Chapter 2

Space-Filling Curves

Summary. We begin with an example of a space-filling curve and demonstrate how it can be used to find a short tour through a set of points. Next we give a general introduction to space-filling curves and discuss properties of them. We then consider the space-filling curve heuristic for the traveling salesperson problem and show how a corresponding order of the points can be computed fast, in particular in a probabilistic setting. Finally, we consider a discrete version of space-filling curves and present experimental results on discrete space-filling curves optimized for special tasks.

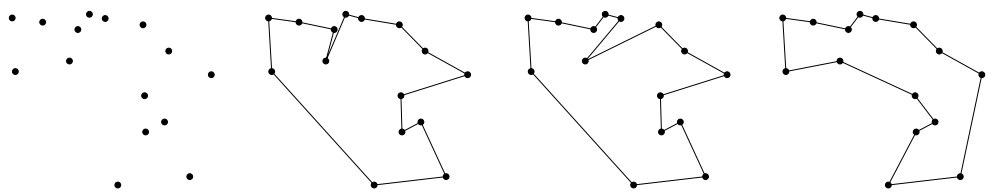
2.1 Introduction

2.1.1 Example: Heuristic for Traveling Salesperson Problem

A space-filling curve maps a 1-dimensional space onto a higher-dimensional space, e.g., the unit interval onto the unit square. We will use space-filling curves in the form of the space-filling curve heuristic for the NP-hard Euclidean traveling salesperson problem [67]. The Euclidean traveling salesperson problem is the problem of finding the shortest closed tour through a set of points.

Figure 2.1 shows a point set, tours through the point set constructed using the space-filling curve heuristic with two different space-filling curves, and the shortest tour through the point set.

We demonstrate the space-filling curve heuristic for this task by the example of the two-dimensional Hilbert curve [78]. Consider the following construction



(a) Point set (b) Heuristic using Moore curve (c) Heuristic using Sierpiński curve (d) Shortest tour

Figure 2.1: Short tours through a point set.

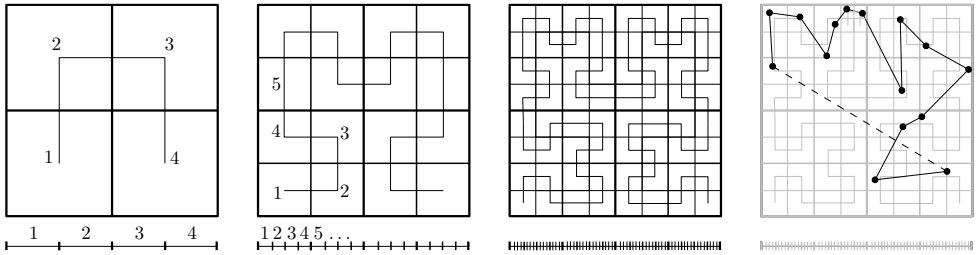


Figure 2.2: Hilbert curve and order.

of a map from the unit interval to the unit square: Divide the unit interval into four intervals, divide the unit square into four squares, and assign each interval to one of the squares. This is shown in Figure 2.2(a). This process can be continued recursively and furthermore it can be done in such a way that neighboring intervals are assigned to neighboring squares. The first three steps of this construction are shown in Figure 2.2(a-c). In the limit this yields a surjective, continuous map from the unit interval to the unit square.

For our purposes it suffices to repeat the subdivision process until we can read off the order of the points along the curve. In this example, for one of the squares one more subdivision step is necessary. Figure 2.2(d) shows the resulting order (and indicates the additional subdivision). We call this order of the points a *space-filling curve order* – or in the special case of the Hilbert curve *Hilbert order* – of the point set.

Since the Hilbert curve starts in the lower left corner and ends in the lower right corner, the last edge of the round-trip (shown in Figure 2.2(d) as dashed line) is likely to be long. Thus, the Hilbert curve is only suited for applications where a short path through the points is needed. If a closed tour is required as in the case of the traveling salesperson problem then a closed space-filling curve is more suited, i.e., a curve starting and ending in the same point. The Moore and the Sierpiński curve used in Figure 2.1(b–c) are examples for closed curves. See Section 2.1.5 for more details on these curves. The space-filling curve heuristic for the traveling salesperson problem yields a tour which is at most a logarithmic factor longer than the shortest tour [122]. In the case of uniformly distributed points it is with high probability only longer by a constant factor [65].

2.1.2 Applications

In general, space-filling curves allow one to reduce higher-dimensional proximity problems, e.g., nearest neighbor search, to a one-dimensional problem. Solving such a problem typically involves searching and sorting in the one-dimensional space. Algorithms for searching and sorting have been studied extensively, in particular I/O-efficient and parallel algorithms. Using space-filling curves, these algorithms can be applied to higher-dimensional proximity problems. A further advantage of space-filling curves like the Hilbert curve is their recursive nature which allows to use them for hierarchical indexing of higher-dimensional data.

A common application of space-filling curves is storage and retrieval of multi-dimensional data in a database [88]. Proximity problems for which space-filling curves have been used frequently are approximate nearest neighbors search and finding closest pairs (see [99] and the references therein). Algorithms for these problems using space-filling curves have been analyzed theoretically [32, 96, 93]. Space-filling curves can be used for multi-dimensional optimization problems [128].

Space-filling curves are also used for low-dimensional problems as in the case of the traveling salesperson problem [122]. For instance, they are used to index meshes for parallel and distributed computing [111] and to organize and process raster data (i.e., data on a grid), e.g., images [38, 118, 147], terrains [94], and volumetric data [119].

2.1.3 History of Space-Filling Curves

In 1877/78 Cantor [28, 30] demonstrated that there is a bijective function between any two finite-dimensional smooth manifolds independent of their dimension. This in particular showed that such functions between the unit interval and the unit d-cube for $d > 1$ exist. In 1879 Netto [110] proved that if the dimensions of the manifolds are different such a function is necessarily discontinuous. The question whether a surjective, continuous map from the unit interval to the unit square exists remained open. In 1890 Peano [120] answered this question positively by presenting such a map, now known as Peano (space-filling) curve. In 1891 Hilbert [78] gave the first geometric construction for such a map which we presented above. Many examples followed, of which we will consider the curves by Moore (1900) [105], Lebesgue (1904) [90], and Sierpiński (1913) [133]. For further background on space-filling curves we refer to Sagan [126].

2.1.4 Definition

We refer to surjective, continuous maps from the unit interval to the unit d-cube ($d > 1$) as *d-dimensional space-filling curves*. Note that we refer to the map (and not its image) as curve. We restrict to this kind of space-filling curves because they are useful for computing orderings of point sets.

In general, a space-filling curve is a continuous map from the unit interval to $\mathbb{R}^d$ for which the image is a region with positive Jordan content. A region has positive Jordan content if it can be approximated from the inside and the outside by (disjoint) unions of cubes, and the upper limit for the inner content (defined in the canonical way) and the lower limit for the outer content are equal and positive.

A space-filling curve cannot be injective [110]. For the Lebesgue measure there exist injective, continuous maps from $\mathbb{R}$ to $\mathbb{R}^d$ (with $d > 1$) for which the image has a positive measure [116]. In general the continuous images of line segments can be characterized as the compact, connected, locally connected sets (see [126]).

2.1.5 Examples

Next we present several examples of space-filling curves. For each curve, we give the rule for the recursive, geometric construction. Furthermore, we show the order of the cells of the subdivision after the second subdivision step and after several steps more. We will only present two-dimensional examples. For a three-dimensional example see [125, 126], for higher-dimensional curves see [2].

Hilbert Curve. The Hilbert curve is shown in Figure 2.2. A representation of its construction rule is shown in the left and middle part of Figure 2.3(a). The figure shows that the square with arrow to the left is replaced by four squares. The ordering of the squares is indicated by the grey curve. The arrows show the orientations of the squares. The right part of the figure shows the result of a second application of the rule. Figure 2.4(a) shows the order of the cells after 4 subdivision steps.

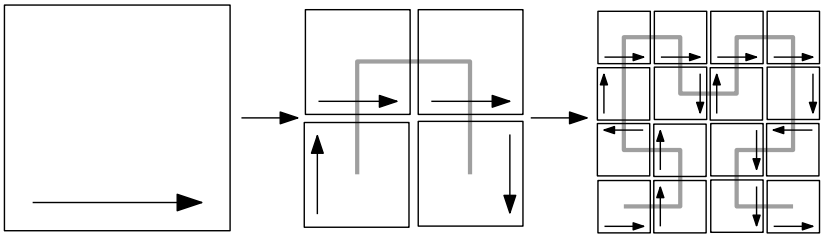
Moore Curve. The Moore curve is a closed version of the Hilbert curve. It is obtained by concatenating four copies of the Hilbert curve as shown in Figure 2.3(b). Figure 2.4(b) shows the order for the Moore curve after 4 subdivision steps.

