Change ringing: exploring the method space

What is this and who are you

This article takes a look at exploration of all possible methods on various stages, via generation of place notation. In particular, it touches on

- what makes a method desirable or undesirable
- how many methods are there for various stages and constraints
- how we can generate methods by enumeration of place notation

We delve into the maths of method enumeration, and identify some of the concepts in mathematics which method enumeration neatly maps onto.

If you're not a ringer or a mathematician, this article might not particularly knead your dough.

Glossary

In this article, when I mention 'method' I'm referring to place notation (PN) which represents a lead (or a six or block etc) of a method. I don't address calls or compositions in this article.

The term 'place notation' refers to a string like `x.14.x.16`. There's no standard term for a single row's information inside the PN, so I'll use the term *notate*. So `x`, `14`, and `16` are notates inside the PN `x.14.x.16`.

Standard place notation

In this article I use only *standard place notation*, by which I mean:

- use only `'x'` to mean 'all swap'
- every notate is separated by `'.'`, even if you have an `'x'` on one side (so you'd write `'x.16'`, rather than `x16`)
- every place made for a notate is explicit. So on stage 6, the notate `1` is avoided: we write `16` instead. And on odd stages, you'd never write `'x'`.

Method in your madness

Well-formed methods

When generating methods, if we find some PN is not *well-formed*, we're not interested in considering it and should discount it as soon as possible.

A method's place notation is **not** well-formed for any of these reasons[1]:

- its PN is repetitive: it is made from a smaller piece of PN repeated one or more times.
- it contains a notate repeated immediately[2], e.g. `x.x`. This makes the method trivially false.

So to illustrate:

Whereas `x.12.x.16` is a well-formed method, `x.14.x.14` is not, since it is `x.14` repeated once.

`x.x.14.18` isn't well formed because it contains an immediate repeat of `x`.

`12.58.x.12` isn't well formed because it contains an immediate repeat of `12` at the wrap-around.

Equivelant methods

If we generate all possible methods for some given stage, we'll find that although they're all distinct in terms of place notation, a lot of them will be the same method for all intents and purposes. This is usually because one or more of the following are true:

1. one method has the reverse place notation of the other (example: `x.12.34.16` and `16.34.12.x`)
2. one method has place notation which is just a rotation of another (example: `x.12.34.16` and `34.16.x.12`)

When any of the above are true for two methods, we say they are *equivalent* methods. We consider them to be duplicates of each other, and we are usually not interested in counting duplicates when generating methods. But which of the duplicates do we keep? See the next section.

Canonical form of a method

In order to help with removing duplicate methods during method generation, it's useful to have a way to convert a method's place notation M to a *canonical form*[3] of place notation M_c. We want the canonical form to meet these conditions:

1. M and M_c are *equivelant*
2. any, and only, methods equivalent to M will also have canonical form M_c

An algorithm for finding the canonical form

Suppose we have generated a method Q using a list of m place notations selected from a dictionary $\{p_0, p_1, ...p_{m-1}\}$.

Our generated method's PN can be written down as a list of indexes into the dictionary, for example: $(0, 2, 0, 7, ...)$.

We can convert such an index list into an integer (call it the 'measure') by considering it to be a string of digits base in L.

Now, convert *all possible rotations* of our place notation index list and *all possible rotations of its reversal*[4] into measures.

The canonical form of our method is the place notation corresponding to the the largest measure.

Note that if we find multiple (equal) largest measures, the place notation is repetitive, and so is not a well formed method.

---

[1] And by well-formed I don't mean any higher-level judgements like "this looks like a nice method to ring" or "it is symmetrical" or "it is in the plain bob lead head group" etc.

[2] Don't forget to consider the wrap-around point (first and last parts of the PN) when looking for immediately repeated notates

[3] Algorithmic aside: canonical form is useful because it reduces runtime complexity in comparing methods to each other, e.g. de-duplicating a collection of methods

[4] Hint: for methods with traditional half-lead symmetry, reversed place notation is equivelant to the original place notation, so there's no need to do the *all possible rotations of its reversal* part for those methods

Worked example of finding canonical form

Suppose we're considering minimus methods. Our dictionary of place notations on stage 4 is $\{$"x", "12", "14", "34"$\}$, and the method we want to convert to canonical form is Plain Bob Minimus, with PN being `x.14.x.14.x.14.x.12`.

