Short Notes

Alternative Algorithm for Hilbert's Space-Filling Curve

ARTHUR R. BUTZ, MEMBER, IEEE

Abstract—An algorithm for generating Hilbert's space-filling curve in a byte-oriented manner is presented. In the context of one application of space-filling curves, the algorithm may be modified so that the results are correct for continua rather than for quantized spaces.

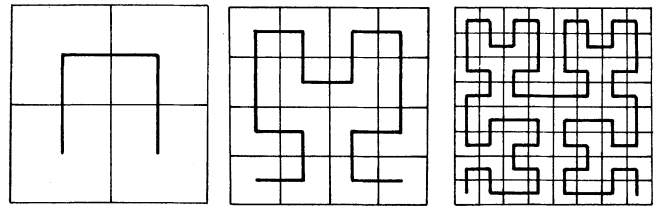
Index Terms—Algorithms, bandwidth reduction, display, mathematical programming, pattern recognition, solution of equations, space-filling curves, transforming multidimensions to one dimension.

INTRODUCTION

A space-filling curve is a mapping, $h_n: R \rightarrow U_n$, that maps the unit interval onto the n-dimensional unit hypercube continuously. It was shown in [1] that space-filling curves may be employed to converge to solutions in U_n to sets of equations or sets of inequalities. The principle of convergence is "implicitly exhaustive search." Solutions, or approximate solutions, are found by exhausting the possibilities by making the basic increments, or steps, in the convergent process in the space R rather than in the space U_n. Although the asymptotic rate of convergence is rather slow, the method has proved to be particularly effective in finding the coarse solutions which are usually required as starting points for the convergence of classical methods, such as the Newton–Raphson method.

The method should be distinguished from conventional "indexing schemes." Convergence in the usual sense is obtained; the relevant convergence theorems hold in spaces of the cardinality of the unit interval and therefore the principle of exhaustion is in no sense enumerative. The practical implication of this is that the number of steps required for convergence to a solution within a given tolerance is not dependent on any initial choice of a particular degree of quantization of the space.

Several other applications of space-filling curves have been suggested. Abend *et al.* [2] proposed using the Hilbert space-filling curve as the scan line in scanning a two-dimensional pattern representation. This appears to have been the first application of space-filling curves proposed in the literature. Patrick *et al.* [3] proposed using space-filling curves to map domains of multidimensional space onto subintervals of R for purposes of output display. Bially [4], following Laemmel's suggestion [5], discussed employing space-filling curves to achieve bandwidth reduction (n samples of a waveform may be considered a vector in U_n, and then by obtaining a point r in the inverse image, the information in the n samples is carried in one sample of a new waveform).



(a) (b) (c)

Fig. 1. Hilbert's geometric construction of a space-filling curve.

The applications of [2]–[4] have in common the necessity of choosing, *a priori*, some degree of quantization of R and U_n, and hence only approximations to space-filling curves are employed. In the first two cases there must be only a finite number of points in U_n considered in order that the reduction to one dimension take only a finite amount of time. In the case of bandwidth reduction the mapping must be invertible; this means in practice that only quantized finite-precision approximations to space-filling curves are employed.

Hilbert's space-filling curve for U_2 is the limit of the sequence of curves shown in Fig. 1. An algorithm generalizing Hilbert's curve to n dimensions is given in [1]; this is also a quantized approximation of a space-filling curve. The mapping was from a space R_1^N (quantized version of R) consisting of 2^N numbers expressed as N-bit binary numbers, where $N=mn$, onto a space R_n^m, a quantized version of U_n, with each of the n coordinates expressed as m-bit binary numbers. This mapping was called H_n^m. When $m=\infty$, R_1^N becomes the unit interval R, R_n^m the unit hypercube U_n, and H_n^m the continuous mapping h_n; h_n is not one-to-one.

Here two alternatives to the algorithm presented in [1] are described. That algorithm was designed for problems being solved on a specific machine. The machine had a 60-bit word length and it was assumed that the problem was such that $N=mn \leq 60$. On the one hand the problem may be such that $N>60$ and on the other hand the machine may have a shorter word length or be byte organized. The algorithm given in [1] assumed that $r \in R_1^N$ was being represented in one word, and in fact the initial operation on r in that algorithm strongly relied on that assumption. Here an algorithm for generating the same function is presented; however the new algorithm is more compatible with machines having shorter word lengths or byte organization. It is implicitly assumed that $r \in R_1^N$ is represented in m bytes of n bits each. This modification of the algorithm of [1] pertains to any of the proposed applications of space-filling curves.