Peano Curve. The Peano curve uses a ternary subdivision instead of a binary subdivision. It is the first known space-filling curve, found by Peano in 1890. Its original construction was arithmetic but it can be constructed geometrically as shown in Figure 2.3(c). The order after the third subdivision is shown in Figure 2.4(c).

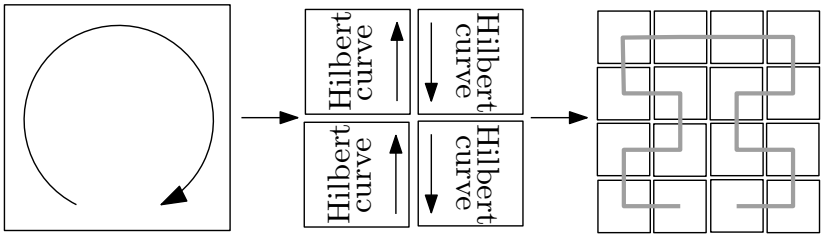
Lebesgue Curve. The Lebesgue curve uses a similar subdivision as the Hilbert and Moore curves. Figure 2.3(d) shows its construction rule and Figure 2.4(d) the subdivision after 4 steps.

Constructing the curve exactly like the Hilbert curve as described in Section 2.1.1 would result in a discontinuous mapping. Instead the construction uses the *Cantor set* [29, 135], i.e., the set of numbers in the unit interval for which the ternary expansions do not contain 1. Geometrically it can be constructed by dividing the unit interval into three equal parts, removing the middle (open) part, and continuing the construction recursively on the first and third (closed) parts.

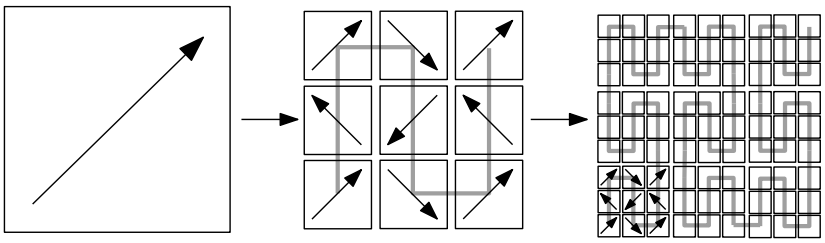
On the Cantor set the construction of the curve is done as for the Hilbert curve on the unit interval. The parts of the unit interval not belonging to the Cantor set are used for linear interpolation. For example in the first step of the geometric construction of the Cantor set the unit interval is subdivided into 3 parts. For the construction of the Lebesgue curve the first part, i.e., $[0, 1/3]$, is mapped to the left half of the unit cube, i.e., $[0, 1/2] \times [0, 1]$. The third part, i.e, $[2/3, 1]$, is mapped to the right half, i.e., $[1/2, 1] \times [0, 1]$. On the middle part, i.e., the open interval $(1/3, 2/3)$, the curve interpolates between the points $(1/2, 1)$ and $(1/2, 0)$.



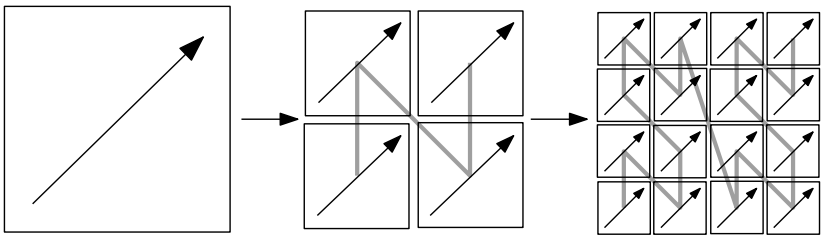
(a) Hilbert curve



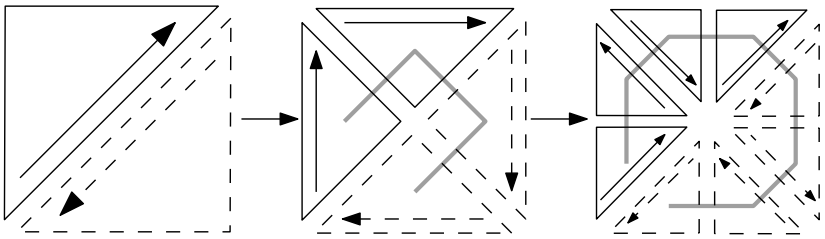
(b) Moore curve



(c) Peano curve

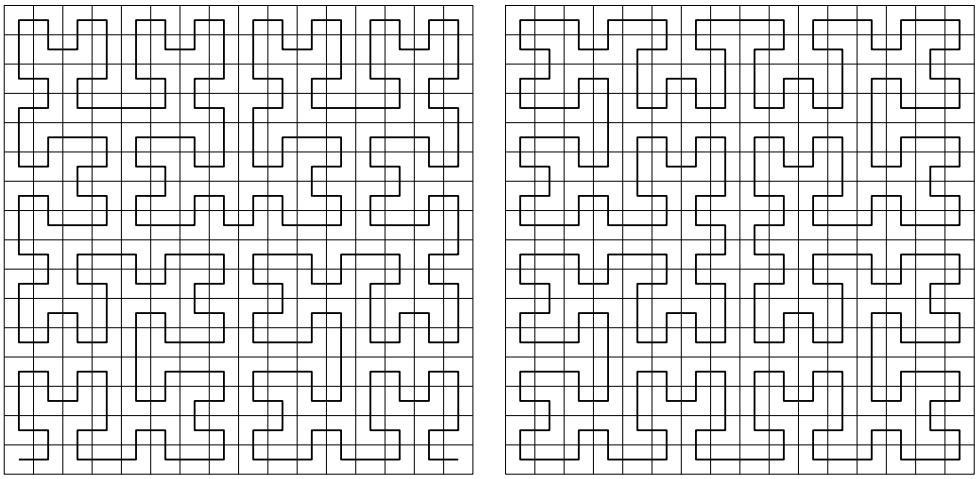


(d) Lebesgue curve



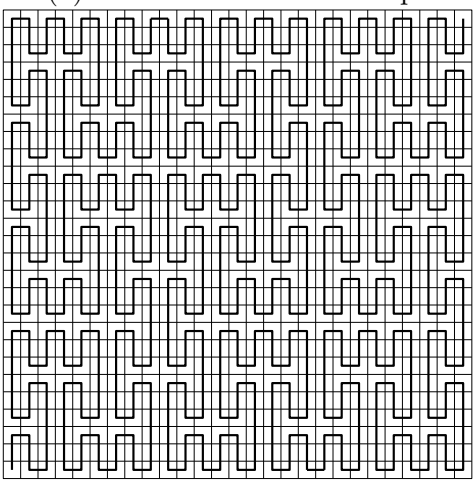
(e) Sierpiński curve

Figure 2.3: Recursive, geometric construction of space-filling curves.

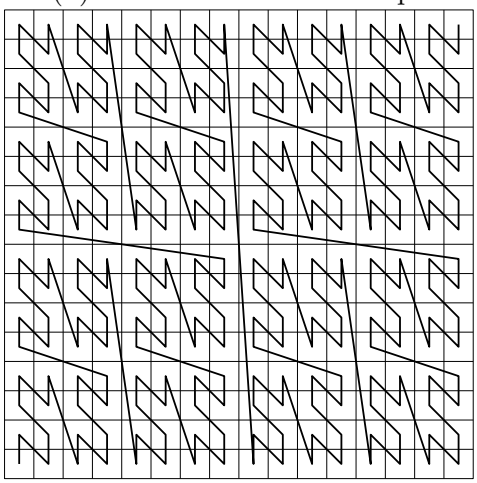


(a) Hilbert curve after 4 steps

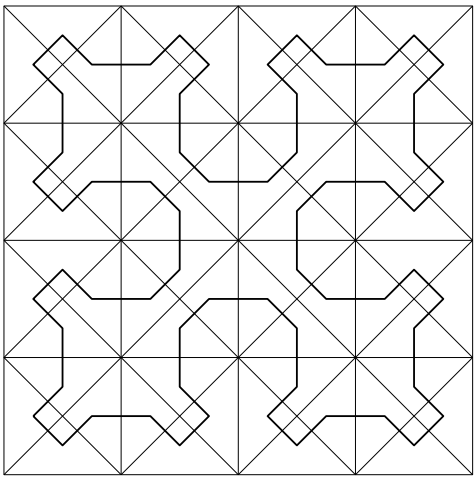
(b) Moore curve after 4 steps



(c) Peano curve after 3 steps



(d) Lebesgue curve after 4 steps



(e) Sierpiński curve after 5 steps

Figure 2.4: Geometric construction of space-filling curves after several subdivisions.