For each rotation of our PN and its reversal, we calculate the measure:

place notation	as indexes into dictionary	as measure integer
`x.14.x.14.x.14.x.12`	$(0, 2, 0, 2, 0, 2, 0, 1)$	$02020201_4 = 8737$
`14.x.14.x.14.x.12.x`	$(2, 0, 2, 0, 2, 0, 1, 0)$	$20202010_4 = 34948$
`x.14.x.14.x.12.x.14`	$(0, 2, 0, 2, 0, 1, 0, 2)$	$02020102_4 = 8722$
`14.x.14.x.12.x.14.x`	$(2, 0, 2, 0, 1, 0, 2, 0)$	$20201020_4 = 34888$
`x.14.x.12.x.14.x.14`	$(0, 2, 0, 1, 0, 2, 0, 2)$	$02010202_4 = 8482$
`14.x.12.x.14.x.14.x`	$(2, 0, 1, 0, 2, 0, 2, 0)$	$20102020_4 = 33928$
`x.12.x.14.x.14.x.14`	$(0, 1, 0, 2, 0, 2, 0, 2)$	$01020202_4 = 4642$
`12.x.14.x.14.x.14.x`	$(1, 0, 2, 0, 2, 0, 2, 0)$	$10202020_4 = 18568$

The largest measure here is 34948, in the second row. Therefore the place notation for that row is the canonical form of Plain Bob Minimus: `14.x.14.x.14.x.12.x`.

You might rightly balk at the idea of a method beginning with the treble making a place! Canonical form is not there to look pretty, particularly, but if there's an easy way to make it produce more palatable PN, we'll take it. Happily, there's an easy way: notice that items towards the end of the place notation dictionary tend to appear at the start of the canonical place notation produced. So we can tweak our PN dictionary to instead be $\{$"12", "14", "34", "x"$\}$. This results in our canonical PN for Plain Bob Minimus being the usual PN: `x.14.x.14.x.14.x.12`.

Counting and enumerating possible place notation for a stage

If we want to try to count how many possible methods there are of various types (plain, treble bob, etc) on different stages, we start with a fundamental question:

How many distinct place notations are there for a row on stage n?

Let's start with an example, a list of all possible place notation for stage 4:

Table 2: All possible place notations for stage 4

Schematic	Place notation
`1234`	
`><><`	`x`
`><||`	`34`
`|><|`	`14`
`||><`	`12`
`||||`	`1234`

Note that we use `><` to represent two bells crossing: we use two characters because of course two bells are involved. The `|` character represents a bell remaining in place.

This is looking like a combinatorial problem – given a items, choose b of them – but there's a catch here: the `><` occupies two characters, but it represents a single thing conceptually: two bells crossing. To add some clarity, let's rewrite our table, but using `x` instead of `><`:

Table 3: Altered schematic for all possible PN for stage 4

Schematic	Place notation
`xx`	`x`
`x||`	`34`
`|x|`	`14`
`||x`	`12`
`||||`	`1234`

Since the string for each code above contains a single character – "`x`" or "`|`" – for each possibility, it's now more amenable to the *a choose b* combinatorial approach. But note that we have strings of varying lengths.

With a little thought you can see that the length of each string is n minus the count of x in the string. And from there we can break this down into ${}_n\mathrm{C}_r$ expressions for combinations of `x` in the string:

Code	string length	x count	Combinations for x occuring
`xx`	2	2	${}_2\mathrm{C}_2 = 1$
`x||`	3	1	]
`|x|`	3	1	] ${}_3\mathrm{C}_1 = 3$
`||x`	3	1	]
`||||`	4	0	${}_4\mathrm{C}_0 = 1$

Let's use $\mathbb{P}(n)$ to denote 'number of possible place notations on stage n'. In this case we have:

$$\mathbb{P}(4) = {}_2\mathrm{C}_2 + {}_3\mathrm{C}_1 + {}_4\mathrm{C}_0 = 5$$

If you analyse stage 5, you get:

$$\mathbb{P}(5) = {}_3\mathrm{C}_2 + {}_4\mathrm{C}_1 + {}_5\mathrm{C}_0 = 5$$

And for stage 6:

$$\mathbb{P}(6) = {}_3\mathrm{C}_3 + {}_4\mathrm{C}_2 + {}_5\mathrm{C}_1 + {}_6\mathrm{C}_0 = 8$$

