Design Sequences: Meandering Strings

Given j characters, such as a, b, c, …, a *j-k-meandering string*, or simply *j-k-meander*, contains all substrings of j characters of length k [1]. Meandering strings are considered to wrap around from end to beginning for the purpose of representing substrings.

For example, the length-2 strings composed from the 3 characters a, b, and c are aa, ab, ac, ba, bb, bc, ca, cb, and cc. A 3-2-meander that contains all these strings as substrings is $aabacbbcc$. The substring ca comes from the last character of the meander followed by the first.

For j characters, it can be shown that:

- A j-k-meander must contain at least j^k characters.
- It is always possible to construct a *minimal* j-k-meander that contains exactly j^k characters.
- There is a straightforward way of constructing minimal j-k-meanders.

Meanders become long as j and k increase. Here is a table of some values:

j	k	*length*
2	2	4
2	3	8
2	4	16
2	5	32
3	2	9
3	3	27
3	4	81
3	5	243
4	2	16
4	3	64
4	4	256
4	5	1024
5	2	25
5	3	125
5	4	625
5	5	3125

Design Uses

Meanders can be used in design in a variety of ways. In this regard, the value of j determines the number of design objects, while, somewhat in the manner of Dietz polynomials [2], the value of k corresponds to the degree of interaction among the objects.

Note that it doesn't matter what characters are used. They can be interpreted in a variety of ways.

One interpretation of meanders is as sequences of blocks for profile drafting.

Another interpretation of meanders is as sequences of colors, which can be used, for example, for stripes. In this interpretation, the characters of the meander are mapped into colors from a palette. For example, p might stand for peach, w for white, and b for sky blue.

Another possible interpretation of the characters is as widths. For example, *3* might stand for 3 threads, *8* for 8 threads, and *5* for 5 threads.

There are many other possibilities. For example, characters in odd-numbered positions might stand for colors and characters in even-numbered positions might stand for corresponding widths.

The possibilities are limited only by your ingenuity.

Some examples based on some of the ideas given above are shown on the last page of this article.

Constructing Meanders

The process of constructing minimal meanders is relatively simple, although for long ones it is tedious and error-prone if done by hand.

1. Put the characters in some order. The order does not matter except as it affects the details of the result.

2. Start with a string of k–1 copies of the first character.

3. Append the last character to the current string, provided it does not produce a duplicate k-length substring (wrap-around doesn't apply here). If the last character would pro-

duce a duplicate, try the next-to-last, and so on, until one works. If no character works, go to Step 4. Otherwise repeat Step 3.

4. Remove the starting string. The result is a minimal *j*-*k*-meander.

As an example, suppose $j = 3$, $k = 2$, and the three characters are *a*, *b*, and *c*. Suppose we take them in this order.

Then the starting string is *a*.

The first application of Step 3 appends *c* to this result, giving *ac*. Repeating Step 3, the string becomes *acc*.

So far, so good. However, in trying Step 3 again, appending *c* would produce *accc*. But now we have two (albeit overlapping) instances of *cc*, so this won't do. Following the instructions, we try the next-to-last character, *b*, and get *accb*.

Now when we do Step 3 again, we can append the last character, *c*, to give *accbc*.

We can't append *c* again, and we can't append *b* either, since that would produce a duplicate also. We're left with appending *a*, giving *accbca*.

Starting Step 3 over, we can't append *c*, since there already is an *ac*, but we can append *b*, giving *accbcab*.

Trying Step 3 once again, we can't append *c*, but we can append *b*, giving *accbcabb*.

And we are at Step 3 again. We can't append *c*, since there already is a *bc*. We can't append *b* either, since there already is a *bb*. We can, however, append *a*, giving *accbcabba*.

And Step 3 another time. We can't append *c* or *b*, but we can append *a*, giving *accbcabbaa.*

When we try Step 3 again, nothing works, since there already are substrings *ac*, *ab*, and *aa*.

We're almost done; on to Step 4. All we have to do is remove the starting string *a*, giving *ccbcabbaa*. That is the result.

Note how the order in which the characters are placed affects their order in the result.

All this detail seems tedious, but it practice it goes quickly, at least for small *j* and *k*.

If you're going to write a program to do this, note that while finding duplicate substrings is easy for human beings (if *k* is small), it's a bit tricky to formulate in some programming languages.

References

1. *Minimal Meandering Strings*, James F. Gimpel and William Keister: http://www.cs.arizona.edu/patterns/weaving/seqdocs/meander.pdf

2. *Design Inspirations from Multivariate Polynomials*, Ralph E. Griswold, 2001: http://www.cs.arizona.edu/patterns/weaving/webdocs/gre_pol1.pdf

Ralph E. Griswold
Department of Computer Science
The University of Arizona
Tucson, Arizona

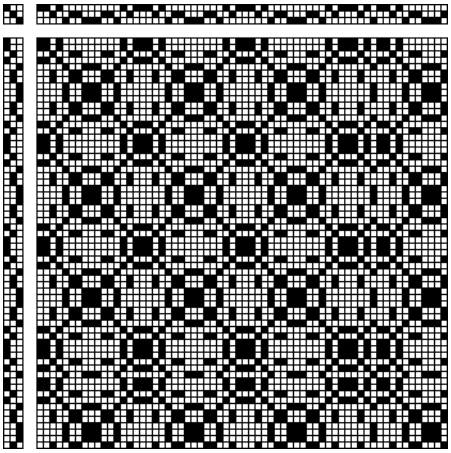

Figure 1. Profile Draft for Mirrored 3-2-Meander

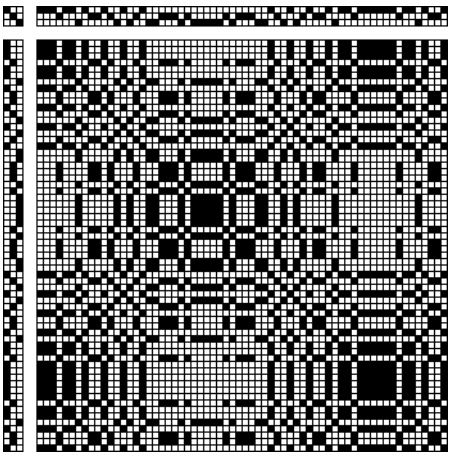

Figure 2. Profile Draft for Mirrored 3-3-Meander

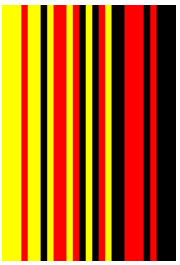

Figure 3. Stripe Colors from 3-3-Meander

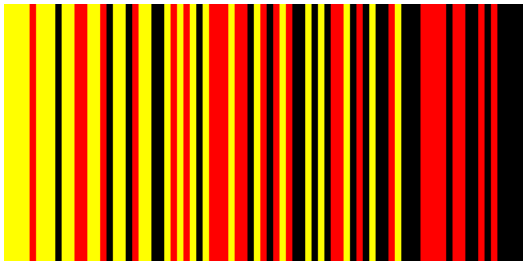

Figure 4. Stripe Colors from 3-4-Meander

Figure 5. Stripe Colors from 4-3-Meander