Although the construction using the Cantor set yields a continuous curve, the resulting curve lacks important properties which other space-filling curves share. In particular it is not bi-measure preserving and is not Hölder continuous of order $1/2$ (or $1/d$ in dimension d); see Section 2.1.6 for more details on these properties. Nonetheless, the Lebesgue curve is useful and used for ordering points because of its simplicity.

Sierpiński Curve. In contrast to the previously discussed space-filling curves the Sierpiński curve is based on a triangular and not a cubical subdivision. Its geometric construction is shown in Figure 2.3(e). The construction for the upper left and the lower right triangle are the same (rotated by an angle of π). The order after the fifth step of the subdivision is shown in Figure 2.4(e). The Sierpiński curve is used by Platzman and Bartholdi [122] for constructing a short tour through a set of points.

2.1.6 Properties of Space-Filling Curve

In the following we survey important properties of space-filling curves. The two properties we will need in the next chapters are *Hölder continuity* and the *bi-measure-preserving property*. In the choice and formulation of the properties we will mostly follow the exposition of Steele [137]. These properties are not shared by all space-filling curves, in particular the properties do not hold for the Lebesgue curve. We will illustrate them by the example of the Hilbert curve. Let $\psi_d \colon [0,1] \to [0,1]^d$ denote a d-dimensional Hilbert curve. Note that for dimensions $d > 2$ there is more than one d-dimensional Hilbert curve [2].

Hölder Continuity. A map $f \colon I \subset \mathbb{R} \to \mathbb{R}^d$ is called Hölder continuous of order $1/k$ (or short Hölder-$1/k$) with Hölder constant c_f if for all $s, t \in I$

$$\|f(s) - f(t)\| \leq c_f |s - t|^{1/k}.$$

This property is also referred to as *Lipschitz continuity*. Space-filling curves in dimension d are typically Hölder-$1/d$. This is not the case for the Lebesgue curve since it linearly interpolates outside of the Cantor set.

For the two-dimensional Hilbert curve ψ_2, it follows directly from the recursive construction that the image of an interval of length $1/4^m$ ($m \geq 0$) stays in two neighboring squares of side length $1/2^m$. Therefore the distance between the endpoints is at most $\sqrt{5}/2^m$. Thus, for $s, t \in [0,1]$ and $m > 0$ with

$$1/4^{m+1} < |s - t| \leq 1/4^m$$

we get

$$\|\psi_2(s) - \psi_2(t)\| \leq \frac{\sqrt{5}}{2^m} = \frac{2\sqrt{5}}{\sqrt{4^{m+1}}} \leq 2\sqrt{5}|s - t|^{1/2}.$$

This yields an upper bound of $2\sqrt{5}$ on the Hölder constant c_{ψ_2} of the two-dimensional Hilbert curve but actually the exact value of $c_{\psi_2} = \sqrt{6}$ is known [10].

The bound of $2\sqrt{5}$ also holds for the Moore curve. For the Peano curve the argument above yields a bound of $3\sqrt{5}$ and for the Sierpiński Curve a bound

of $4\sqrt{2}$. A 2-dimensional space-filling curve cannot have a Hölder constant smaller than $\sqrt{5}$ [130].

For the d-dimensional Hilbert curve the argument above yields

$$\|\psi_d(s) - \psi_d(t)\| < 2\sqrt{d+3}|s-t|^{1/d}.$$

Nowhere Differentiability. The Hilbert curve, the Moore curve, the Peano curve, and the Sierpinski curve are nowhere differentiable [126]. In contrast, the Lebesgue curve is differentiable almost everywhere, since the Cantor set has measure 0.

Bi-Measure-Preserving Property. By its recursive construction the Hilbert curve maps an interval to a region with an area equal to the length of the interval. In general we have for d-dimensional Hilbert curves that for any Borel set $A \subset [0,1]$

$$\lambda_1(A) = \lambda_d(\psi_d(A)),$$

where λ_1 and λ_d denote the one- and d-dimensional measure, respectively. This property is called the *bi-measure-preserving property*. It again holds for all curves mentioned except for the Lebesgue curve.

Dilation and Translation Property. If we consider the second step of the recursive construction of the Hilbert curve, we see that the Hilbert curve starts with a scaled copy of itself, followed by several translated and rotated copies. In general, a space-filling curve ψ has the *dilation property* and the *translation property* if there is a $p \geq 2$ such that for all $s, t \in [0,1]$

$$\|\psi(s) - \psi(t)\| = \sqrt{p}\|\psi(s/p) - \psi(t/p)\|$$

and for all $1 \leq i \leq p$ and $s, t \in [(i-1)/p, i/p]$

$$\|\psi(s) - \psi(t)\| = \|\psi(s + 1/p) - \psi(t + 1/p)\|.$$

For the Hilbert curve p can be chosen as any power of 16.

2.2 Space-Filling Curve Heuristic for the Traveling Salesperson Problem

As discussed in Section 2.1.1, space-filling curves can be used in a heuristic for the Euclidean traveling salesperson problem. Conceptually, the heuristic consists of two steps (Algorithm 1).

Algorithm 1: General space-filling curve heuristic with space-filling curve ψ

Input: Point set $\{x_1, \ldots, x_n\} \subset [0,1]^d$
Output: Ordering of the point set

1 For $i = 1, \ldots, n$ compute $t_i \in [0,1]$ with $\psi(t_i) = x_i$.
2 Sort $x_1, \ldots x_n$ by the order $x_i \prec x_j \;:\Leftrightarrow\; t_i < t_j$ for $1 \leq i, j \leq n$.

For space-filling curves like the Hilbert curve which can be constructed by recursive subdivision, preimages of points in $[0,1]^d$ can be computed based on the subdivision. We will discuss this in more detail in Section 2.2.2.

The space-filling curve heuristic has been popularized by Bartholdi and Platzman [9, 122] but has a longer history (see [137], p. 49). If the result of the heuristic should be a closed tour – as in the case of the traveling salesperson problem – then a closed space-filling curve suggests itself.

2.2.1 Properties

Experimental Evaluation of Tour Length. Compared to other heuristics for the traveling salesperson problem, in experiments the space-filling curve heuristic is fast but also results in relatively long tours [83]. For uniformly distributed points the resulting tours have a length which is about 35% longer than the Held-Karp lower bound on the shortest tour length (see [6] for details on computational aspects of the traveling salesperson problem). For clustered points the tour is between 41% and 96% longer, for instances from the TSPLIB it is about 40% longer. In contrast the lengths of the tours computed by the *Concorde** implementation of the Lin-Kernigham algorithm are only about 2.5% longer for uniform points. But the time needed is also by far longer, e.g., about 180 times longer for ten million points.

Worst-Case Tour Length. For the space-filling curve heuristic to be effective the space-filling curve should map two numbers of small distance to points of small distance. This is guaranteed for most of the classical space-filling curves by their Hölder continuity of order $1/d$. The following was observed by Platzman and Bartholdi [122] for the two-dimensional case.

Observation 2.1. *Let $\psi\colon [0,1] \to [0,1]^d$ be Hölder continuous of order $1/d$ with Hölder constant c_ψ. Then the length of the one-way, non-closed tour through n points computed by the space-filling curve heuristic is bounded by*

$$c_\psi \cdot (n-1)^{1-1/d}.$$

If ψ is closed then the length of the closed tour is bounded by

$$c_\psi \cdot n^{1-1/d}.$$

Proof. By the Hölder continuity of ψ the length of the one-way, non-closed tour is bounded by

$$c_\psi \sum_{i=1}^{n-1} |t_{i+1} - t_i|^{1/d}$$

where t_i denotes the ith preimage of the ordered preimages for $i = 1, \ldots, n$. This sum is maximized by equidistant preimages and therefore bounded by

$$c_\psi \cdot (n-1) \cdot (n-1)^{-1/d} = c_\psi \cdot (n-1)^{1-1/d}.$$

The proof for the closed tour is the same except for the additional summand $|t_1 - t_n|^{1/d}$. □

*`http://www.tsp.gatech.edu/concorde.html`

By the same argument the length of the computed tour using squared Euclidean distances between points is constant in the plane:

Observation 2.2. *Let $\psi\colon [0,1] \to [0,1]^2$ be Hölder continuous of order $1/2$ with Hölder constant c_ψ. Then the length of the one-way, non-closed tour – and if ψ is closed also the length of the closed tour – through n points computed by the space-filling curve heuristic is bounded by c_ψ^2.*

Worst-Case Approximation Factor. The following properties hold for the two-dimensional Sierpiński Curve, and are conjectured to be true for other Hölder-$1/d$, bi-measure preserving, closed space-filling curves. The tour through n points computed by the space-filling curve heuristic is at most by a factor $O(\log n)$ longer than the shortest tour [122]. This bound is tight [13].