There's a pattern emerging. The equation for any stage is:

$$\mathbb{P}(n) = \sum_{i=0}^{\lfloor n/2 \rfloor} {}_{n-i}\mathrm{C}_i \tag{1}$$

If you look at the first few values for the $\mathbb{P}$ function, you'll see a familiar sequence emerging:

n	0	1	2	3	4	5	6	7	8	9
$\mathbb{P}(n)$	1	1	2	3	5	8	13	21	34	55

It's the Fibonacci sequence[5]. So we now have a tidy definition for $\mathbb{P}$:

$$\mathbb{P}(n) = fib(n+1) \tag{2}$$

See Appendix A for an aside on Fibonacci in Pascal's Triangle.

No-constraint method with plain lead length

This is a method of lead length $2n$ with no other restrictions (the treble can be doing anything and the transposition at the end of the block can be anything).

If we can choose any possible place notation for each row, we calcalate the number of combinations as $\mathbb{P}_n^{2n}$. However, that allows for immediately repeated notates. If we disallow those, the number of combinations is $(\mathbb{P}_n - 1)^{2n}$

Preventing repetetive pn:

if n = 3, we have 6 rows, 3 choices for each (incl |||).

---

ways to get repeats: of len 3:

ABCABC

- so equal to ways of choosing 3 in set way, which dictate the last 3.

of len 2:

ABABAB

- so equal to ways of choosing 2 in set way, which dictate the last 4.

Say

Plain hunt methods

Table 6: Plain Hunt Major combinations schematic

Schematic	Combos
`12345678`	
`><......`	$\mathbb{P}(6)$ or $\mathbb{S}(0, 6)$
`|><.....`	$\mathbb{P}(5)$ or $\mathbb{S}(1, 5)$
`..><....`	$\mathbb{S}(2, 4)$
`...><...`	$\mathbb{S}(3, 3)$

[5] Proofs are available, e.g. by induction, that the given combinatorial sum in (1) generates the Fibonacci sequence

<table>
<tr><th>Schematic</th><th>Combos</th></tr>
<tr><td><code>....><..</code></td><td>$\mathbb{S}(4,2)$</td></tr>
<tr><td><code>.....><|</code></td><td>$\mathbb{P}(5)$ or $\mathbb{S}(5,1)$</td></tr>
<tr><td><code>......><</code></td><td>$\mathbb{P}(6)$ or $\mathbb{S}(6,0)$</td></tr>
<tr><td><code>.......|</code></td><td>$\mathbb{P}(7)-1$</td></tr>
<tr><td><code>   ><|</code></td><td></td></tr>
<tr><td><code>   ><</code></td><td></td></tr>
<tr><td><code>   ><</code></td><td></td></tr>
<tr><td><code> ><</code></td><td></td></tr>
<tr><td><code>|><</code></td><td></td></tr>
<tr><td><code>><</code></td><td></td></tr>
<tr><td><code>|.......</code></td><td>$\mathbb{P}(7)-1$</td></tr>
</table>

Table 7: Plain Hunt Caters combinations schematic

<table>
<tr><th>Schematic</th><th>Combos</th></tr>
<tr><td><code>123456789</code></td><td></td></tr>
<tr><td><code>><.......</code></td><td>$\mathbb{P}(7)$ or $\mathbb{S}(0,7)$</td></tr>
<tr><td><code>|><......</code></td><td>$\mathbb{P}(6)$ or $\mathbb{S}(1,6)$</td></tr>
<tr><td><code>..><.....</code></td><td>$\mathbb{S}(2,5)$</td></tr>
<tr><td><code>...><....</code></td><td>$\mathbb{S}(3,4)$</td></tr>
<tr><td><code>....><...</code></td><td>$\mathbb{S}(4,3)$ or $\mathbb{S}(3,4)$</td></tr>
<tr><td><code>.....><..</code></td><td>$\mathbb{S}(5,2)$ or $\mathbb{S}(2,5)$</td></tr>
<tr><td><code>......><|</code></td><td>$\mathbb{P}(6)$ or $\mathbb{S}(6,1)$</td></tr>
<tr><td><code>.......><</code></td><td>$\mathbb{P}(7)$ or $\mathbb{S}(7,0)$</td></tr>
<tr><td><code>........|</code></td><td>$\mathbb{P}(8)-1$</td></tr>
<tr><td>$\vdots$</td><td></td></tr>
<tr><td><code>|........</code></td><td>$\mathbb{P}(8)-1$</td></tr>
</table>

