dimensions, with A_i being the coordinate of the center of A in the i-th dimension, this distance is defined as follows: $d_r(S, T) = (\sum_{0\le i<d} |S_i - T_i|^r)^{1/r}$ for $r \in \mathbb{N}$, and $d_\infty(S, T) = \max_{0\le i<d} |S_i - S_t|$.

The measure is called WL_r for worst-case Locality and it gives an indication of how the distance along the curve and the distance in the d-dimensional space relate to each other.

Intuitively having points that are close to each other along the curve also be close together according the distance in the d-dimensional space will be beneficial when constructing bounding-box hierarchies because the bounding boxes will stay smaller.

3.3 Bounding-box measures

In addition to using locality measures, [6] also defines two bounding-box measures that directly measure the bounding box. One of these measures uses the bounding-box area (WBA) while the other uses the bounding-box perimeter (WBP).

In light of our dimension-neutral terms this we will name these measures *worst-case bounding-box volume* (WBV) and *worst-case bounding-box boundary* (WBB).

These measures are then defined as follows:

$$WBV := \lim_{k\to\infty} \sup_{a,b\in N_k} \frac{|bbox(C(a, b))|}{|C(a, b)|}$$

$$WBB := \lim_{k\to\infty} \sup_{a,b\in N_k} \frac{(boundary(bbox(C(a, b)))/2d)^{d/(d-1)}}{|C(a, b)|}$$

with $bbox(S)$ being the bounding box of S: the smallest axis-aligned hyperrectangle that contains S, and $boundary(S)$ being the size of the boundary of S, while d is the number of dimensions. The division of the size of the boundary by $2d$ makes the resulting value equal to 1 for the unit hypercube. To account for scaling we then take the power to $d/(d-1)$, so that $WBB = 1$ for hypercubes of any size.

3.4 Boundary ratio

For some applications, like the partial differential equations we saw in Section 1.2.2, we would like to have a measure relating the size of the boundary of a curve segment with its size.

While the bounding-box boundary measure gives some indication of this, it would be better to directly measure the boundary of a curve segment instead of the boundary measure of the bounding-box of that curve segment. For this we introduce the *worst-case boundary* (WB) measure:

$$WB := \lim_{k \to \infty} \sup_{a,b \in N_k} \frac{(boundary(C(a,b))/2D)^{d/(d-1)}}{|C(a,b)|}$$

Just like for the WBB measure we normalize the value here such that WB equals 1 for any hypercube.

Chapter 4

Approximating worst-case values of quality measures

Here we will describe a method to calculate worst-case approximations of the quality measures we discussed in Chapter 3. This approximation is achieved by devising lower and upper bounds on the worst-case quality measures, and improving these until the desired precision is reached.

We use a mapping μ which maps regions to real numbers such that it is invariant under all transformations $\tau(r, i)$ in the curve definition. The actual mapping will depend on the measure we want to approximate, for example $\mu(R)$ will be $|bbox(R)|/|R|$ for the bounding-box-area measure. We want an algorithm to approximate $\mu^* = \lim_{k\to\infty} \sup_{i\prec j\in N_k} \mu(C(i, j))$, the worst-case measure value over all possible subsegments of the curve.

4.1 Original algorithm

In [6] an algorithm is presented that approximates worst-case locality and bounding-box measures. This algorithm works by exploring *Probes*. A probe consists of three consecutive subsegments of a space-filling curve: a head part, a mid part, and a tail part, which are subregions of the curve. Such a probe represents all contiguous subsegments which start in the head part and end in the tail part. The mid part is the part of the curve all those contiguous subsegments have in common. A special probe which we call the master probe has an empty mid part and the head part is equal to the tail part. Such a probe represents all possible contiguous subsections of a curve. For any probe P, let $M(P)$ be the mid part, let $R_h(P)$ be the rule of the head region, and $R_t(P)$ the rule of the tail region. Then let $\alpha(P)$ be the transformation that transforms $U(R_h(P))$ into the head part of P, and let $\omega(P)$ be the transformation that transforms $U(R_t(P))$ into the tail part of P.

A subsegment $P(i, j)$ of P is the region $\alpha(P)(C(R_h(P), \succeq i)) \cup M(P) \cup \omega(P)(C(R_t(P), \preceq j))$. We will use $\mu^*(P)$ for the worst-case value of $\mu(S)$ over all subsegments S represented by probe P, so $\mu^*(P) = \lim_{k\to\infty} \sup_{i,j\in N_k} \mu(P(i, j))$.

By expanding the head and tail parts of a probe using the curve definition, choosing a subregion in the head part and a subregion in the tail part, we can *refine* such a probe to create a new probe which represents a subset of the contiguous subsections the original probe represents.

A refinement r_{ij} of a probe P $(i, j \in 0, ..., n-1)$ can be obtained by using $\alpha(P) \circ \tau(R_h(P), i)$ as the head transformation, $\omega(P) \circ \tau(R_t(P), j)$ as the tail transformation, and $\alpha(P)(C(R_h(P), \succ i)) \cup M(P) \cup \omega(P)(C(R_t(P), \prec j))$ as mid section.

A probe is in canonical form if $\alpha(P)$ is the identity transformation, and for any probe P we

can create its canonical form P' by setting $\alpha(P')$ to the identity transformation, setting $M(P')$ to $\alpha(P)^{-1}(M(P))$, and setting $\omega(P')$ to $\alpha(P)^{-1} \circ \omega(P)$.

The canonical children of a probe are the canonical forms of all the refinements of that probe.

The algorithm works starting with a queue containing only the master probe, and then repeatedly replacing probes with their refinements to obtain tighter lower and upper bounds on the measure μ^*. For each probe P we will make use of a lower bound $\mu^-(P)$ and an upper bound $\mu^+(P)$ which must satisfy $\mu^-(P) \leq \mu^*(P) \leq \mu^+(P)$. (We will so how to calculate these in chapter 5)

We keep track of the highest lower bound of the probes we encounter, as this is the lower bound on μ^*. The highest upper bound on the probes we have not expanded yet is the upper bound on μ^*. When the difference between the lower and upper bound is lower than the desired precision, we are done.

Algorithm ComputeWorstCaseCurveQuality()
1. $Q \leftarrow$ an empty first-in-first-out queue
2. $R \leftarrow$ an empty dictionary
3. Insert the master probe into Q and R
4. $lowerbound \leftarrow \max_{P \in Q} \mu^-(P)$
5. **while** $\max_{P \in Q} \mu^+(P) - lowerbound >$ desired precision **do**
6. $P \leftarrow$ extract probe from the head of Q
7. **for each** $P_{ij} \in$ Canonical children of P **do**
8. **if** $\mu^+(P_{ij}) \geq lowerbound$ **and not** $P_{ij} \in R$ **then**
9. Insert P_{ij} into Q and R
10. $lowerbound \leftarrow \max(lowerbound, \mu^-(P_{ij}))$
11. Report μ^* lies between $lowerbound$ and $\max_{P \in Q} \mu^+(P)$

For probes with an empty mid part we have $\mu^+ = \infty$. We call such probes *degenerate* probes. The algorithm is only useful if only finitely many degenerate probes are added to the queue to be expanded. For most curves the degenerate probes have tail transformation with scale factor 1, and with only reflections and 90 degree rotations this means there is a finite number of different degenerate probes. So because we only add each probe to the queue once, only a finite number of degenerate probes will be added the the queue, and they will quickly be expanded and removed.

4.2 Heuristic for better performance

The original algorithm simply expanded probes in the same order they were encountered, allowing the use of a simple FIFO queue.

By using a priority queue instead, we can expand the probe with the highest upper bound still in the queue. This leads to significant improvements on both the running time and the memory usage of the algorithm.

For probes which have an upper bound on the worst-case value which is higher than the actual worst-case value of the curve, if the difference is more than the desired precision, that probe must be expanded before the algorithm can return. This gives an intuition about why highest upper bound first is a good heuristic, as it starts with the probes that we are sure must be expanded. In practice this also seems to result in the fastest tightening of the lower bounds.

4.3 Symbolic math

4.3.1 Balanced Peano

A variation on the Peano curve is the balanced Peano curve which is introduced in [6]. The balanced Peano is the same as the regular Peano curve, except the height is scaled by *sqrt*3 (See figure 4.1).

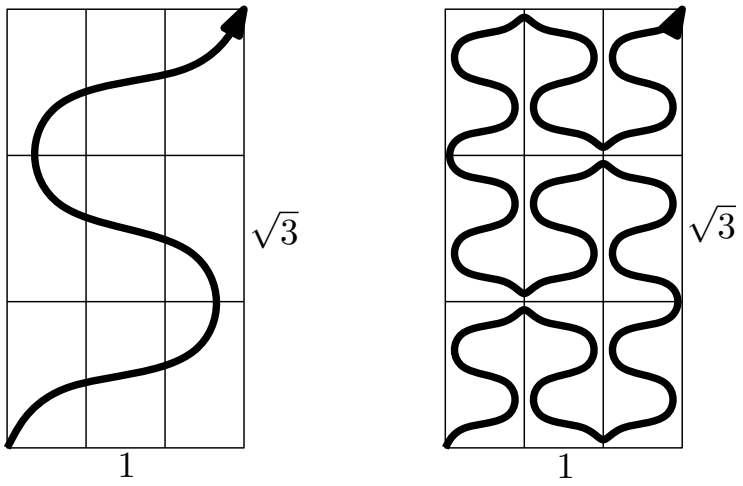


Figure 4.1: Balanced Peano curve

For this balanced Peano curve we need scale ratios which are not real numbers. While it is possible to approximate the scale ratios with real numbers, this will cause some imprecision which we would like to avoid.

We have solved this by using the symbolic math library GiNaC [1] for all mathematical operations in the algorithm. Instead of working with numbers it works with expressions, and operates directly on these. So for example $\sqrt{3}$ is an expression that is not further evaluated because that is impossible without losing precision. Some simplifications are performed so for example $\sqrt{4}$ would be simplified to 2.