Probabilistic Analysis of Tour Length. The expected tour length $\mathrm{E}\left[L_n^{\mathrm{SFC}}\right]$ computed by the space-filling curve heuristic through n independent uniformly distributed points in the unit square is analyzed by Gao and Steele [65, 66]. Interestingly, although the expected tour length is of order $\sqrt{n}$, the quotient $\mathrm{E}\left[L_n^{\mathrm{SFC}}\right]/\sqrt{n}$ does not converge. Instead the following quotient converges:

Theorem 2.3 (Gao, Steele [65, 137])**.** *If a heuristic tour is built using a space-filling curve that satisfies the dilation property, the translation property, and the bi-measure-preserving property, then there exists a continuous function φ of period one such that*

$$\lim_{n\to\infty} \frac{E\left[L_n^{\mathrm{SFC}}\right]}{\sqrt{n\varphi(\log_p n)}} = 1 \quad \textit{almost surely,}$$

where p is the integer appearing in the dilation and translation property.

The tour length is with high probability close to its expected value:

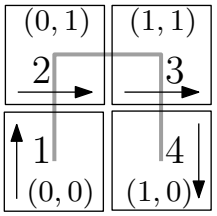
Theorem 2.4 (Gao, Steele [65, 137])**.** *If a space-filling curve has the bi-measure-preserving property, then there are constants A and B such that for all $t \geq 0$,*

$$P\left[\left|L_n^{\mathrm{SFC}} - E\left[L_n^{\mathrm{SFC}}\right]\right|\right] \leq B\, \exp(-At^2/\log t).$$

The space-filling curve heuristic in combination with Talagrand's inequality [142] can be used to prove Gaussian tail bounds (i.e., bounds as in Theorem 2.4) for geometric optimization problems, in particular the traveling salesperson problem and the Steiner tree problem (see [101, 137]).

2.2.2 Computation

For the Hilbert curve and curves with a similar recursive subdivision scheme a preimage of a point can be computed iteratively.



init	coord	rank	orient
	(0, 0)	1	
	(1, 0)	4	
	(0, 1)	2	
	(1, 1)	3	

Figure 2.5: Subdivision and corresponding table.

We will illustrate the computation by the example of the two-dimensional Hilbert curve. Consider a point (x, y) in the unit square for which the coordinates have a binary representation with k bits each, i.e.,

$$x = \sum_{i=1}^{k} a_i \cdot 2^{-i}, \qquad y = \sum_{i=1}^{k} b_i \cdot 2^{-i}.$$

In the first subdivision step, (a_1, b_1) determines the square in which the point lies. A point on the boundary between two squares has more than one preimage and could be assigned to either of the squares. Moreover in the case of the Lebesgue curve, a point might have even more preimages (outside of the Cantor set). From these preimages we choose a preimage out of the interval mapped to the square corresponding to (a_1, b_1).

In the ith subdivision step for $2 \leq i \leq k$, (a_i, b_i) determines the square in which the point lies and for this square

1. its *rank* in the space-filling curve order relative to the other squares of the current subdivision step and

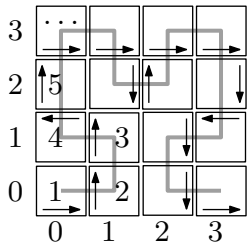
2. the *orientation* of the curve in the space-filling curve in the square.

Figure 2.5 shows for one subdivision step a table storing for one initial orientation (*init*) the bits specifying the squares (*coord*), the ranks (*rank*), and the resulting orientations (*orient*). The orientation of the space-filling curve in a sub-square is needed for the next subdivision step. For a subdivision process using squares there are at most 8 such orientations, i.e., the symmetries of the square. We can either store a table for each initial orientation, or store a table for one orientation and provide transformations of the table for the other orientations. In our implementation we store a table for each orientation.

With k table lookups we can compute a preimage of (x, y). By computing preimages for all points and sorting them by a worst-case optimal comparison-based sorting algorithm, this yields an algorithm which runs in $O(kdn + n \log n)$ time where n is the number of points in $\mathbb{R}^d$ and k the average number of bits per coordinate. This conceptually corresponds to the algorithm described by Platzman and Bartholdi except that their algorithm uses a Sierpiński Curve.

In the following we will consider the following extensions and variants:

- Computing several steps of the subdivision at once.
- Using a sorting algorithm based on bucketing.
- Combining different sorting strategies based on the problem size.



init	coord	rank	orient
	(0, 0)	1	
	(1, 0)	2	
	(2, 0)	15	
	(3, 0)	16	
	(0, 1)	4	
	...	...	...

Figure 2.6: Two subdivision steps and corresponding table.

Several Steps at Once. After two subdivision steps a square is subdivided into 16 squares. For these 16 squares we can store the ranks and the orientations in tables. An example is shown in Figure 2.6. This halves the number of subdivision steps needed to compute a preimage of a point.

The ranks and orientations for m subdivision steps can be stored in tables of size of order $O(4^m)$. As long as $m \leq \log_4 n$ the tables can be computed in linear time (in n) and stored in linear space. As index of the square (a, b) for $0 \leq a, b < 2^m$ we use the integer $a + 2^m b$.

Assume we can in constant time compute the index (in the tables) of the square in which a point lies. Then this allows us to determine for a point a logarithmic number of bits of its preimage in constant time. This is not possible in the standard model of computation, i.e., using only constant-time basic arithmetic operations and comparisons.

The assumption can be fulfilled with a floor function which can handle a logarithmic number of bits at once. If the unit square is subdivided into $2^m \times 2^m$ squares then for a point (x, y) with $0 \leq x, y < 1$ the index of the square is

$$\lfloor 2^m x \rfloor + 2^m \lfloor 2^m y \rfloor,$$

i.e., we use a floor function on m bits. As input for the next subdivision step we use the point

$$(2^m x - \lfloor 2^m x \rfloor, 2^m y - \lfloor 2^m y \rfloor).$$

In the theoretical analysis we will instead of a constant-time floor function use the following assumption: We assume that for a point set of size n in $[0, 1]^d$ for $d, n > 0$ and a regular subdivision of $[0, 1]^d$ into $O(n)$ cubes (called *buckets*), the assignment of points to buckets can be determined in $O(n)$ time. We will refer to this assumption as *linear time bucketing*.

For the algorithm to run fast in practice, the implementation should take into account typical issues concerning lookup tables, in particular problems related to memory use. Memory access to tables is expensive when there are a large number of cache misses. In an implementation this can be prevented by using small tables, e.g., on 256 buckets.

Also for the theoretical analysis, it is not necessary that the number of buckets is linear in the number of points. Instead it suffices that the number of buckets is chosen such that the number of points is bounded by a polynomial in the number of buckets. For example, assume we have m buckets and at most

m^k points for a constant k. Then after k subdivision steps we have m^k buckets, i.e., as many buckets as points.

Using comparison-based sorting the table-lookup approach allows to compute the space-filling curve order in $O(nd \log k + n \log n)$ time for n points in $\mathbb{R}^d$ with an average number of k bits per coordinate assuming linear time bucketing.

Sorting with Buckets. The computation of the preimages involves computing point-to-bucket assignments. Therefore it makes sense to combine the computation of the preimages with a sorting algorithm based on bucketing.

Within one level of subdivision *bucket sort* or *counting sort* can be used. Further subdivision of buckets naturally suggests using *radix sort*. Algorithms based on radix sort either start with the least or the most significant digit. For an overview of radix sort algorithms see [112] and for a general overview of sorting algorithms see [41, 87]. For I/O-efficient implementations and experimental evaluations of sorting algorithms based on bucketing, see for instance [81, 123].