It should be mentioned that Bially's algorithm [4] is byte oriented. However the operations required there are quite awkward compared to the operations required here and in

Manuscript received March 24, 1970; revised August 27, 1970. This research was supported in part by National Science Foundation Grant GK-1540 to Northwestern University.

The author is with the Department of Electrical Engineering, Northwestern University, Evanston, Ill. 60201.

[1]; those in [1] center around the circular shift and EXCLUSIVE-OR operations.

A second modification of the algorithm of [1] is presented here. It is shown that in the context of the methods of [1], all effects of quantization may be removed so that the n-dimensional vector that is the result of the algorithm represents the exact vector in U_n; in other words it is possible to realize h_n as well as H_n^m and no idea of quantization of the spaces is necessary. Hence this feature appears to be useful only in the context of the application of [1].

BYTE-ORIENTED ALGORITHM

The space R_n^m consists of vectors $\boldsymbol{a} \in R_n^m$; $\boldsymbol{a}$ is an n vector whose n components, $a_1, a_2, \cdots, a_n$, are expressed as m-bit binary numbers

$$a_j = .\alpha_j^1 \alpha_j^2 \cdots \alpha_j^m.$$

The space R_1^N, where $N = mn$, consists of quantities $r \in R_1^N$; r is expressed as an $N = mn$-bit binary number

$$r = .\rho_1^1 \rho_2^1 \cdots \rho_n^1 \rho_1^2 \rho_2^2 \cdots \rho_n^2 \rho_1^3 \rho_2^3 \cdots \rho_n^m.$$

The notation ρ^i is used to stand for the ith byte of r:

$$\rho^i = \rho_1^i \rho_2^i \cdots \rho_n^i.$$

As in [1], the "principal position" of ρ^i is the last position in ρ^i such that $\rho_j^i \neq \rho_n^i$ (if all bits of ρ^i are equal, the principal position is the nth). The remaining entities necessary to define the present algorithm are given in Table I.

The mapping H_n^m is defined in terms of the entities of Table I by taking $a_j = .\alpha_j^1 \alpha_j^2 \cdots \alpha_j^m$; α_j^i is the jth bit of the byte α^i. That this defines a one-to-one relationship between R_1^N and R_n^m is clear by considering the fact that the mapping is invertible; given $\boldsymbol{a} \in R_n^m$, $r \in R_1^N$ may be calculated. The definitions are illustrated in the calculation of an $\boldsymbol{a}$ vector from a given $r \in R_1^N$ in Table II.

This gives the same function as the algorithm given in [1] but the interactions among the bytes are reduced. The ith bytes are influenced by preceding bytes only through a shift and a single EXCLUSIVE-OR operation.

REALIZATION OF CONTINUOUS MAPPING

In the application of [1], it is typical that an initial value $r = 0$ is taken; in the process of converging to a solution of a system of equations, increments of the form 2^{-k}, where k is an integer, are made. Hence at any point, r is expressible in terms of a finite number of bits. Moreover, since the practical use of the method entails converging only to a neighborhood of a solution (after which some other method, such as Newton's, should be used to get an accurate solution), an upper bound may be placed on k *a priori*. Hence if we regard r as expressed in terms of an infinite number of bits, we nevertheless know that after a certain point all bits of r are zero. This permits computation of h_n. This is best shown by an example.

Suppose in the example of Table II we imagine the bits of r that are given there as being followed by an infinite string of zeros. Take the first byte of this string and use it to generate one more section of Table II:

TABLE I

ENTITIES USED IN SPACE-FILLING CURVE ALGORITHM

J_i: An integer between 1 and n equal to the subscript of the principal position of ρ^i. In the following four examples of ρ^i for the case $n = 5$, the values of J_i are 5, 2, 4, and 5, respectively (the principal positions are circled):

$$\begin{array}{ccccc} 1 & 1 & 1 & 1 & \textcircled{1} \\ 1 & \textcircled{0} & 1 & 1 & 1 \\ 0 & 0 & 1 & \textcircled{1} & 0 \\ 0 & 0 & 0 & 0 & \textcircled{0}. \end{array}$$

σ^i: A byte of n bits, such that $\sigma_1^i = \rho_1^i$, $\sigma_2^i = \rho_2^i \oplus \rho_1^i$, $\sigma_3^i = \rho_3^i \oplus \rho_2^i$, $\cdots$, $\sigma_n^i = \rho_n^i \oplus \rho_{n-1}^i$, where $\oplus$ stands for the EXCLUSIVE-OR operation.