One complication here is the need for comparison between such expressions by the algorithm to compare the bounds, and for the highest-upper-bound-first heuristic. In general it is not always possible to determine for two expressions which one of them is greater or if they are equal. For two expressions a and b we implement comparison $a < b$ by letting GiNaC numerically evaluate the expression $a - b$ and checking if the resulting value is smaller than 0. The numeric evaluation does cause some imprecision here, but because we just use the resulting value once and then throw it away, this imprecision is not propagated to the rest of the algorithm.

By using these expressions in the algorithm we can allow them in our curve definition too, such as the $\sqrt{3}$ scale ratio for the balanced Peano curve. (See Appendix A for examples)

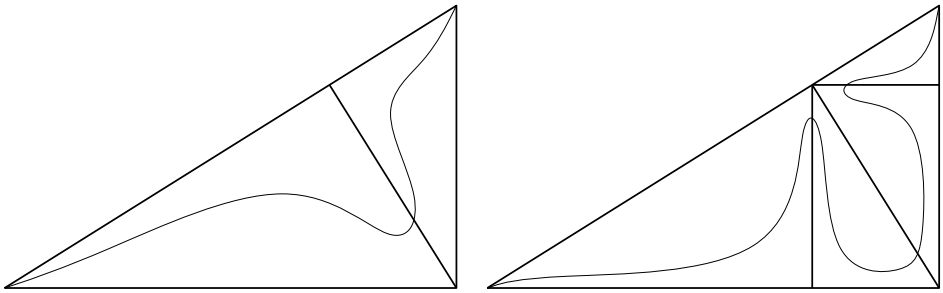
4.3.2 Multiple scale ratios

The implementation provided by [6] requires that the transformations for each section in all rules in a curve definition all have the same scale ratio. Because of this the scale factor of the tail transformation in a probe was always one, and did not have to be stored. We have implemented support for multiple different scale ratios, so that the transformations for sections in any rule can have different scale ratios, and store the tail transformation scale ratio in our

probes to make this possible. However this introduces a new problem which we will discuss here.

When the different sections in a rule can have different scale factors, a degenerate probe can have a degenerate child in which the scale factor of the tail transformation has changed. In turn this child can have another degenerate child with yet another tail transformation scale factor. In general, infinitely many degenerate descendants can exist all with different scale ratio's, so they are all unique. This will prevent the algorithm from terminating as the upperbound will always be infinity.

To illustrate this, we look at the Pólya [11] curve, which divides a right-angled triangle into two smaller right-angled triangles as shown in figure 4.2(a). Expanding these triangles one more time to get rid of the non-90 degree rotations results in the division shown in 4.2(b). As can be seen different subsections have different scale factors.



(a) (b)

Figure 4.2: Pólya curve definition

In figure 4.3, you can see a degenerate probe of the Pólya curve and two of its degenerate descendants, all with different scale ratios for the tail section. Such probes are obtained by expanding a degenerate probe, and picking the last section in the old head section as the new head section, and the first section on the old tail section as the new tail section.

Because the scale ratio of the first and last section in the curve definition are different, this can be repeated an infinite number of times to generate an infinite number of unique degenerate descendants. If we could create an equivalent curve definition where the scale ratios of the first and last section are the same, we would not have this problem.

For the Pólya curve, if the scale ratio of the first section is a, and the scale ratio of the last section is b, and there exists some i, j for which $a^i = b^j$ holds, we can actually construct this new curve definition. Note that not for all a, b such i, j will exist, which means this solution will only work for some Pólya curves.

For the Pólya curve for which $a^2 = b^1$ holds, this new curve definition is shown in figure 4.4. Here we show the division of the triangle twice to make it clear that the first and last section in the rule that divide this triangle have the same scale ratio. It may be interesting to note that the ratio of the long leg and the short leg of the triangles equals $\sqrt{1/\phi}$ for this curve, where ϕ is the golden ratio $\frac{1+\sqrt{5}}{2}$.

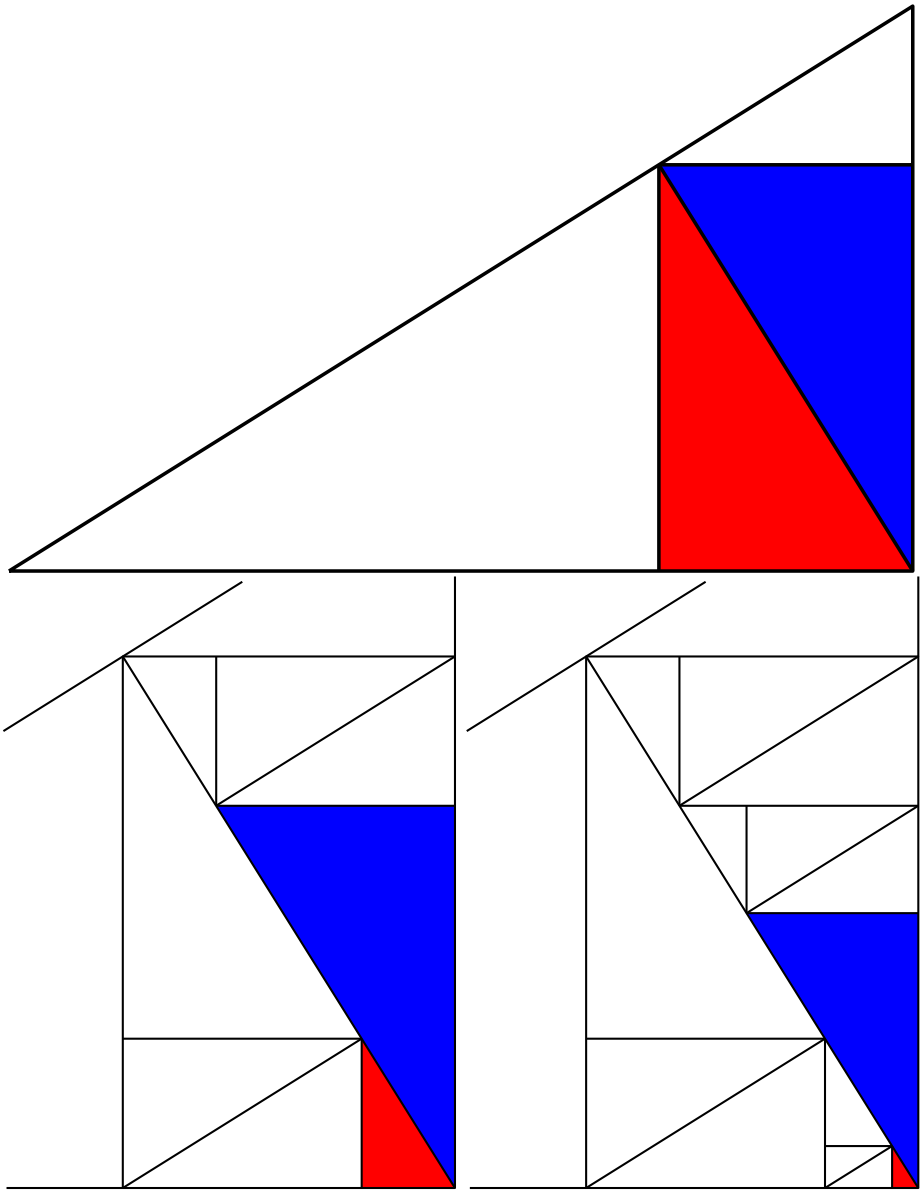


Figure 4.3: Pólya curve degenerate probes (Red=head part, Blue=tail part)

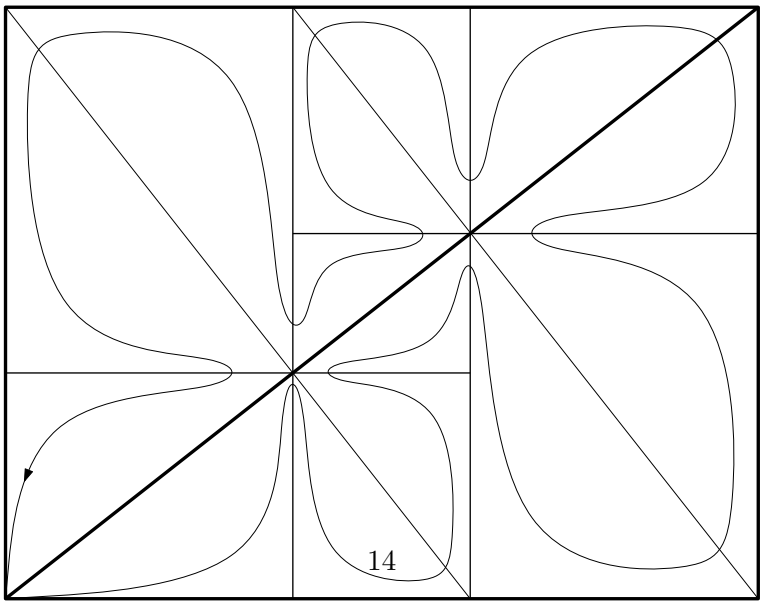


Figure 4.4: Polya curve

Chapter 5

Upper/Lower bounds on measure values for probes

The worst-case approximation algorithm requires that, for a canonical probe P, we can quickly calculate upper ($\mu^+(P)$) and lower ($\mu^-(P)$) bounds for $\mu^*(P)$. The tightness of these bounds will influence the running time of the algorithm.

For enumerating the children of a canonical probe, we will need to store $R_h(P)$ (the rule of the head section) and $R_t(P)$ (the rule of the tail region). Also we need to store $\omega(P)$ (transformation of the tail section), but we don't need to store $\alpha(P)$ (transformation of the head section) because it is always the identity transformation.

We need to pick a good representation of the midsections of canonical probes that enables us to calculate $\mu^*(P)$ efficiently. We also need to store enough information in a probe to enumerate the canonical children of that probe, and be able to calculate that information for each of those children.

Here we will describe the information needed in the probes for each of the measures we have investigated.

5.1 Locality measures

For the locality measures L_∞, L_2 and L_1 only storing the size of the midsection allows for good approximations:

- $\mu^-(P) = \frac{\min_{a \in U(R_h(P)), b \in \omega(P)U(R_t(P))} d_r(a,b)}{|U(R_h(P))|+|M(P)|+|\omega(P)U(R_t(P))|}$
- $\mu^+(P) = \frac{\max_{a \in U(R_h(P)), b \in \omega(P)U(R_t(P))} d_r(a,b)}{|M(P)|}$

When enumerating the children the new midsection size can easily be calculated by adding the size of the segments added to the probe to the old midsection size.