Although the sorting algorithms described below can be used in a more general context, we will refer to

- the items to be sorted as points and
- the number by which a point is sorted in the current subdivision level as rank.

Radix sort starting with the least significant rank/digit (*LSD-first radix sort*) is only applicable if the number of (relevant) bits per point is known in advance. Even then it does not combine well with space-filling curve computations since the computation of the preimages starts with the top level of the subdivision, i.e., the most significant rank. A possible way to apply LSD-first radix sort in this situation is to use it only for the most significant ranks, and then to switch to a different sorting strategy [98].

Instead of LSD-first radix sort, we use radix sort starting with the most significant rank/digit (*MSD-first radix sort*). We will focus on two variants, *forward radix sort* [4, 112] and *adaptive radix sort* [55, 63]. For both algorithms there are fast implementations [5]. We present adaptive radix sort since we use it for computing a space-filling curve order in our implementation. It is also the fastest algorithm in the experimental comparison of MSD-first radix sorting algorithms and quicksort by Andersson and Nilsson [5]. We present forward radix sort because of its good theoretical running time which is also simple to analyze in a probabilistic setting. In contrast, the probabilistic analysis of adaptive radix sort is more involved [48, 143, 144].

Adaptive Radix Sort. The principle idea of adaptive radix sort is to adapt the number of bits used in one step and the number of buckets to the number of elements to be sorted. Algorithm 2 outlines the computation of a space-filling curve order using adaptive radix sort.

Algorithm 2: Computing a space-filling curve order using adaptive radix sort

Input: Point set in $\mathbb{R}^d$
Output: SFC order of the point set

1 Compute a bounding cube.
2 Partition the cube into approximately n grid cells by a $2^\kappa \times \cdots \times 2^\kappa$ grid with $\kappa = \lfloor \log \sqrt[d]{n} \rfloor$.
3 Compute the space-filling curve order and orientations for grid cells by the geometric construction scheme of a space-filling curve.
4 Assign points to cells.
5 Repeat from step 2 on grid cells using the cell as bounding cube, the orientation as determined in step 3, and the number of points in the cell for determining κ.

In our implementation we use counting sort within a subdivision step. The basic idea of counting sort is that in a first pass of the point set the number of points per bucket is determined. This is then used in a second pass to insert the points at the correct position in an auxiliary array. After these two passes the points within buckets are still unsorted. In our implementation we sort the points in a bucket recursively, re-using the original and the auxiliary array (with possibly the original array as auxiliary array if the points of the bucket are currently stored in the auxiliary array).

Forward Radix Sort. Forward radix sort combines the advantages of MSD-first and LSD-first radix sort. Like other algorithms based on MSD-first radix sort it only inspects the distinguishing prefixes. The main disadvantage of many algorithms based on MSD-first radix sort is that the algorithm is called recursively for every bucket separately while LSD-first radix sort proceeds to work on the whole input. Forward radix sort overcomes this disadvantage by processing all buckets that are not yet sorted in one scan.

For a description of the algorithm see [4, 112]. As in Algorithm 2 we again first compute a bounding cube, and apply radix sort by subdividing this cube and using lookup tables. The general idea of forward radix sort is to do the following in each pass:

1. store with every point its prefix, i.e., the part of its preimage computed so far,

2. sort all points by their rank in the next subdivision step, ignoring in which cube it was previously,

3. sort points by their prefix, keeping points with the same prefix in the order computed in step 2

The steps described above are not applied to parts of the point set for which we know that they are already correctly sorted. Furthermore, some additional work is necessary to reduce the number of empty buckets that are visited. This

can be done either by further preprocessing, or by switching to a comparison-based sorting algorithm if the number of unsorted points drops below some threshold.

The running time of the algorithm is formulated for binary strings. It depends on the average number $\bar{B}$ of bits in a distinguishing prefix and on the number w of bits in a machine word on a unit cost random access machine.

Furthermore, the running time depends on the subroutine used for sorting integers. We only consider the simplest version with bucketing as a subroutine for sorting integers. For the case $w \in O(\log n)$ the other variants do not improve on the running time asymptotically, and $w = \log n + O(1)$ is the number of bits we can sort at once in a lookup table of linear size. For larger w better bounds can be obtained by using different subroutines [4, 112].

Theorem 2.5 (Andersson and Nilsson [4, 112])**.** *Given n binary strings let $\bar{B}$ be the average number of bits in a distinguishing prefix. Assume the number of bits in a machine word is in $\Omega(\log n)$. Then forward radix sort using bucketing as a subroutine for integer sorting sorts the binary strings in time*

$$\Theta\left(n\left(\frac{\bar{B}}{\log n}+1\right)\right).$$

This running time can also be achieved by using the algorithm by Paige and Tarjan [117]. If $w \in \Theta(\log n)$ then this running time is optimal on a unit cost random access machine with word length w since the minimal time needed for inspecting all distinguishing bits and to process each element at least once is in $\Omega(n\bar{B}/w + n)$.

Combining Different Sorting Strategies. To prevent visiting too many empty buckets in our implementation of adaptive radix sort, we switch to a comparison-based algorithm, insertion sort, if a bucket contains only a small number of points. In our experiments this leads to a considerable speed-up. This matches the experimental results by Andersson and Nilsson [5]. They report that switching to insertion sort if only about 10–30 points are left, reduces the running time of forward radix sort by about 40% and of adaptive radix sort by about 50%.

Another example for a sorting algorithm combining different sorting strategies is *introspective sort* [108]. Introspective sort is the standard sorting algorithm in the C++ standard template library [109]. The algorithm sorts using quicksort but with heapsort as a fallback for the case that a certain recursion step is reached. Furthermore, instead of recursing on arbitrary small subproblems, sequences of a length below a certain threshold are first left unsorted. They are then sorted in a final pass using insertion sort.

In our implementation we apply radix sort choosing a lookup table for the space-filling curve depending on the number of points (in the current bucket) except until the number of points in a buckets drops below a threshold (e.g., 32 points). Then the algorithm switches to insertion sort, again using the lookup tables. If the points are stored in the auxiliary array (see above) before the insertion sort, the points are written back to the original array during this step.

square	disc	chessboard	normal	Kuzmin	line	cube	Buddha
0.3813	0.4118	0.4608	1.3570	0.6238	0.6239	0.3041	0.4432

Table 2.1: Running times for space-filling curve computation in CPU seconds.

In the following we report on the running time of our implementation of adaptive radix sort switching to insertion sort if the number of points in a bucket dropped below 32. The measurements were performed on a Intel(R) Pentium(R) 4 CPU 3.00GHz with 2.048KB cache size and using the g++ 3.3.5 compiler with the option $-$O2. All running times were averaged over 10 runs. In two dimensions we measured the running times for 1 000 000 points for various distributions. The distributions tested are points distributed uniformly in a square, uniformly in a disk, pseudo-uniformly with a density varying according to a chessboard pattern, normally with identity matrix as covariance matrix, according to the Kuzmin distribution, and close to a line. The point distributions used are described in more detail in Section 4.5 and are illustrated there in Figure 4.5. In three dimensions we measured the running times for 500 000 uniformly distributed points and for 543 652 points from the *Happy Buddha* data set[†]. The running times are summarized in Table 2.1.

There are two variants in our implementation of the algorithm. First, instead of applying insertion sort one can choose to leave short sequences unsorted. Even though this does not result in a correct space-filling curve order, the resulting order might be sufficient for the application. Second, instead of choosing the size of the lookup tables depending only on the number of points in a bucket, suitable thresholds for lookup tables can be precomputed using a sample point set. The motivation for this is that a good choice for the size of the lookup table might depend not only on the size of the point set but also on the point distribution and aspects not related to the point set like the cache size.

Probabilistic Analysis of Running Time. Next we analyze the expected time needed to compute the space-filling curve order of independent identically distributed points in $\mathbb{R}^d$ using forward radix sort. In the analysis we only use that the running time is in $O\left(n\left(\frac{\bar{B}}{\log n}+1\right)\right)$. Thus, the analysis holds for any radix sort algorithm with this running time.

For a point, the number of bits in a distinguishing prefix can be expressed in terms of the size of the bounding cube and the distance of the point to a nearest neighbor.

Lemma 2.6. *Let ℓ be the side length of the bounding cube used to compute the space-filling curve order of a point set P. If the L_∞-distance of a point $p \in P$ to a nearest neighbor in P is larger than $s > 0$ then a prefix with $d\log_2(\ell/s)$ bits suffices to distinguish p.*

Proof. We first handle the case of $[0,1]^d$ as a bounding cube, i.e., $\ell = 1$. Consider the subdivision process, where in one step a cube is replaced by 2^d cubes.