Table 8: Plain Hunt Royal combinations schematic

<table>
<tr><th>Schematic</th><th>Combos</th></tr>
<tr><td><code>123456789E</code></td><td></td></tr>
<tr><td><code>><........</code></td><td>$\mathbb{P}(8)$</td></tr>
<tr><td><code>|><.......</code></td><td>$\mathbb{P}(7)$</td></tr>
<tr><td><code>..><......</code></td><td>$\mathbb{S}(2,6)$</td></tr>
<tr><td><code>...><.....</code></td><td>$\mathbb{S}(3,5)$</td></tr>
<tr><td><code>....><....</code></td><td>$\mathbb{S}(4,4)$</td></tr>
<tr><td><code>.....><...</code></td><td>$\mathbb{S}(5,3)$ or $\mathbb{S}(3,5)$</td></tr>
<tr><td><code>......><..</code></td><td>$\mathbb{S}(6,2)$ or $\mathbb{S}(2,6)$</td></tr>
<tr><td><code>.......><|</code></td><td>$\mathbb{P}(7)$</td></tr>
<tr><td><code>........><</code></td><td>$\mathbb{P}(8)$</td></tr>
<tr><td><code>.........|</code></td><td>$\mathbb{P}(9)-1$</td></tr>
<tr><td>$\vdots$</td><td></td></tr>
<tr><td><code>|.........</code></td><td>$\mathbb{P}(9)-1$</td></tr>
</table>

$$\mathbb{C}_p(n) = \prod_{i=2}^{n-4} fib(i)$$

scratch

Fibonacci factorial ('fibofac') definition

Method type	Denoted	Definition
Plain Double	$\mathbb{C}_{pd}(n)$	$ff(n)$
Plain	$\mathbb{C}_{p}(n)$	$ff(n)^2$
Treble Bob Double	$\mathbb{C}_{td}(n)$	$ff(n)^4$
Treble Bob	$\mathbb{C}_{t}(n)$	$ff(n)^8$

$$ff(n) = \prod_{i=0}^{n} fib(i)$$

$$\dot{a} = fib(a)$$

$\binom{n}{x}$

Split parity paradox

${}_n\mathrm{C}_{r-1} \cdot b$

${}_n\mathrm{C}_r - 1$

$\ddot{8} \cdot \dot{7}!$

$ff(7)$

$f f(7)$

rest

divisor function: $\sigma_x(n) = \sum_{d|n} d^x$

https://oeis.org/A051137

Non-reversed and non-rotated.

$$T(n,k) = \frac{k^{\lfloor (n+1)/2 \rfloor} + k^{\lceil (n+1)/2 \rceil}}{4} + \frac{\sum_{d|n} \phi(d) \cdot k^{n/d}}{2n}$$

```
T(n, k) = (k^floor((n+1)/2) + k^ceiling((n+1)/2)) / 4 + (1/2n) * Sum_{d divides n} phi(d) * k^(n/d)
```

Appendix A: Fibonacci and Pascal's Triangle

We know that $\mathbb{P}(n)$ is the sum of ${}_n\mathrm{C}_r$ combinatorial terms, and that these terms can be read off Pascal's Triangle.

This means that if you view Pascal's triangle left-aligned, the Fibonacci numbers (and hence $\mathbb{P}(n)$) appear as the sum of diagonals running SW-NE:

1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7	21	35	35	21	7	1

If you skew the triangle rows more to the right, the pattern is even more obvious:

1							
	1	1					
		1	2	1			
			1	3	3	1	
				1	4	6	4
					1	5	10
						1	6
							1