5.2 Bounding-box measures

For the bounding-box measures we also need to store a minimal bounding box for the mid section. Then we can use the following approximations:

- $\mu^-(P) = \frac{|bbox(M(P))|}{|U(R_h(P)|+|M(P)|+|\omega(P)U(R_t(P))|}$

- $\mu^+(P) = \frac{|bbox(U(R_h(P)) \cup bbox(M(P)) \cup \omega(P)U(R_t(P)))|}{|M(P)|}$

For enumeration of child probes the new bounding box can easily be calculated by 'merging' the old bounding box with the new parts we add to the mid sections.

5.3 Boundary ratio

For the new Boundary Ratio measure we introduce, storing the bounding box of the mid section does not help us much. While the probe length can be calculated as before, we now need bounds on the probe boundary, for which we need to store different information in our probes.

In order to approximate the boundary ratio, we need to find a good representation of the boundary of the midsection to store in the probe. Storing the shape of the midsection would meet all our needs, but is not simple as this shape can be arbitrarily complex. Instead we would like to store only the length of the midsection boundary, as we need to know at least that in order to calculate meaningful lower and upper bounds.

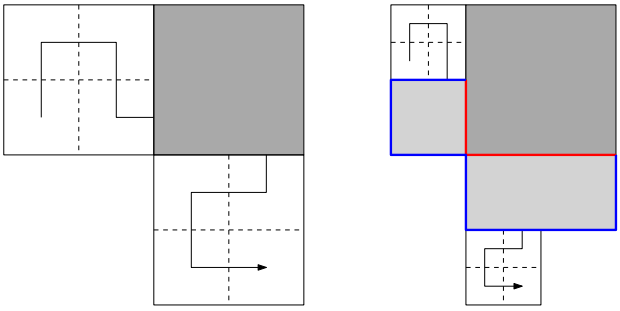


Figure 5.1: A probe and one of its children

When enumerating canonical children of a probe, the child probe will have curve sections added to its mid section like the example shown in figure 5.1. The change in boundary between the parent and the child is the part of the boundary of the new section that does not overlap with the old mid section boundary (marked blue) minus the part that does overlap (marked red). In order to calculate these overlaps we need to know how much the boundaries of the sections in each rule overlap with each other, and how much the boundaries of these sections overlap with the boundary of the unit region of their rule.

In general this information is not easy to store for any curve, like the Pólya curve where a section can overlap with multiple other sections on a single edge (see figure 5.2). Therefore we limited ourselves to curves in which all rules divide their unit region into a regular grid. In this case we can calculate these overlaps easily without needing additional information in our curve definition.

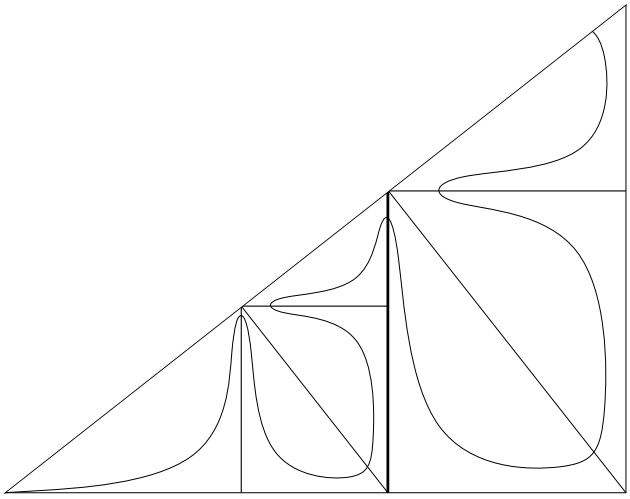


Figure 5.2: Pólya curve - Multiple neighbors on a single edge

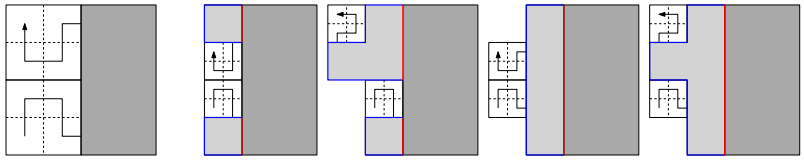


Figure 5.3: A probe and some of its children

In figure 5.3 you can see a probe and some of its children with the relevant boundaries marked. In order to be able to calculate the boundary of the mid section of the children, we keep track of the status of the neighbors of the head and tail sections. This status consists of, for each dimension, in both directions, what is present on the other side of the boundary in this direction. This can be either *empty* (nothing), *mid* (mid section), *head* (head section) or *tail* (tail section). For example for the head section in figure 5.3(a) this will be: *empty* (left), *mid* (right), *empty* (up), *tail* (down). For the head section in figure 5.3(b) it will be: *empty* (left), *mid* (right), *mid* (up), *tail* (down).

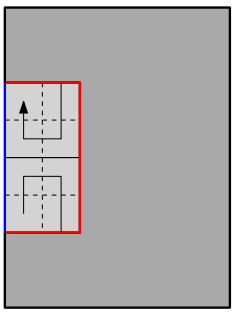


Figure 5.4: Probe midsection boundary length can be longer than the boundary of a curve segment represent by that probe

For the lower bound approximation we need to consider the possibility that length of the midsection boundary of a probe is not a lower bound on the boundary of the curve segments

represented by that probe, like shown in 5.4, where completely filling both the head and the tail section results in a boundary shorter than the boundary of the midsection of the probe. In general the lowest possible boundary of segments represented by a probe is the one of these four cases:

- head section empty, tail section empty : $boundary(midsection)$
- head section empty, tail section filled : $boundary(midsection \cup tail)$
- head section filled, tail section empty : $boundary(midsection \cup head)$
- head section filled, tail section filled : $boundary(midsection \cup head \cup tail)$

When using $neighbors(a, b)$ for the length of the part of the boundary of the a section ($a \in head, tail$) that has class b ($b \in empty, mid, head, tail$) on the other side, we can calculate the values for those four cases like this:

- $boundary(midsection) = boundary(midsection)$
- $boundary(midsection \cup tail) = boundary(midsection) - neigbours(tail, mid) + neighbors(tail, empty) + neighbors(tail, head)$
- $boundary(midsection \cup head) = boundary(midsection) - neigbours(head, mid) + neighbors(head, empty) + neighbors(head, tail)$
- $boundary(midsection \cup head \cup tail) = boundary(midsection) - neighbors(head, mid) - neighbors(tail, mid) + neighbors(head, empty) + neighbors(tail, empty)$

By taking the minimum of these four values we get a tight lower bound on the boundary of all segments of a probe P, which we will call $minboundary(P)$. Because we also have an upper bound on the volume of those segments, $|U(R_h(P))| + |M(P)| + |\omega(P)U(R_t(P))|$, we can now formulate a lower bound on the WB measure:

$$\mu^-(P) := \frac{(minboundary(P)/2d)^{d/(d-1)}}{|U(R_h(P))|+|M(P)|+|\omega(P)U(R_t(P))|}$$

Unlike for the locality and bounding-box measures, there is no easily calculated upper bound for the boundary of a probe, even if we know the boundary of the mid part. The maximum boundary the head or tail sections can add to a probe depends on the curve definition. For curves having only rules which divide their unit region into regular grids, the highest boundary is achieved if the ordering of subregions in that rule allows making a checkerboard pattern on the grid (see figure 5.5). In such a configuration the size of the overlapping boundaries of the subregions will be 0, and completely filling any other subregion will not increase the boundary.

For a head or tail section of a non-degenerate probe, partially filling one subregions is possible, and doing so in the same checkerboard pattern again maximizes the boundary added. This yields a recursive formula for B_n, the maximum head or tail section boundary of a rule dividing a rectangle with boundary 1 into a regular grid of n^2 subregions:

- $B_n \leq \frac{1}{n}\lceil n^2/2 \rceil + \frac{1}{n}B_n$

The right hand side of this equation can be rewritten as a geometric series: $B_n \leq \sum_{k=0}^{\infty}(\frac{1}{n})^k \frac{\lceil n^2/2 \rceil}{n}$. This series converges to $\frac{\lceil n^2/2 \rceil}{n-1}$ which we can use to get an upper bound on the WB measure of a probe:

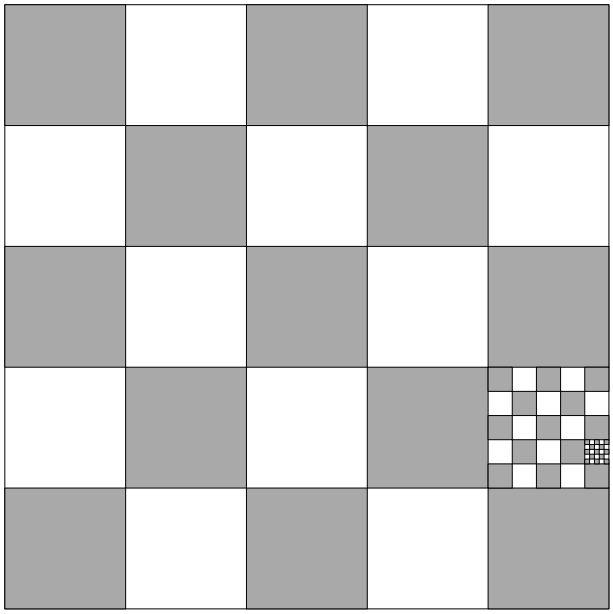


Figure 5.5: Checkerboard pattern on a 5x5 grid

$$\mu^+(P) := \frac{boundary(midsection)+(boundary(U(R_h(P)))+boundary(\omega(P)U(R_t(P))))\lceil n^2/2\rceil/(n-1)}{|M(P)|}.$$

However this upper bound turns out not to be tight enough to achieve reasonable progress in our algorithm. This worst-case upper bound is very pessimistic in the sense that it does not take into account anything about our curve definition except the size of the grid it uses in its rules. In the curve definition the order of the subregions in the rules is given, and this usually prevents grid-like patterns from actually occurring in the curve. In order to improve this we look at all possible ways a head/tail section can be expanded one level deep. By using their rules we can calculate a worst case for each of those ways and take the minimum. The probes already store the neighboring cells of the head/tail section so we can take that information into account as well. However as this calculation is not trivial we precalculate it for each possible combination of neighbor status (here we consider *head* and *tail* the same as *empty*, which makes the calculation much simpler and the bound only slightly less tight).