τ^i: A byte of n bits obtained by complementing σ^i in the nth position and then, if and only if the resulting byte is of odd parity, complementing in the principal position. Hence, τ^i is always of even parity. Note that the parity of σ^i is given by the bit ρ_n^i and that a mask for performing the second complementation may be set up in the same process which calculates J_i.

$\tilde{\sigma}^i$: A byte of n bits obtained by shifting σ^i right circular a number of positions equal to

$$(J_1 - 1) + (J_2 - 1) + \cdots + (J_{i-1} - 1).$$

There is no shift in σ^1.

$\tilde{\tau}^i$: A byte of n bits obtained by shifting τ^i in exactly the same way.

ω^i: A byte of n bits where

$$\omega^i = \omega^{i-1} \oplus \tilde{\tau}^{i-1}, \qquad \omega^1 = 0\,0 \cdots 0\,0$$

and where $\oplus$ indicates the EXCLUSIVE-OR operation on corresponding bits.

α^i: A byte of n bits where $\alpha^i = \omega^i \oplus \tilde{\sigma}^i$.

TABLE II

EXAMPLE OF CALCULATION OF $\boldsymbol{a}$ FROM r

$n=5$ $m=4$ $N=20$

$r = 0.10011000100010111000$

i	1	2	3	4
ρ^i	1 0 0 1 1	0 0 0 1 0	0 0 1 0 1	1 1 0 0 0
J_i	3	4	4	2
σ^i	1 1 0 1 0	0 0 0 1 1	0 0 1 1 1	1 0 1 0 0
τ^i	1 1 0 1 1	0 0 0 0 0	0 0 1 1 0	1 1 1 0 1
$\tilde{\sigma}^i$	1 1 0 1 0	1 1 0 0 0	0 0 1 1 1	1 0 0 1 0
$\tilde{\tau}^i$	1 1 0 1 1	0 0 0 0 0	0 0 1 1 0	1 0 1 1 1
ω^i	0 0 0 0 0	1 1 0 1 1	1 1 0 1 1	1 1 1 0 1
α^i	1 1 0 1 0	0 0 0 1 1	1 1 1 0 0	0 1 1 1 1

$a_1 = 0.1010$
$a_2 = 0.1011$
$a_3 = 0.0011$
$a_4 = 0.1101$
$a_5 = 0.0101.$

i	5
ρ^i	0 0 0 0 0
J_i	5
σ^i	0 0 0 0 0
τ^i	0 0 0 0 0
$\tilde{\sigma}^i$	0 0 0 0 0
$\tilde{\tau}^i$	0 0 0 0 0
ω^i	0 1 0 1 0
α^i	0 1 0 1 0.

It is clear that all succeeding α^i in the infinite representation are identical to this one, and $x=h_5(r)$ may be calculated. In this example, x_2 and x_4 are obtained by adding 2^{-4} to a_2 and a_4, respectively, and $x_j=a_j$ for $j=1, 3$, and 5:

$$x_1 = 0.1010$$
$$x_2 = 0.1100$$
$$x_3 = 0.0011$$
$$x_4 = 0.1110$$
$$x_5 = 0.0101$$

where $x=h_5(r)$, $r\in R$, $x\in U_n$.

Conclusion

An algorithm for generating Hilbert's space-filling curve in a byte-oriented manner has been presented. In the context of one application of space-filling curves, the algorithm may be modified so that the results are correct for continua rather than for quantized spaces.