[†]Stanford $3D$ scanning repository, `http://www-graphics.stanford.edu/data/3Dscanrep`

If after k steps ($k \geq 0$) a point lies in a cube only containing this point then the distinguishing prefix for this point has at most kd bits.

After k steps the side lengths of the cubes in the subdivision is 2^{-k}. If for a point the L_∞-distance to a nearest neighbor is larger than 2^{-k} the cube containing the point is empty. Therefore, $k \leq \log_2(1/s)$, and the number of distinguishing bits is bounded from above by $d \log_2(1/s)$.

If the side length of the bounding cube is ℓ then this scales the side lengths of the cubes in the subdivision by ℓ. Thus the number of distinguishing bits is bounded from above by $d \log_2(\ell/s)$. □

For independent identically distributed points $X_1, \ldots, X_n$ in $\mathbb{R}^d$ let L and S be defined as

$$\begin{aligned} L &:= \max\{\, \|X_i - X_j\|_\infty \mid 1 \leq i, j \leq n\} \\ S &:= \min\{\, \|X_1 - X_i\|_\infty \mid 1 \leq i \leq n\}. \end{aligned}$$

For $X_1, \ldots, X_n$ Lemma 2.6 yields

$$\mathrm{E}\left[\bar{B}\right] \leq \frac{d}{\log 2} \left(\mathrm{E}\left[\log L\right] + \mathrm{E}\left[\log S^{-1}\right]\right).$$

This gives the following bound on the expected time to compute a space-filling curve order.

Proposition 2.7. *For independent identically distributed $X_1, \ldots, X_n$ in $\mathbb{R}^d$ and the random variables L, S defined as above, a space-filling curve order can be computed in expected time*

$$O(n \frac{E\left[\log L\right] + E\left[\log S^{-1}\right]}{\log n})$$

using forward radix sort and assuming linear time bucketing.

Next we bound $\mathrm{E}\left[\log L\right]$ and $\mathrm{E}\left[\log S^{-1}\right]$ from above by simpler expressions.

Proposition 2.8. *Let $X_1, \ldots X_n$ in $\mathbb{R}^d$ be independent identically distributed random variables and let $\varepsilon > 0$ be a real number with $E[\|X_1 - X_2\|_\infty^{\varepsilon}] < \infty$ and $E[\|X_1 - X_2\|_\infty^{-\varepsilon}] < \infty$. Then with linear time bucketing, the expected time for computing a space-filling curve order of the points using forward radix sort is linear in n.*

Proof. It suffices to prove $\mathrm{E}\left[\log L\right] + \mathrm{E}\left[\log S^{-1}\right] \in O(\log n)$. First we have

$$\mathrm{E}\left[\log L\right] + \mathrm{E}\left[\log S^{-1}\right] = \frac{1}{\varepsilon} \left(\mathrm{E}\left[\log L^{\varepsilon}\right] + \mathrm{E}\left[\log S^{-\varepsilon}\right]\right).$$

By Jensen's inequality we get

$$\mathrm{E}\left[\log L^{\varepsilon}\right] + \mathrm{E}\left[\log S^{-\varepsilon}\right] \leq \log \mathrm{E}\left[L^{\varepsilon}\right] + \log \mathrm{E}\left[S^{-\varepsilon}\right].$$

For L, the maximum must be attained by one of the pairwise distances, thus

$$\begin{aligned} \mathrm{E}\left[L^{\varepsilon}\right] &\leq \mathrm{E}\left[\sum_{1\leq i<j\leq n} \|X_i - X_j\|_{\infty}^{\varepsilon}\right] \\ &\leq \binom{n}{2}\mathrm{E}\left[\|X_1 - X_2\|_{\infty}^{\varepsilon}\right]. \end{aligned}$$

By the same argument

$$\mathrm{E}\left[S^{-\varepsilon}\right] \leq (n-1)\mathrm{E}\left[\|X_i - X_j\|_{\infty}^{-\varepsilon}\right].$$

Using these inequalities to bound $\mathrm{E}\left[\log L\right] + \mathrm{E}\left[\log S^{-1}\right]$ yields

$$\mathrm{E}\left[\log L\right] + \mathrm{E}\left[\log S^{-1}\right] \in O(\log n).$$

□

If $\varepsilon \leq 1$ then in the bound for $\mathrm{E}\left[L^{\varepsilon}\right]$ in the above proof the $\binom{n}{2}$ can be replaced by $n-1$ by taking all distances to a single point and using the triangle inequality. If $\varepsilon > 1$ this remains true up to a constant (depending on ε).

Observation 2.9. *Let X_1, X_2 be independent identically distributed random variables in $\mathbb{R}^d$.*

(a) $E[\|X_1 - X_2\|_{\infty}^{\varepsilon}] \leq 2^{\varepsilon} E[\|X_1\|_{\infty}^{\varepsilon}]$.

(b) If the density function of X_1 and X_2 is bounded by a constant c then $E[\|X_1 - X_2\|_{\infty}^{-\varepsilon}] < \frac{2^d c}{d/\varepsilon - 1}$ holds for $\varepsilon < d$.

Proof. Observation (a) follows directly from the triangle inequality.

For Observation (b) we use that

$$\mathrm{E}\left[\|X_1 - X_2\|_{\infty}^{-\varepsilon}\right] \leq \max_{x\in\mathbb{R}^d} \mathrm{E}\left[d_{\infty}(x, X_1)^{-\varepsilon}\right].$$

For a point $x \in \mathbb{R}^d$

$$\mathrm{P}\left[d_{\infty}(x, X_1) < s\right] < c(2s)^d.$$

Therefore,

$$\begin{aligned} \mathrm{P}\left[d_{\infty}(x, X_1)^{-\varepsilon} > t\right] &= \mathrm{P}\left[d_{\infty}(x, X_1) > t^{-1/\varepsilon}\right] \\ &\leq 2^d c t^{-d/\varepsilon}. \end{aligned}$$

This yields

$$\begin{aligned} \mathrm{E}\left[d_{\infty}(x, X_1)^{-\varepsilon}\right] &= \int_0^{\infty} \mathrm{P}\left[d_{\infty}(x, X_1) > t^{-1/\varepsilon}\right] dt \\ &\leq 2^d c \int_0^{\infty} t^{-d/\varepsilon} dt \\ &= \frac{2^d c}{\frac{d}{\varepsilon} - 1}. \end{aligned}$$

□

Proposition 2.7 shows for many common distributions that a space-filling curve order can be computed in expected linear time. Proposition 2.8 and Observation 2.9 provide simple bounds for proving this for many distributions. We use these bounds here for the uniform distribution and normal distribution for which we will later analyze incremental constructions of Delaunay tessellation.

Corollary 2.10. *For independent, normally distributed points in $\mathbb{R}^d$ and for independent, uniformly distributed points from a bounded region in $\mathbb{R}^d$, a space-filling curve order of the points can be computed in expected linear time assuming linear time bucketing.*

Proof. For a uniformly distributed point X from a bounded region in $\mathbb{R}^d$, $\|X\|_\infty$ is bounded by the side length of a bounding cube containing the region. Thus, $\mathrm{E}\left[\|X\|_\infty^\varepsilon\right] < \infty$ for any $\varepsilon > 0$. If V is the volume of the region then the density function of X is bounded by $1/V$. Thus by Observation 2.9 the assumptions of Proposition 2.8 are fulfilled.

Let X be normally distributed in $\mathbb{R}^d$ with covariance matrix $(\sigma_{ij})_{1,\leq i,j\leq d}$. Then

$$\begin{aligned}\mathrm{E}\left[\|X\|_\infty\right] &\leq \mathrm{E}\left[\|X\|_1\right] \\ &= \sum_{i=1}^{d} \mathrm{E}\left[|X_{(i)}|\right] \\ &\leq \sum_{i=1}^{d} \sqrt{\sigma_{ii}},\end{aligned}$$

where $X_{(i)}$ denotes the ith coordinate of X. The density function of X is bounded by $\frac{1}{(2\pi)^{d/2}|\Sigma|^{-1/2}}$. Thus again the assumptions of Proposition 2.8 are fulfilled. □

In a deterministic setting with integer coordinates bounded by a polynomial in n, Proposition 2.7 directly yields a linear running time.