What we now have is a way to calculate lower and upper bounds on the size of the boundary of the curve segments represented by a probe. For this we store the size of the boundary of the mid section, and neighbor information for the head and tail sections. This neighbor information is also used when refining a probe to calculate how much the boundary size of the midsection will change. Thus the algorithm described in chapter 4 can now be applied to calculate an approximation of the worst-case boundary measure, WB.

Chapter 6

Approximating average values of quality measures

We also implemented a deterministic algorithm to approximate the average measure value for a curve, which is the average of the measure values of all possible subsections of the curve, assuming the start and end of those subsections are uniformly distributed.

We calculate an approximation of this average by keeping track of a set of equations, where the variables in these equations represent the average measure of (canonical) probes. A refinement of a probe can be represented by an equation stating that the average measure of a probe is equal to the weighted average of the average measure of the canonical children of that probe. Our goal is to calculate tight enough bounds on the average measure of the master probe, which represents the entire curve, so that we can give an approximation of the average measure.

In addition to the set of equations, we also use a queue containing probes we have encountered but which have not yet been used to generate an equation. Initially this queue will contain only the master probe so the first equation we will add will describe the average measure of the master probe, and this equation will be used to calculate the bounds on the average measure of the curve. These bounds will simply be the weighted sum of the bounds of the probes in that equation.

Each iteration we will look at one of the probes in the queue and look at its refinement. Then we will add the equation corresponding to that refinement and do gaussian elimination on the set of equations we have so far.

Algorithm ComputeAverageCurveMeasure()
1. $Q \leftarrow$ an empty queue of probes
2. $R \leftarrow$ an empty list of equations
3. Insert the master probe into Q
4. **repeat**
5. $P \leftarrow$ extract probe from Q
6. Add equation corresponding to the refinement of P to R
7. Add any newly encountered probes occurring in that equation to Q
8. Perform Gaussian elimination on the equations in R
9. $upperbound \leftarrow \sum_{t \in Q} \mu^+(t)$
10. $lowerbound \leftarrow \sum_{t \in Q} \mu^-(t)$
11. **until** $upperbound - lowerbound >$ desired precision
12. Report μ^{avg} lies between $lowerbound$ and $upperbound$

The probe we chose to extract in each iteration, is the probe P in Q which has the highest value for $weight(P) * (\mu^+(P) - \mu^-(P))$, where $weight(P)$ is the coefficient of probe P in the

equation for the master probe (which is the first equation in R). This probe P is the probe which has the largest effect on the difference between our current lower and upper bounds.

In practice this algorithm turned out to be too slow to calculate the average measure values with a reasonable precision. It was observed that for most curves the number of already encountered probes, and thus the amount of recursion in the equations, was very low after the first few probes were refined. Using this information we modified the algorithm such that after there are no more degenerate probes in the right hand side of the first equation, it would treat any probe as a new probe, regardless of whether we encountered it before or not. By doing this we only need to keep track of the first equation, and the gaussian elimination is only applied to that equation and one new equation corresponding to a probe.

Chapter 7

Three- and higher-dimensional space-filling curves

Space-filling curves are usually used on only 2 or 3 dimensions. However for some applications more dimensions can be interesting. For this reason we wanted to have some generalized curve definitions which could be used in any number of dimensions. In [9] (based on [2]) we found a method of ordering d-dimensional points along an d-dimensional Hilbert curve, and adapted this to be able to generate a curve definition of this Hilbert curve for any n such that our algorithm could use it.

Some other interesting curves using a 3x3 grid were examined in [3], like the Luxburg Variation 1 curve (which we call the *Coil* curve, see figure 7.2) and the Luxburg Variation 2 curve (which we will call the *Luxburg2* curve, see figure 7.3).

These curves order the nine cells in their grid in the 'Serpentine' pattern also used by the Peano curve. They only differ in the transformations applied to each subsection of the curve. Each subsection can be *flipped* or not, where flipping means the x and y axes in the transformation will be swapped. In addition to this some axes are mirrored in the transformations for both flipped and nonflipped subsections in order to have the entry and exit points of the subsections connect.

The Peano curve (figure 7.1) has none of its subsections flipped, and the Coil curve (figure 7.2) has all of its subsections flipped. The Luxburg2 curve (figure 7.3) alternates between flipped and nonflipped subsections. In these figures the rightmost grid shows which of the subsections are flipped (1) and which of them are not (0).

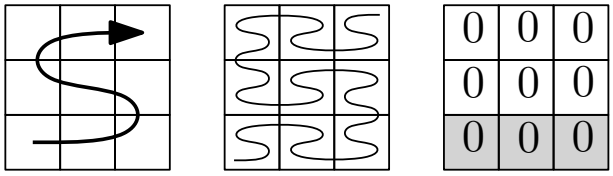


Figure 7.1: Peano in 2D

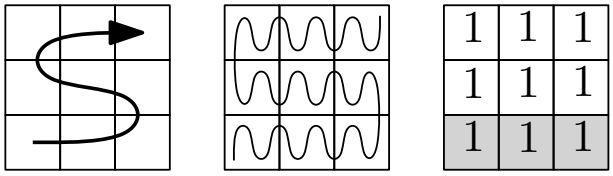


Figure 7.2: Coil Curve in 2D

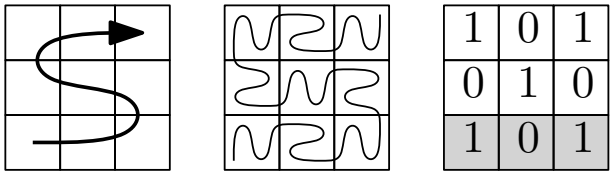


Figure 7.3: Luxburg2 Curve in 2D

The serpentine pattern in 2D starts with the three block on the first row, then the three blocks on the second row in reverse order, and then the three blocks on the third row in the same order as the first row. Your can generalize this for any dimension d by first doing the pattern of dimension $d-1$, and then reversing that pattern for the second row/layer/etc in dimension d, and finally applying the original $d-1$ pattern again for the third row/layer/etc. For 3D this can be seen in figure 7.5.

Looking at Luxburg2 we can see that the pattern of the flips of the first three cells is 101. Note that in this pattern the second cell is the opposite of the first cell while the third cell is the same as the first cell. Looking at the rows in the grid this same pattern occurs again: the second row is the opposite of the first row while the third row is the same as the first row. This observation allows us to extend this pattern into the third dimension by making the second layer the opposite of the first layer and the third layer the same as the first. This is shown in figure 7.4. This approach can be extended to any number of dimensions.

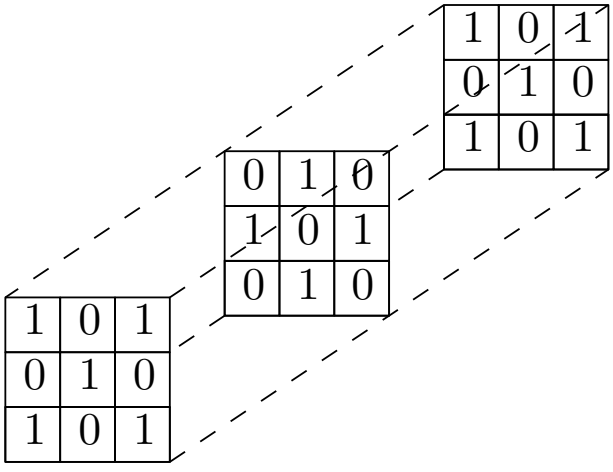


Figure 7.4: Luxburg Curve Pattern in 3D

However for this we need to define what transformation to use for a flipped cell in higher dimensions. We have chosen do a rotation such that axis i ($0 \leq i < d$) before rotation will coincide with axis $i+1$ **mod** n after rotation. This generalizes easily into multiple dimensions and is consistent with what happens in 2D: the swapping of the x and y axis. Example

transformations for 3D are shown in figures 7.5 (nonflipped) and 7.6 (flipped).

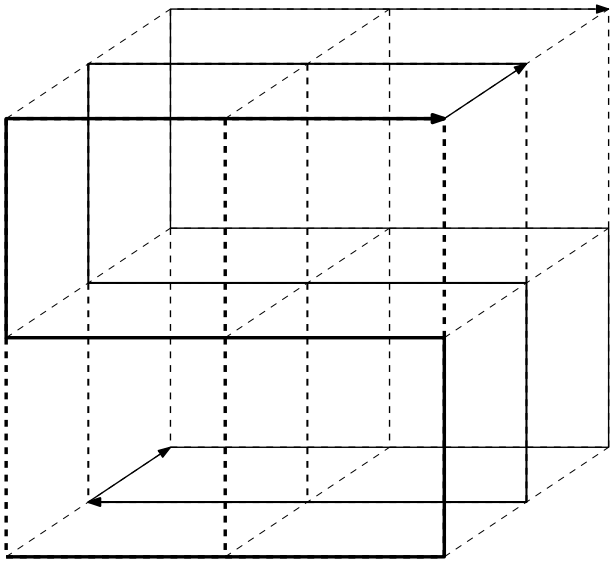


Figure 7.5: Luxburg Curve - Identity Transformation (nonflipped)

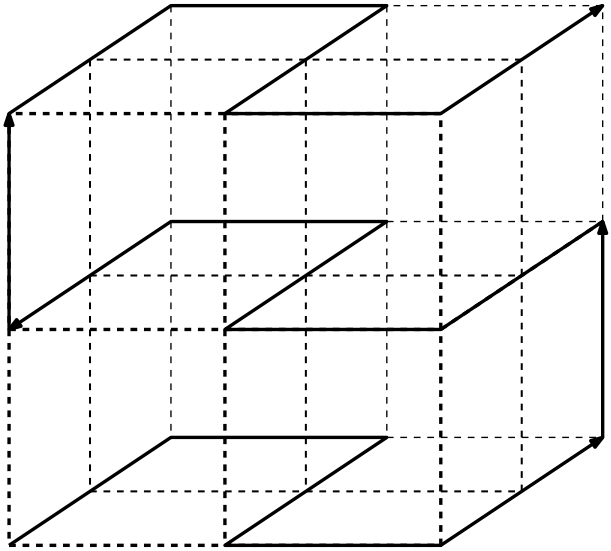


Figure 7.6: Luxburg Curve - Rotation Transformation (flipped)

In figure 7.7 you can see all the actual transformations for the 3D Luxburg2 curve obtained by this method.