References

[1] A. R. Butz, "Convergence with Hilbert's space filling curve," *J. Comput. Sys. Sci.*, vol. 3, May 1969, pp. 128–146.
[2] K. Abend, T. J. Harley, and L. N. Kanal, "Classification of binary random patterns," *IEEE Trans. Inform. Theory*, vol. IT-11, Oct. 1965, pp. 538–544.
[3] E. A. Patrick, D. R. Anderson, and F. K. Bechtel, "Mapping multidimensional space to one dimension for computer output display," *IEEE Trans. Computers*, vol. C-17, Oct. 1968, pp. 949–953.
[4] T. Bially, "Space-filling curves: Their generation and their application to bandwidth reduction," *IEEE Trans. Inform. Theory*, vol. IT-15, Nov. 1969, pp. 658–664.
[5] A. E. Laemmel, "Dimension reducing mappings for signal compression," Microwave Res. Inst., Polytechnic Institute of Brooklyn, Brooklyn, N. Y., Rept. R-632-57, PIB 560, 1967.

A Characterization of Some Asynchronous Sequential Networks and State Assignments

LARRY L. KINNEY, MEMBER, IEEE

Abstract—An asynchronous sequential network is an interconnection of several asynchronous sequential circuits. Asynchronous networks are classified according to the relative time delay of the individual circuit's feedback signals and input signals. They are also classified by the manner in which the circuits interact. In addition, another network model uses duplicate paths for the feedback signals and the input signals of a circuit where the duplicate paths have differing delays. The partition theory of Hartmanis and Stearns [5] is used for characterizing some of the network types in a manner analogous to the characterization of synchronous sequential networks. The special case where each circuit in the network has only two states is equivalent to specifying a state assignment for the overall network and this leads to a characterization of some state assignments for asynchronous circuits. In particular, any arbitrary state assignment for the network which has duplicate paths and delays is shown to be free of critical races, and another network model leads to a variation of the minimum transition-time state assignments which in certain cases may require fewer state variables.

Index Terms—Asynchronous circuits, decompositions, partition theory, state assignments.

Manuscript received September 12, 1969; revised August 1, 1970. A portion of the work in this note was presented at the IEEE Ninth Annual Symposium on Switching and Automatic Theory, October 1968. A portion of this work was supported by the National Science Foundation under Grant GK 4835.

The author is with the Department of Electrical Engineering, University of Minnesota, Minneapolis, Minn. 55455.

I. Introduction

Previous work has considered the problem of realizing an asynchronous machine (or flow table) by a network of asynchronous circuits consisting of a serial, parallel, or a combination of serial and parallel connections of asynchronous circuits [1]–[4]. These realizations have been called loop-free decompositions by Hartmanis and Stearns [5]. This note is also concerned with the decomposition of asynchronous circuits, but the decompositions will not be restricted to loop-free decompositions, i.e., networks of asynchronous circuits will be considered where the pattern of interconnections is arbitrary. As a special case of this general decomposition problem, if each asynchronous circuit within the network of circuits has only two states, then the problem of realizing an asynchronous machine by a network of this type is equivalent to the state assignment problem, i.e., the problem of finding a valid state assignment for the asynchronous machine.

The methods and results of this paper rely heavily on the partition theory developed by Hartmanis and Stearns [5] for synchronous sequential circuits. The more general case of using set systems (covers) is not considered. Since only partitions are used, the state assignments which result are those that assign a single binary code to each state of the asynchronous machine (unicode assignments).

It will be assumed that the reader is somewhat familiar with asynchronous sequential circuit theory and with partition algebra. The next section presents the formal definitions related to the partition algebra and some terminology which are used later in the paper. Section III discusses the difficulties which are encountered in applying the partition theory to asynchronous circuits. Section IV outlines some network types by the timing relationships between changing signals within the network and by the manner in which the circuits interact with each other. Lastly, Sections V, VI, and VII contain the theorems which characterize some of the network types. In each case, the results are specialized so that they apply to state assignments.

II. Definitions and Terminology

We will begin with a definition of a sequential machine.

Definition 1: A sequential machine is a 5-tuple $M=(S, I, O, f, g)$ where

1) S is a finite nonempty set called the states;
2) I is a finite nonempty set called (external) inputs;
3) O is a finite nonempty set called the outputs;
4) f is a next-state mapping from the Cartesian product of S and I into S, $f: S\times I\rightarrow S$; and
5) g is a output mapping from the Cartesian product of S and I into O, $g: S\times I\rightarrow O$.

If, for each s in S and each x in I, $f(f(s, x), x)=f(s, x)$, then M is an asynchronous sequential machine. It will be assumed that f and g are defined for each element in $S\times I$