Corollary 2.11. *For n points in $\mathbb{R}^d$ with integer coordinates bounded by a polynomial in n, a space-filling curve order of the points can be computed in linear time assuming linear time bucketing.*

2.3 Discrete Space-Filling Curves

In this section we discuss discrete space-filling curves, i.e., indexings of a grid. In the geometric construction of space-filling curves we obtain mappings from a discrete set of intervals to a discrete set of grid cells. This motivates the definition of a *discrete space-filling curve* as a bijective map from $\{1,\ldots,n^r\}$ to $\{1,\ldots,n\}^r$. Of particular interest are discrete space-filling curves that preserve locality, either in the sense that nearby points in the linear space are mapped to nearby points in the multi-dimensional, or that the pre-image of nearby points in the multi-dimensional space are nearby in the linear space.

If $A = (a_{ij})_{i,j\in\{1,\dots,n^r\}}$ measures the distance in the linear space and $B = (b_{ij})_{i,j\in\{1,\dots,n\}^r}$ measures the distance in the multi-dimensional space, then the average locality of a discrete space-filling curve C can be expressed as

$$\frac{1}{n^{2r}} \sum_{i=1}^{n^r} \sum_{j=1}^{n^r} a_{ij} b_{C(i)C(j)}.$$

2.3.1 Optimizing Space-Filling Curves

We consider discrete space-filling curves optimized for proximity problems.

Quadratic Assignment Problem. By mapping $\{1,\dots,n\}^r$ back to $\{1,\dots,n^r\}$ by a fixed bijective map, e.g., $\iota\colon (k_1,\dots,k_r) \mapsto \sum_{j=1}^{r} k_j n^{j-1}$ the optimization problem can be formulated as a *Quadratic Assignment Problem*(QAP) [26], i.e.,

$$\textbf{Minimize} \sum_{i=1}^{n^r} \sum_{j=1}^{n^r} a_{ij} \tilde{b}_{\phi(i)\phi(j)}$$

over all permutations $\phi \in \mathcal{S}_{n^r}$, where $(\tilde{b}_{ij})_{i,j\in[n^r]}$ is defined by $\tilde{b}_{ij} := b_{\iota^{-1}(i)\ \iota^{-1}(j)}$.

Therefore, techniques for quadratic assignment problems can be used to optimize discrete space-filling curves. In the following we use algorithms from the QAPLIB[‡] [27], a quadratic assignment problem library, to optimize space-filling curves. We applied heuristics for quadratic assignment problems to two cases in two dimensions: the *average path length* $\Delta_{f(m)}$ between neighboring points weighted by the function $f(m)$ and the *expected tour length* $E(L)$ of the *Probabilistic Traveling Salesperson Problem* on a grid.

Average Path Length. If points which are close to each other in space should have similar indices, a possible quality measure is the average path length. The *path length* between two grid points is the number of grid points lying on the curve between them (including the last one). For instance the path length is one if the points are adjacent in the traversal. We consider the path length weighted by a function $f\colon \mathbb{N} \to \mathbb{R}$. The average is taken over all adjacent grid points. For the average path length $\Delta_{f(m)}$ the matrix A is given by $a_{ij} = f(|i-j|)$ and B is the adjacency matrix of the grid points.

The average path length has been considered for functions $f_q(m) := m^q$. For $q = 1$ the class of optimal curves is known [64, 103, 107]. An example of an optimal curve in shown in Figure 2.7(b). If $q \to \infty$ then the maximal distance between two neighbors dominates all other terms. For two curves with equal maximal distance the one with fewer occurrences of the maximal distance has the smaller average path length. This yields that an optimal curve for $q \to \infty$ is a *diagonal curve* [104] as shown in Figure 2.7(e). For $q \geq 1$ it is known that there is always an optimal curve that is *ordered*, i.e., for every grid cell the cells above and to the right have a larger index [103].

[‡] `http://www.seas.upenn.edu/qaplib/`

		$f(m)$			
n	**Curve**	$\log m$	m	m^2	m^3
	sim. ann.	0.8077	4.2500	30.8036	212.8304
	ant	0.8237	4.2232	**30.5893**	211.2143
8	taboo	**0.7872**	**4.2143**	**30.5893**	**211.1964**
	Hilbert	0.8206	5.0714	114.2143	4489.0179
	Lebesgue	1.0030	4.5000	51.0714	892.9286
	sim. ann.	1.5314	**7.8542**	117.3354	2061.7813
	ant	1.1918	7.9104	115.3250	1596.9063
16	taboo	1.2657	11.5813	**115.0729**	**1586.9583**
	Hilbert	**1.0051**	9.9167	852.4500	134048.9583
	Lebesgue	1.2100	8.5000	355.8333	24685.5208

Table 2.2: Average path length $\Delta_{f(m)}$ on an $n \times n$ grid.

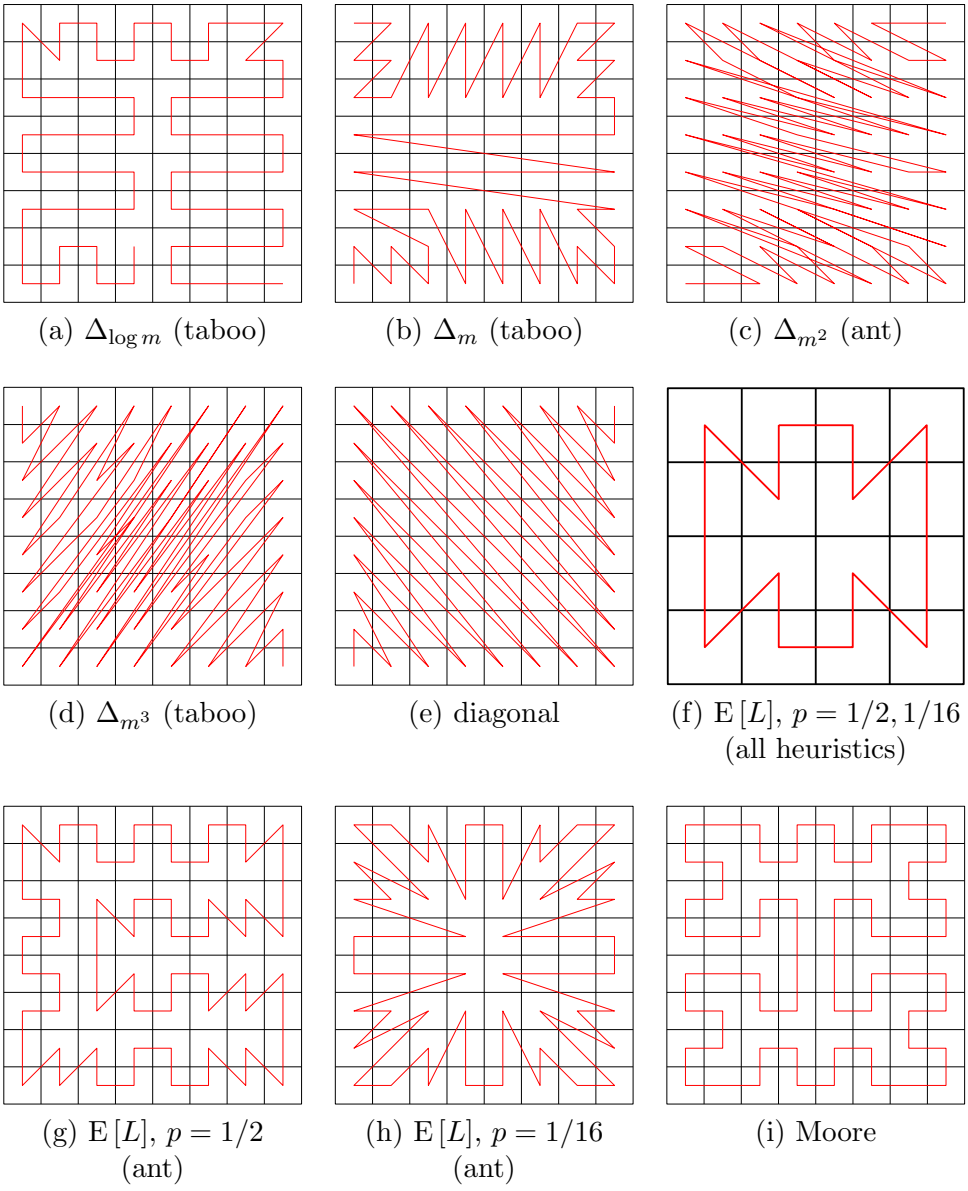
Expected Tour Length. In its most general form an instance of the probabilistic traveling salesperson problem consists of a traveling salesperson problem instance together with a random variable for each vertex. The random variables take values in $\{0, 1\}$ and determine whether a vertex must be visited or not. The problem is to find a priori an order of the vertices which minimizes the expected tour length. The probabilistic traveling salesperson problem was first considered by Jaillet [80]. See [14] for an overview of results for the case that the vertices are points in the Euclidean plane and are visited independently with a probability $p > 0$. Here we also consider the case where all vertices are visited with the same probability p but for points on a grid. For the expected tour length $E(L)$ we have $a_{ij} = p_{i\to j}$, where $p_{i\to j}$ denotes the probability that the point j is visited directly after the point i.

with the indices i is visited and the point with the index j is visited next, and B is the distance matrix of the grid points.