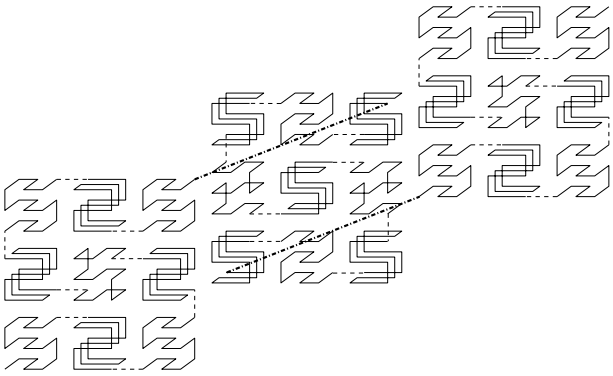


Figure 7.7: Luxburg Curve in 3D

Generalizing this approach we can describe a few similar curves using three numbers which are 0 or 1. The first number determines whether the very first subregion is flipped or not. Then for the second and third cells/rows/layers/etc we need a 0 or a 1 which determines whether that cell/row/later/etc should be inverted (if the 2nd/3rd number is different from the first, the 2nd/3rd cell/row/layer/etc should be inverted). Using this we get patterns describing curves on a 3^n grid for any number of dimensions n. These patterns and the corresponding 2D curves can be seen in figure 7.8.

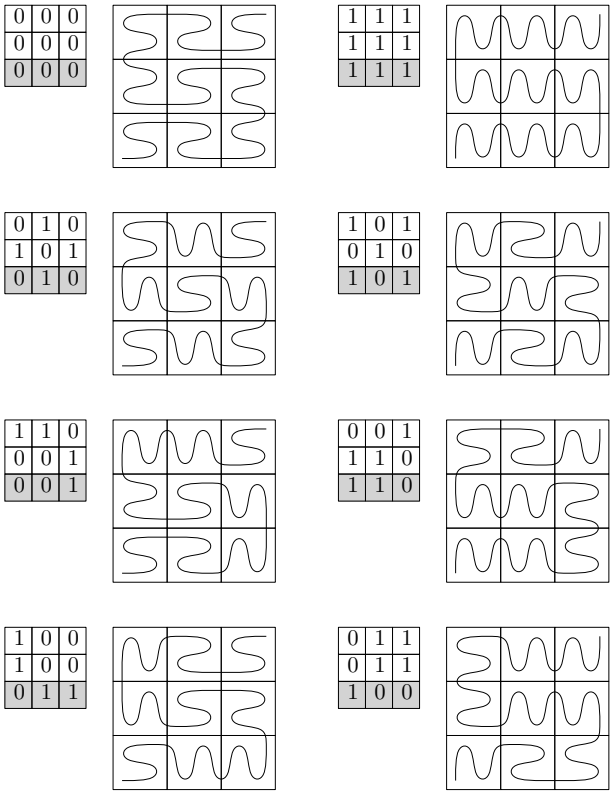


Figure 7.8: Possible patterns for 3^n grid

Some of the patterns correspond to already known curves in 2D, like Peano(000), Coil(111), and Luxburg2(101). Pattern 010 is a slight variation on Luxburg2. To disambiguate we will use 'Luxburg2a' for the original(101) and 'Luxberg2b' for the new variation(010). Also note that pattern 001 and 011 generate the same curve, only rotated 180 degrees. The same holds

for pattern 100 and 110. We will call the curve with pattern 100 'The100a' and the curve with pattern 011 'The100b'.

For the Peano curve we can also create a balanced variant by scaling each dimension i ($0 \leq i < d$) by $3^{i/d}$.

Chapter 8

Results

In this chapter we will analyse the results we obtained by running our algorithm on a few of the space-filling curves we have seen.

First we will show some performance information relating to the heuristic we implemented, and the use of the GiNaC library for symbolic arithmetic. These tests were done calculating all 6 measures ($WL_1, WL_2, WL_\infty / WL_i, WBV, WBB, WB$) of a set of 920 3D Hilbert curves.

Algorithm	Running Time	
Original: FIFO queue, floating point math	2:08:16	(7696 seconds)
Heuristic: Priority queue, floating point math	0:00:43	(43 seconds)
GiNaC: Priority queue, symbolic arithmetic	1:52:36	(6756 seconds)

Here you can see the heuristic we implemented made a huge difference in running time. Also the original algorithm delivered less precision because it expanded more nodes and added many more nodes to the queue, which exhausted available memory and forced the algorithm to return early.

For GiNaC you can see a significant slowdown, even when not actually using any complex expressions in the curve definition.

8.1 Worst-case measures

In tables 8.1 through 8.6 we show the results obtained with our algorithm. The values in these tables are upper bounds on the worst-case measure. The algorithm was configured to stop when the lower bound was less than 1 percent lower than the upper bound, and this precision was reached in most cases. If that was not the case, these tables will show the lower bound in parentheses after the upper bound. The upper bound is shown first because in practice it almost always is tighter than the lower bound, due to the way the heuristic in the algorithm works.

In figures 8.1 through 8.6 the same data, except for the 2D only curves, is shown in graph form. In order to show measures for multiple dimensions in a single figure, so that we can more easily compare these dimensions, these use a logarithmic scale on their y axis.

As can be seen for the WBB, WL_1, and WL_2 measures, the relative ordering of the other curves stays the same as in higher dimensions as in 2 dimensions, except for very minor differences which can be explained by the 1% margin.

For the WB measure this is also mostly the case, except for the Balanced Peano and Hilbert curves, which have swapped positions in 2D compared to the other dimensions.

The WBV measure shows more differences, as Luxburg2a and The100b swap positions with

Luxburg2b and The100a between 2D and 3D, while in higher dimensions all 4 of these curves converge to the same value. Also of interest here is the Hilbert curve, which starts out as third best after Peano and Balanced Peano in 2D, but becomes worse relative to the other curves in each higher dimension, eventually becoming the worst curve in 7D. And while Coil order starts out as 4th best together with Luxburg2a and The100b, in higher dimensions it seems to converge to the same value as Peano and Balanced Peano to have the best value among all curves.

To compare with the Pólya curve (the one with $a^2 = b^1$) we have included some other triangle based curves like the Sierpinski curve [6], which is the square case of the Pólya curve ($a^1 = b^1$). The curve we call 'Stretched' here is a Sierpinski curve stretched to the same dimensions as the Pólya curve with $a^2 = b^1$ we use.

What we see for those curves is that Sierpinski is the best of them, and in cases where there is more than a little difference, Pólya performs better than Stretched.

Table 8.1: WB

Curve	2D	3D	4D	5D	6D
Balanced	4.09	1.74	1.42	1.31	
Coil	3.73	2.08	1.62	1.46	1.39
Hilbert	3.48	1.81	1.51	1.38	1.32
Luxburg2a	3.63	2.07	1.69	1.51	1.42
Luxburg2b	4.70	2.06	1.65	1.50	1.43
Peano	4.71	1.96	1.58	1.46	1.39
The100a	3.81	2.09	1.69	1.51	1.43
The100b	4.71	2.07	1.66	1.51	1.43

Table 8.2: WBB

Curve	2D	3D	4D	5D	6D	7D
Balanced	2.16	4.27	10.9	30.4	89.3	
Coil	2.67	10.0	50.1	258	1360	7230
Hilbert	2.40	5.56	16.7	55.2	191	676
Luxburg2a	2.50	7.25	29.1	139	705	3681
Luxburg2b	2.91	7.23	29.8	141	705	3681
Peano	2.72	10.8	51.3	261	1360	7230
The100a	2.91	10.8	51.3	261	1360	7230
The100b	2.67	10.8	50.1	261	1360	7230
Polya	3.05					
Sierpinski	3.01					
Stretched	3.05					

Table 8.3: WBV

Curve	2D	3D	4D	5D	6D	7D
Balanced	2.00	3.06	3.64	3.90	3.97	
Coil	2.50	3.20	3.72	3.90	4.00	4.00
Hilbert	2.40	3.11	4.74	7.11	10.7	15.1
Luxburg2a	2.50	4.91	7.32	9.82	11.2	11.7
Luxburg2b	2.73	4.68	6.98	9.62	11.2	11.7
Peano	2.00	3.06	3.64	3.90	3.97	4.00
The100a	2.73	4.70	7.32	9.62	11.2	11.7
The100b	2.50	5.07	7.32	9.82	11.2	11.7
Polya	3.04					
Sierpinski	3.00					
Stretched	3.00					

Table 8.4: WL1

Curve	2D	3D	4D	5D
Balanced	8.62	86.1	1037	15015
Coil	10.7	123	1884	33004
Hilbert	9.00	98.3	1275	22744
Luxburg2a	10.0	133	2110	42562
Luxburg2b	11.6	133	2158	43985
Peano	10.7	137	1961	33502
The100a	11.6	133	2134	43980
The100b	10.7	137	2158	42722
Polya	8.29			
Sierpinski	8.05			
Stretched	8.28			

Table 8.5: WL2

Curve	2D	3D	4D	5D	6D
Balanced	4.62	21.3	98.6	476	
Coil	6.67	50.0	336	2377	16925
Hilbert	6.00	26.2	125	675	3859
Luxburg2a	6.25	37.7	251	1673	12744
Luxburg2b	6.67	35.9	247	1671	12738
Peano	8.00	52.7	346	2409	16999
The100a	8.00	52.5	346	2385	16999
The100b	6.67	52.7	342	2409	16981
Polya	4.15				
Sierpinski	4.02				
Stretched	5.10				

Table 8.6: WLi

Curve	2D	3D	4D	5D	6D
Balanced	4.62	17.3	61.6	215	748(66)
Coil	6.67	48.1	312	1920	11616
Hilbert	6.00	24.2	99.7	404	1628
Luxburg2a	5.63	27.5	156	960	5808
Luxburg2b	6.00	27.7	160	968	5824
Peano	8.00	52.0	320	1936	11648
The100a	8.00	52.0	320	1936	11648
The100b	6.67	52.0	312	1936	11616
Polya	4.12				
Sierpinski	4.00				
Stretched	5.09				

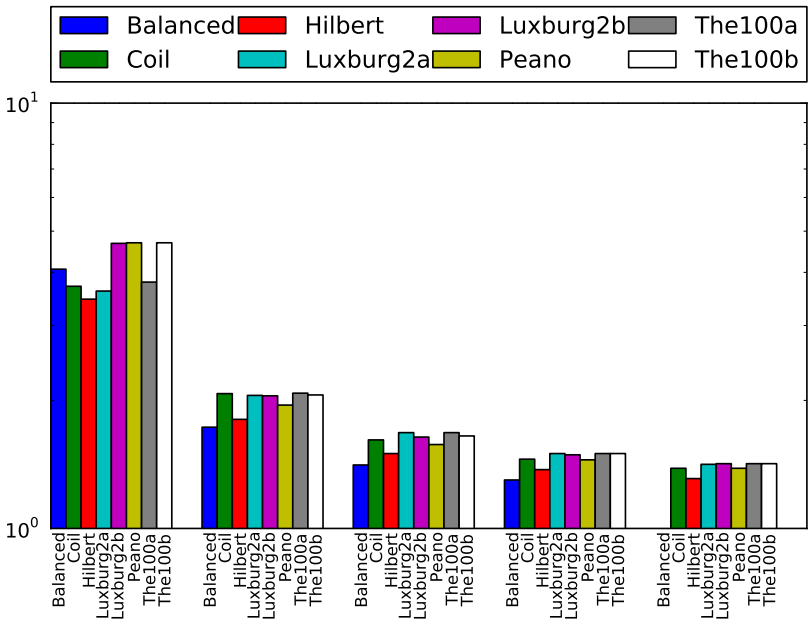


Figure 8.1: WB

8.2 Average measures

The results from the average measure algorithm are show in tables 8.7 through 8.12 and figures 8.7 through 8.12. We show both upper and lower bounds here because we did not reach the desired precision before the algorithm ran out of memory.

As can be seen, due to the limited precision, only very few conclusions can be drawn from this data. A lot of the value ranges overlap which means nothing can be said about which one would have the best value. For the distance measures (AL_1, AL_2, AL_i) no conclusions can be drawn at all, while for the other measures only the 2D and sometimes the 3D results have any meaning.

For AB the only relation that we can be sure of is that the Coil curve is better than the The100b curve in 2 dimensions.

For ABV we can see that, in 2 dimensions, the The100b curve is the worst one, and either the Hilbert, Coil or The100a curve is the best one.

For ABB in 2D, either 100b or Peano is the worst, and either Hilbert, Coil or The100a is the best. For ABB in 3D, either Luxburg2a, Luxburg2b, Peano or The100a is the worst, and Hilbert or Balanced is the best.