Experimental Results. The heuristics applied were simulated annealing [39], taboo search [140] and an ant algorithm [141]. We used the implementations of these heuristics by Éric Taillard in the QAPLIB. We compare the results of these heuristics to the discrete space-filling curves given by the recursive, geometric construction of classical space-filling curves.

The experimental results are summarized in Tables 2.3.1 and 2.3.1 and Figure 2.7. Table 2.3.1 shows experimental results for the average path length. The first column shows the side length of the grid n. The second column names the heuristic or recursive discrete space-filling curve used. The remaining columns show the average path length for different choices of f. Table 2.3.1 shows experimental results for the expected tour length. The first column again shows the side length of the grid and the second column the curve. The remaining two columns show the expected tour length for $p = 1/2, 1/16$.

Figure 2.7 shows drawings for some of the discrete space-filling curves in Tables 2.3.1 and 2.3.1 and of a diagonal ordering. Figure 2.7(a–e) shows curves for the average path length on an 8×8 grid. The curve shown in Figure 2.7(b)



(a) $\Delta_{\log m}$ (taboo) (b) Δ_m (taboo) (c) Δ_{m^2} (ant)

(d) Δ_{m^3} (taboo) (e) diagonal (f) $\mathrm{E}[L]$, $p = 1/2, 1/16$ (all heuristics)

(g) $\mathrm{E}[L]$, $p = 1/2$ (ant) (h) $\mathrm{E}[L]$, $p = 1/16$ (ant) (i) Moore

Figure 2.7: Discrete space-filling curves optimized for average path length (a–e) and expected tour length (f–i).

n	Curve	p = 1/2	p = 1/16
4	sim. ann.	**11.2745**	**1.3145**
	ant	**11.2745**	**1.3145**
	taboo	**11.2745**	**1.3145**
	Hilbert	11.7610	1.3158
	Moore	11.3611	1.31521
	Lebesgue	14.1443	1.3225
8	sim. ann.	128.9140	14.0977
	ant	**44.6264**	**13.8916**
	taboo	128.8940	14.0978
	Hilbert	49.6593	14.1945
	Moore	45.1102	14.0860
	Lebesgue	64.6978	15.0886

Table 2.3: Expected tour length on an $n \times n$ grid.

is optimal [103] for $f(m) = m$. The diagonal curve shown in Figure 2.7(e) is optimal for m^q for $q \to \infty$. For m^2 and m^3 an optimal curve is not known. For the diagonal curve the average path length for $f(m) = m^3$ is 217 for $n = 8$ and 1681.9375 for $n = 16$. Thus, the diagonal curve already compares well with the best curves found. For m^2 and m^3, Figure 2.7(c–d) show the best curves found in our experiments. The curve in Figure 2.7(c) has some similarities with the curves in Figure 2.7(b) and Figure 2.7(e). The curve in Figure 2.7(d) is already more similar to the diagonal curve.

Figure 2.7(f–i) shows curves for the expected tour length for the probabilities $p = 1/2$ and $p = 1/16$. Figure 2.7(f) shows the curve computed for a 4×4 grid. The curve has several edges of length $\sqrt{2}$, thus for $p = 1$ it is not optimal. For $p = 1$ the optimal curves are the closed curves with edges of length 1. The remaining curves are on a 8×8 grid. The curve in Figure 2.7(g) for $p = 1/2$ has only edges of length 1 and $\sqrt{2}$. For small probabilities, the best discrete space-filling curves calculated by the heuristics traverse the points in a nearly angular ordering as the curve in Figure 2.7(h). For $p \to 0$, tours through 4 points dominate the expected length since for tours through 2 and 3 points the length of the tour does not depend on the order of the points. Thus, an ordering is optimal for $p \to 0$ if it minimizes the average perimeter of a (not necessarily simple) 4-gon through 4 points.

For a comparison, Figure 2.7(i) shows a discrete counterpart to the Moore curve. As a closed space-filling curve it is more suitable for the (probabilistic) traveling salesperson problem than open curves like the Hilbert and the Lebesgue curve. In the experiments for the probabilistic traveling salesperson problem the Moore curve achieved good results (Table 2.3.1).

Eigenvalue Based Bounds. We considered bounding $E(L)$ using eigenvalue based bounds for the QAP [26] but only obtained weak bounds in this way. For $E(L)$, the coefficient matrices A, B are symmetric matrices with real entries.

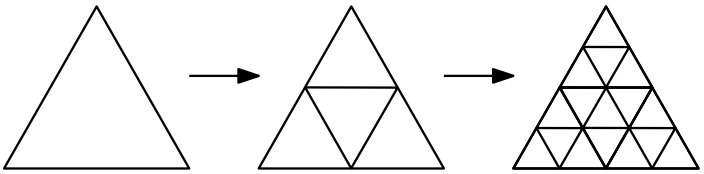
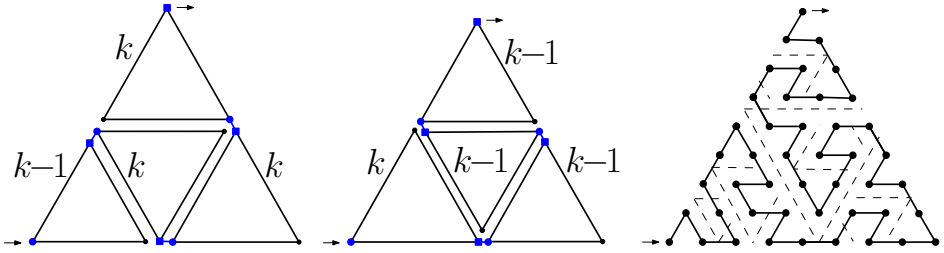


Figure 2.8: Subdivision of the triangular grid.



(a) Even side length (b) Odd side length (c) Example

Figure 2.9: Recursive subdivision.

Let $\lambda = (\lambda_1, \ldots, \lambda_n)$ denote the eigenvalues of A and $\mu = (\mu_1, \ldots, \mu_n)$ denote the eigenvalues of B. The simplest eigenvalue based bounds for the assignment problem are $\langle \lambda, \mu \rangle^-$ as lower bound and $\langle \lambda, \mu \rangle^+$ as upper bound, where $\langle \lambda, \mu \rangle^-$ and $\langle \lambda, \mu \rangle^+$ denote respectively the smallest and largest possible value of the scalar product, allowing permutation of the coordinates.

2.3.2 Triangular Discrete Space-Filling Curve

So far we considered discrete space-filling curves that are defined on a square grid. Next we present a recursive construction for a discrete space-filling curve on a regular triangular grid. An indexing of a grid also gives an indexing of the dual tiling. Thus, an application of the triangular discrete space-filling curve is indexing hexagonal tiles. A different way to index hexagonal tiles would be the use of tiling hierarchies [11].

A regular triangular grid can be obtained by successively subdividing a triangle as shown in Figure 2.8. We assume that the grid is given in this triangular form, i.e., with one vertex in the first row, two in the second, and so on.

For the recursive construction the rules are shown in Figure 2.9(a–b). The order of the discrete space-filling curve is specified by connecting all triangles and specifying for each triangle the ingoing and outgoing vertex. We subdivide into triangles of two different side lengths (measured in the number of grid points per edge). The side length also determines the subdivision procedure. There is one procedure for odd side length and one for even side length. An example is shown in Figure 2.9(c).

Conclusion

The emphasis of this chapter was on the fast computation of space-filling curve orders. Using radix sort with lookup tables yields an algorithm which runs in (expected) linear time as long as the quotient of the (expected) largest and smallest point-to-point distance can be bounded by a polynomial in the number of points. This is a weak condition as we also should by reformulating it for random point sets. The linear running time was confirmed by our experiments. This makes space-filling curve orders a good choice for spatially ordering a point set if the computation time is crucial.