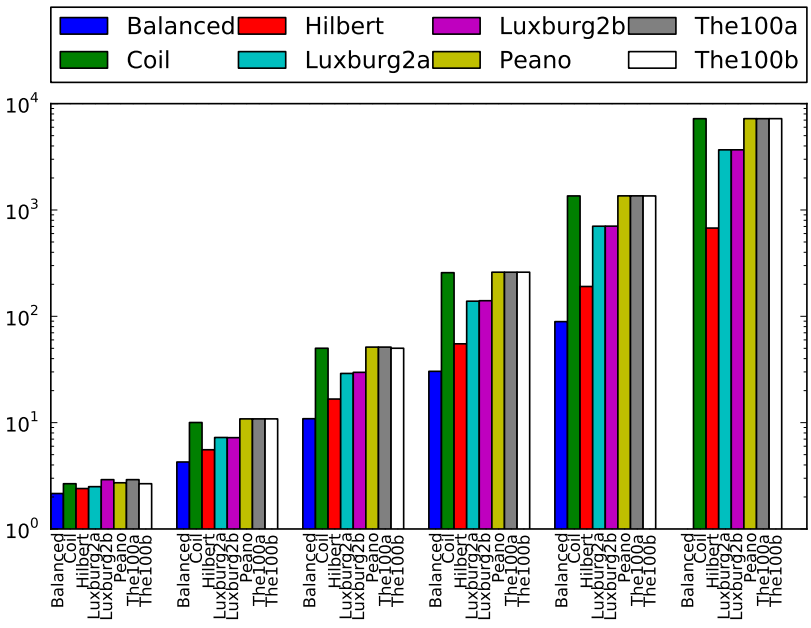


Figure 8.2: WBB

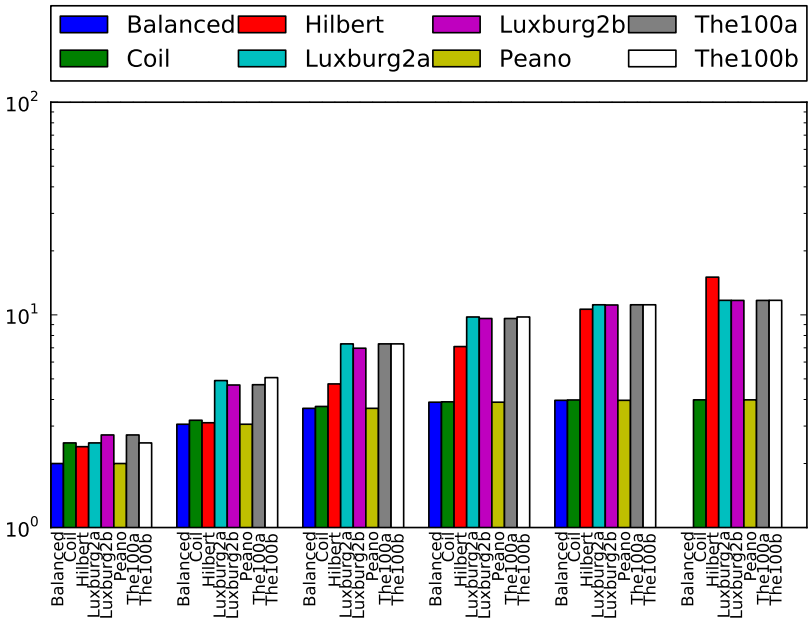


Figure 8.3: WBV

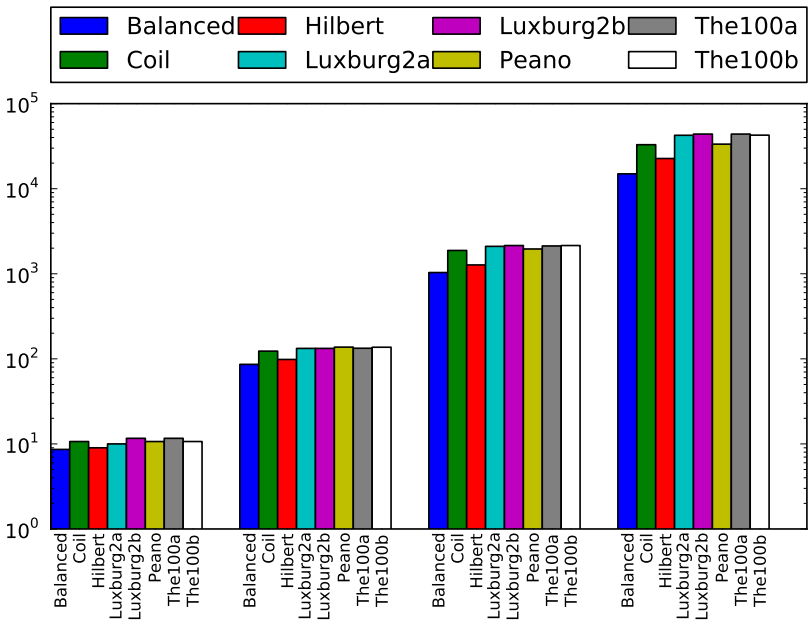


Figure 8.4: WL1

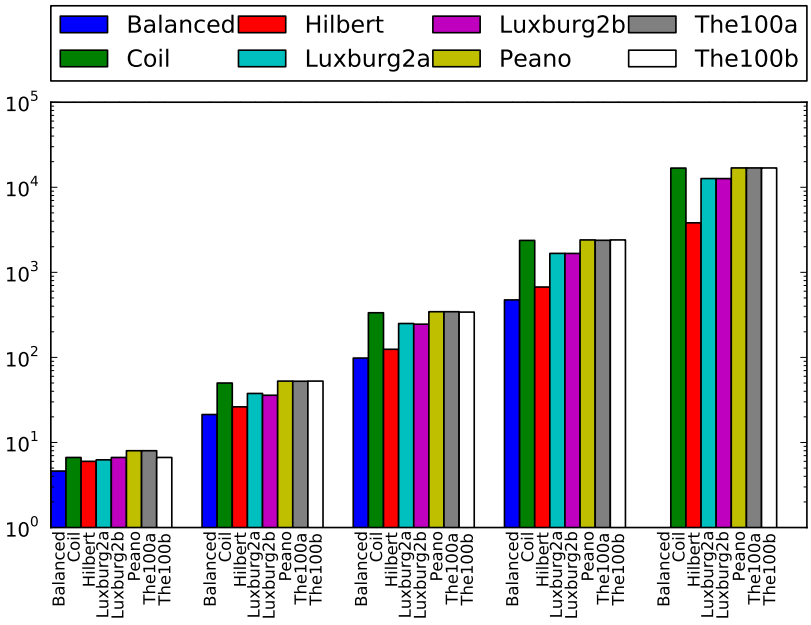


Figure 8.5: WL2

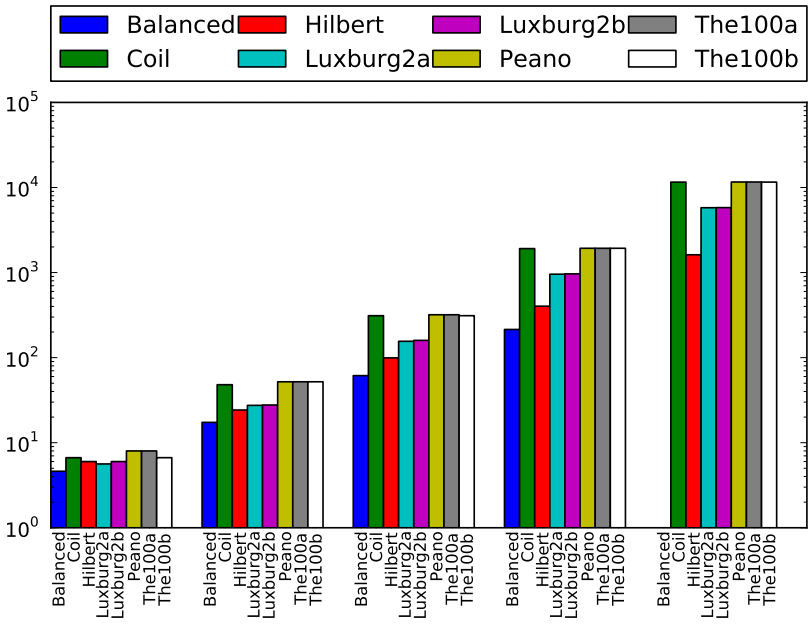


Figure 8.6: WLi

Table 8.7: AB

Curve	2D			3D			4D		
Balanced	1.79	-	2.20	1.30	-	1.39	1.15	-	1.36
Coil	1.58	-	1.83	1.34	-	1.40	1.19	-	1.49
Hilbert	1.77	-	1.95	1.32	-	1.39	1.20	-	1.27
Luxburg2a	1.78	-	2.08	1.36	-	1.43	1.19	-	1.49
Luxburg2b	1.72	-	2.02	1.35	-	1.42	1.19	-	1.49
Peano	1.90	-	2.20	1.37	-	1.43	1.19	-	1.50
The100a	1.63	-	1.93	1.22	-	1.98	1.19	-	1.49
The100b	1.84	-	2.16	1.20	-	2.01	1.19	-	1.49

Table 8.8: ABB

Curve	2D			3D			4D		
Balanced	1.47	-	1.50	1.82	-	1.89	2.04	-	2.51
Coil	1.45	-	1.46	1.93	-	1.97	2.06	-	3.08
Hilbert	1.44	-	1.45	1.78	-	1.82	2.13	-	2.25
Luxburg2a	1.50	-	1.51	1.98	-	2.03	2.07	-	3.09
Luxburg2b	1.49	-	1.50	1.98	-	2.03	2.07	-	3.09
Peano	1.53	-	1.54	2.02	-	2.06	2.07	-	3.09
The100a	1.44	-	1.46	1.99	-	2.04	2.07	-	3.09
The100b	1.54	-	1.55	1.96	-	2.01	2.07	-	3.09

Table 8.9: ABV

Curve	2D			3D			4D		
Balanced	1.40	-	1.43	1.55	-	1.61	1.45	-	1.77
Coil	1.39	-	1.40	1.54	-	1.58	1.36	-	1.85
Hilbert	1.38	-	1.39	1.55	-	1.58	1.60	-	1.69
Luxburg2a	1.43	-	1.44	1.58	-	1.61	1.37	-	1.86
Luxburg2b	1.42	-	1.43	1.57	-	1.61	1.37	-	1.86
Peano	1.41	-	1.42	1.56	-	1.60	1.36	-	1.85
The100a	1.37	-	1.39	1.58	-	1.62	1.37	-	1.85
The100b	1.45	-	1.47	1.55	-	1.58	1.37	-	1.85

Table 8.10: AL1

Curve	2D			3D			4D		
Balanced	1.58	-	2.38	2.2	-	12.9	1	-	226
Coil	1.69	-	2.12	2.5	-	11.1	0	-	259
Hilbert	1.71	-	2.06	2.4	-	10.8	1	-	122
Luxburg2a	1.72	-	2.15	2.5	-	11.3	0	-	259
Luxburg2b	1.71	-	2.15	2.5	-	11.3	0	-	259
Peano	1.73	-	2.17	2.5	-	11.1	0	-	259
The100a	1.64	-	2.17	2.4	-	11.6	0	-	259
The100b	1.71	-	2.20	2.4	-	11.5	0	-	259

Table 8.11: AL2

Curve	2D			3D			4D		
Balanced	1.02	-	1.44	0.80	-	3.37	0.1	-	19.6
Coil	1.06	-	1.28	0.91	-	2.83	0.1	-	20.8
Hilbert	1.06	-	1.24	0.84	-	2.69	0.34	-	9.98
Luxburg2a	1.08	-	1.30	0.90	-	2.88	0.1	-	20.8
Luxburg2b	1.08	-	1.30	0.90	-	2.88	0.1	-	20.8
Peano	1.10	-	1.32	0.93	-	2.86	0.1	-	20.8
The100a	1.04	-	1.31	0.87	-	2.96	0.1	-	20.8
The100b	1.08	-	1.33	0.86	-	2.94	0.1	-	20.8

Table 8.12: ALi

Curve	2D			3D			4D		
Balanced	0.87	-	1.18	0.56	-	1.87	0.09	-	5.79
Coil	0.88	-	1.03	0.63	-	1.49	0.05	-	4.94
Hilbert	0.88	-	1.01	0.56	-	1.38	0.20	-	2.42
Luxburg2a	0.90	-	1.05	0.62	-	1.52	0.05	-	4.94
Luxburg2b	0.90	-	1.05	0.62	-	1.52	0.05	-	4.94
Peano	0.93	-	1.08	0.64	-	1.51	0.06	-	4.95
The100a	0.87	-	1.06	0.59	-	1.57	0.05	-	4.94
The100b	0.90	-	1.08	0.59	-	1.57	0.05	-	4.94

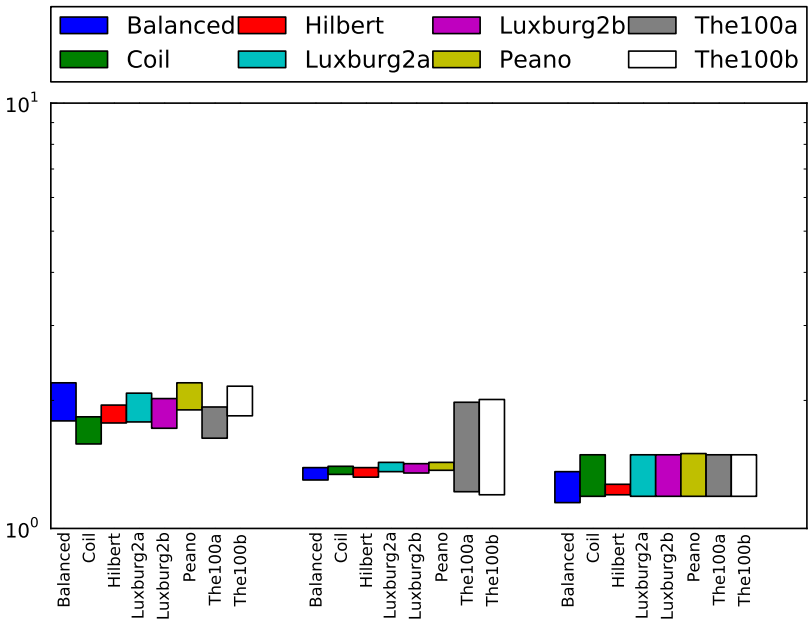


Figure 8.7: AB

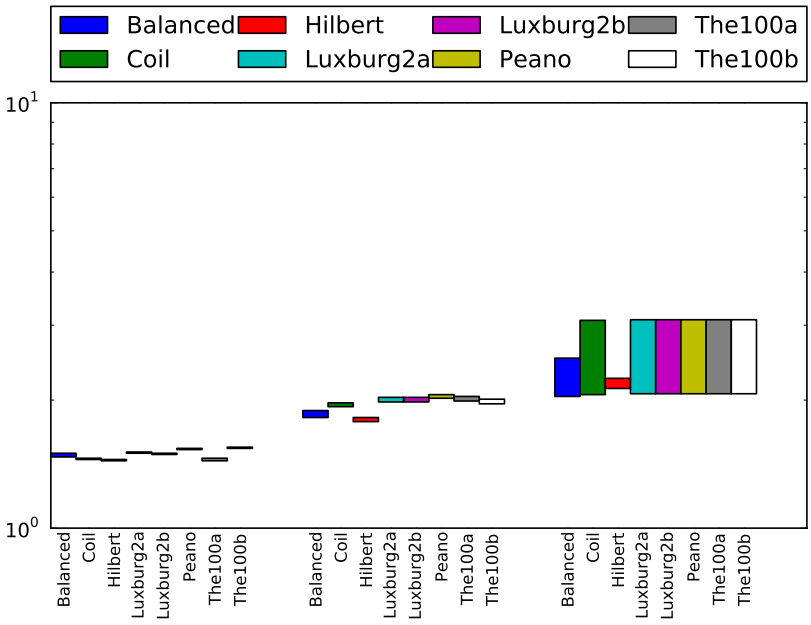


Figure 8.8: ABB

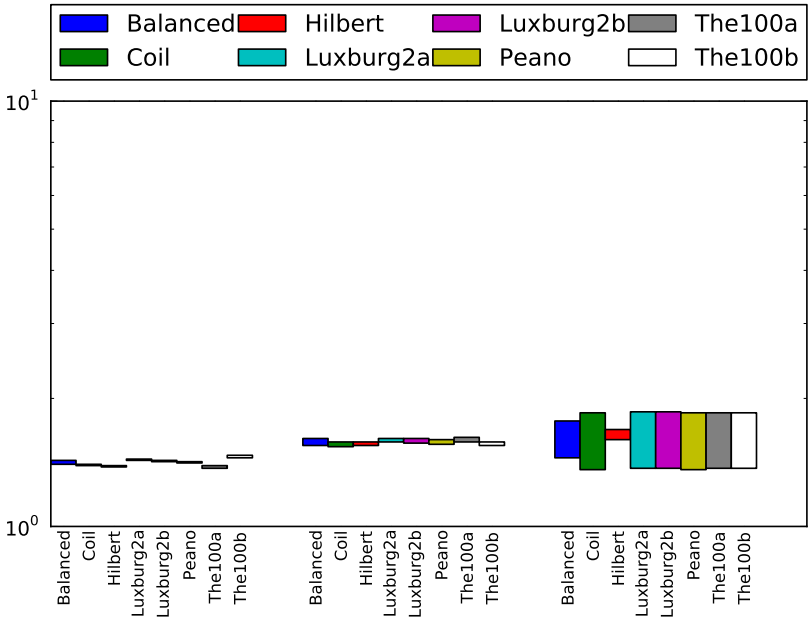


Figure 8.9: ABV

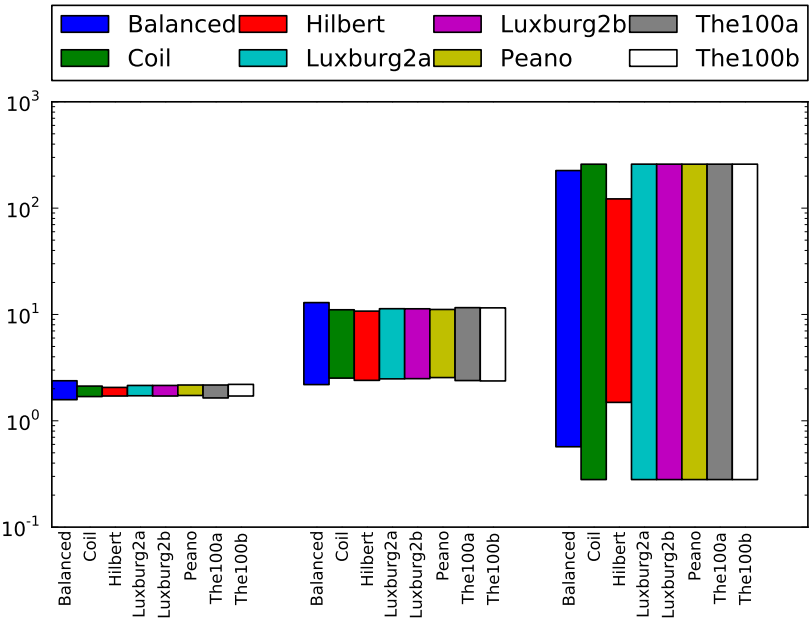


Figure 8.10: AL1

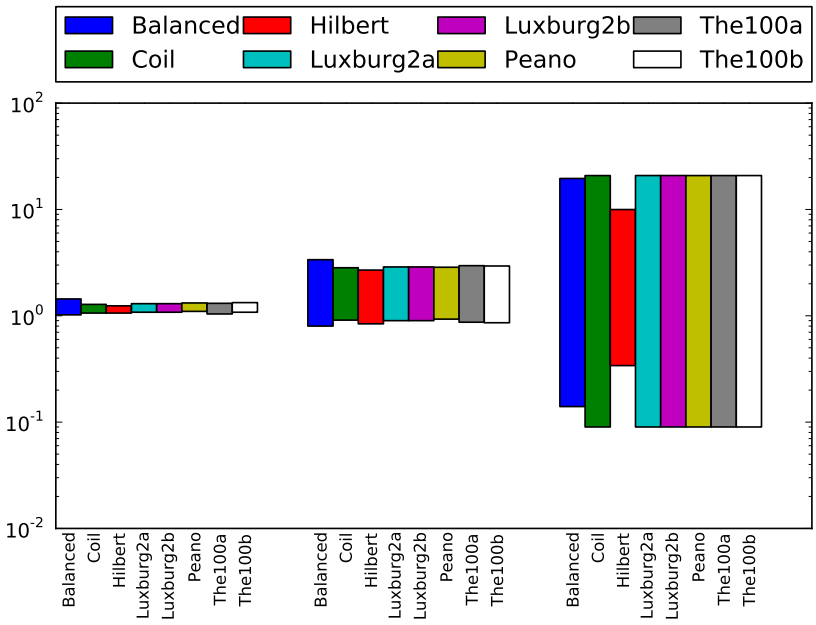


Figure 8.11: AL2

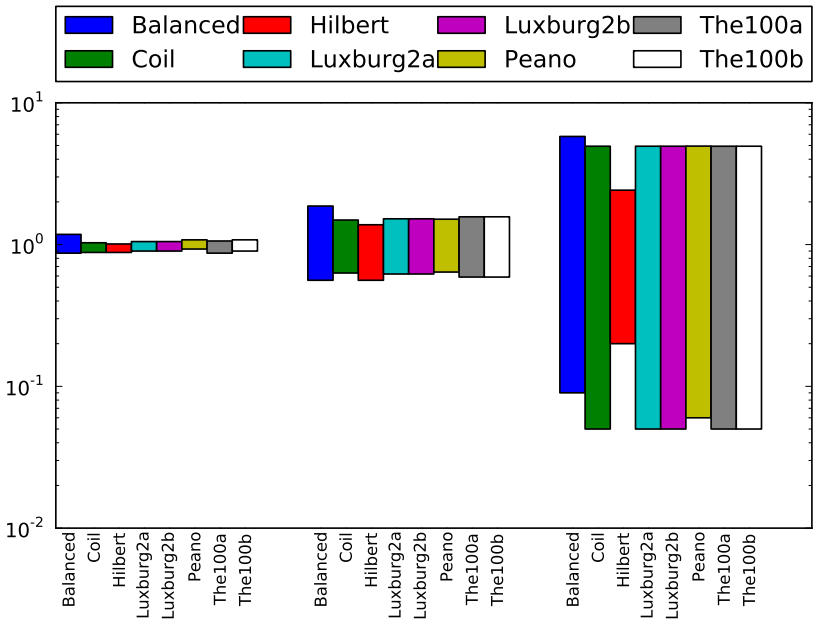


Figure 8.12: ALi

Chapter 9

Conclusions and future work

With the addition of the upper-bound first heuristic, the algorithm for calculating worst-case measures became significantly faster.

A probe representation, along with a method of calculating upper and lower bounds, is introduced for the *WB* measure. This enables the algorithm to calculate worst-case approximations for it.

Implementing the possibility to use symbolic arithmetic to do calculations with expressions instead of number allows the algorithm to handle more possible space-filling-curves.

The algorithm we created and implemented for calculating average measures turned out to be of limited usefulness, yielding only very few useful results, and only for low dimensions.

We have investigated several curves for which we created definitions which can be used for any number of dimensions, and used our algorithm to calculate the measures for each of these curves up to 5 or 6 dimensions.

Of all those curves we investigated, the Balanced Peano curve achieves the best worst-case measure values in almost all cases, only for the *WB* measure in 2 dimensions the Hilbert curve performs better. We conjecture that this trend continues in even higher dimensions, even though we were only able to calculate measure values for balanced Peano up to $d = 5$ (WB) or $d = 6$ (the rest).

In our implementation of calculating the *WB* measure, only curves with rules that divide their unit region into a grid of rectangles of the same size are supported. This is because in such a configuration the question of how long the part of a section boundary is that borders with a particular other section is much simpler, because there is only a fixed number of sections bordering a section, and the length of each of the overlaps is known already. Figuring out exactly what information is needed for a curve and in the probes to allow calculating *WB* measure values for any curve, not just grids, still needs to be done.

The algorithm to calculate average measures did not perform as we had hoped, and other methods of calculating these should be investigated.

Bibliography

[1] GiNaC - http://www.ginac.de/.

[2] Alternative Algorithm for Hilbert's Space-Filling Curve. *IEEE Trans. Comp., April*, pages 424–426, 1971.

[3] Lokalitätsmaße von Peanokurven, Student project report, Universität tübingen, Wilhelm-Shickard-Institut für Informatik. 1998.

[4] Michael Bader, Hans-Joachim Bungartz, Anton Frank, and Ralf Mundani. Space tree structures for pde software. In *International Conference on Computational Science (3)*, volume 2331 of *Lecture Notes in Computer Science*, pages 662–671, 2002.

[5] C. Gottsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE Trans. Image Processing*, 5:794–797, 1996.

[6] Herman Haverkort and Freek van Walderveen. Locality and bounding-box quality of two-dimensional space-filling curves. *Comput. Geom. Theory Appl.*, 43:131–147, February 2010.

[7] David Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[8] Jan Hungershöfer and Jens-Michael Wierum. On the quality of partitions based on space-filling curves. In *International Conference on Computational Science*, volume 2331 of *Lecture Notes in Computer Science*, pages 36–45, 2002.

[9] Doug Moore. Fast Hilbert Curve Generation, Sorting, and Range Queries - http://www.tiac.net/~sw/2008/10/Hilbert/moore/. 2000.

[10] Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890.

[11] G. Pólya. Über ber eine Peanosche Kurve. *Bull. Acad. Sci. Cracovie A*, pages 305–313, 1913.

[12] A.N. Papadopoulos Y.Theodoridis Y. Manolopoulos, A. Nanopoulos. R-trees: Theory and applications. *Springer*, 2005.

Appendix A

Curve Definitions

Here we show some example curve definitions which can be used as input for our implementation.

Peano:

```
name = Peano
dim 2
generator bbox = (0,0)-(1,1) volume = 1 exit = (1,1) scale = 3
section xy 1 section -xy 1 section xy 1
section x-y 1 section -x-y 1 section x-y 1
section xy 1 section -xy 1 section xy 1
.
```

Hilbert:

```
name = Hilbert
dim 2
generator bbox = (0,0)-(1,1) volume = 1 exit = (1,0) scale = 2
section yx 1 section xy 1 section xy 1 section -y-x 1
.
```

Balanced Peano:

```
name = BalancedPeano
dim 2
generator bbox = (0,0)-(1,[sqrt(3)]) volume = [sqrt(3)] exit = (1,[sqrt(3)]) scale = 3
section xy 1 section -xy -1 section xy 1
section x-y -1 section -x-y 1 section x-y -1
section xy 1 section -xy -1 section xy 1
.
```