The Longest Prime Siteswap Patterns

by Jack Boyce

with many thanks to Johannes Waldmann

This document assumes a familiarity with basic siteswap notation, and is a discussion of some of the more mathematical aspects of these patterns. A good introduction to siteswap notation is the JIS siteswap page.

Contents

Prime Patterns

A **prime** siteswap is one which contains no repeatable subsequences. For example, the pattern `423` isn't prime since the subsequence `'3'` can be repeated indefinitely within the pattern:

```
          ... 4234234233333333423423 ...
```

An important fact is that if you restrict yourself to some maximum throw value, then there are a finite number of prime patterns for a given number of balls (below we will see why). For example, with 3 balls if you restrict yourself to throw values no greater than 5, the exhaustive listing of prime patterns is:

```
            3
            42
         4  51  2
            441
            522
            531
         4  450  2
            4440
            4530
            5241
```

```
           5340
           5511
           5520
           45501
           52440
           52530
           53502
           55140
           55500
        4  55050  2
           455040
           525501
           551502
           5255040
           5350530
           55150530
```

where the excited-state ones are shown with starting and ending throws.

The main point is that any 3 ball, height 5 siteswap you write down can be decomposed into combinations of these patterns (try it). In this sense the prime patterns are like the prime numbers, however here there are a finite number (restricting yourself to some maximum throw). In particular there is a longest one, and that is in fact this document's main focus of study.

Juggling States and State Graphs

The easy way to tell whether a pattern is prime is to look at the juggling states that it traverses. A **state** is a record of the times in the future when the balls are going to land (think of it as the rhythm of the balls hitting your hands if you suddenly stopped juggling). A thorough discussion of juggling states and their associated graphs is given in Allen Knutson's Siteswap FAQ.

The states traversed by `423` are:

```
                xxx-
        --4-->  xx-x
        --2-->  xxx-
        --3-->  xxx-
```

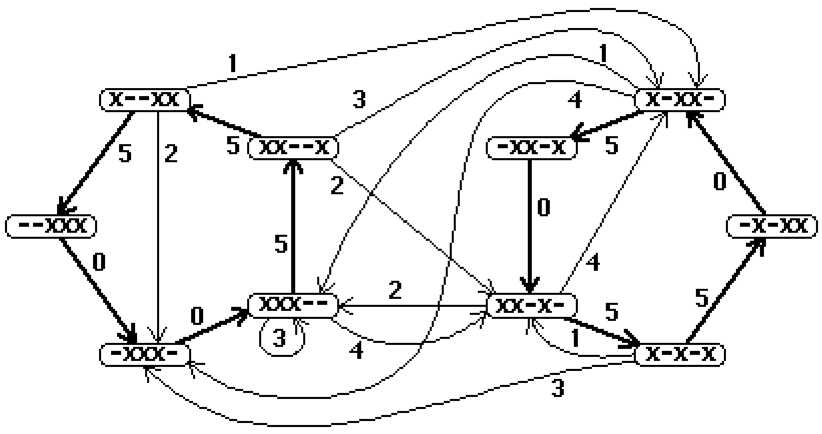
and we can immediately tell that: (1) `423` is a valid siteswap, since it returns to its initial state, and (2) it isn't prime, since the state `xxx-` is repeated (and the subsequences `42` and `3` are repeatable).

The virtue of the juggling state concept is that there are simple rules telling us how the state evolves from one throw to the next, according to what throw is made:

1. The entire state shifts to the left on each beat. The leftmost entry is removed, and a `'-'` shifts on from the right.
2. If a `'x'` shifted off the left, we must make a nonzero throw `T`. A throw `T` is allowed if and only if the shifted state has a `'-'` at position `T`; position `T` is converted to `'x'` and

we have our new state.
3. If a `'-'` shifted off the left, we must throw a `0` (no throw) for the current beat.

For some number of balls `b` and maximum throw value `h` the number of possible states is just `(h` choose `b)`, the number of distinct ways of arranging the `'x'`s in the state. We form the **state graph** `(b,h)` by writing down all possible states, and then drawing an arrow from `S1` to `S2` if there is a an allowed single-beat transition (via the rule above) taking us from `S1` to `S2`. These arrows are labeled by the throw values made. Shown below is the state graph `(3,5)`.



Once the graph is drawn, it's very easy to find patterns -- we just start at some state and follow the arrows around the graph until we return to our starting point. We then have a repeatable pattern since we could continue repeating the same circuit through the graph. If a pattern goes through the unique state with all the `'x'`s on the left (`xxx--` in the figure above) then it is called **ground state**; otherwise it is called **excited state**. Finding transitions between patterns is just a matter of finding a path through the graph from some state on pattern 1 to some state on pattern 2.

For our present purposes, though, the most important is the following: **A prime pattern is one whose circuit in the graph touches upon no state more than once.** In the language of graph theory, a prime pattern is a **cycle** in the graph. It is now obvious that no prime pattern can be longer than `(h` choose `b)`, the total number of vertices in the graph. Below we will find an even stronger upper bound on the lengths of prime patterns.

Block Form

It is an empirical fact, first noted by Johannes Waldmann, that very long prime patterns are comprised mostly of throws `0` and `h`. Johannes also noted that in very long prime patterns, the `0` and `h` throws commonly occur in groupings of length `h-2`. Using these observations he wrote a very efficient program to generate long prime patterns, and conjectured that they were maximally-long (in almost all cases he was correct). Later we will understand why this form is so prevalent in long prime patterns.

An asynch siteswap in `(b,h)` is in **block form** when it is composed of one or more **blocks**.

Each block consists of:

1. at most `h-2` throws (called the **shift throws**), each of value `0` or `h`, followed by
2. a throw (called the **link throw**) of value greater than `0` but less than `h`.

The **block size** is the total number of throws per block, at most `h-1`. It is convenient when writing block patterns to use `'-'` for `0` and `'+'` for `h`, writing only the link throws explicitly. Thus the `(4,7)` pattern `0077717007730770717077060707 74` in the list below is written `--+++1+--++3-++-+1+-++-6-+-++4`. (It is also conventional to write ground-state patterns so that they start from the ground state, so this pattern is actually printed as `+++1+--++3-++-+1+-++-6-+-++4--`.)

Why do we restrict ourselves to no more than `h-2` shift throws in a row? It's a straightforward exercise to prove the following

> **Theorem:** Let `P` be a prime pattern in `(b,h)` of length > `h`. Then `P` contains no runs of `+/-` throws longer than `h-2`.

Consequently every prime pattern longer than `h` will be in block form, as defined.

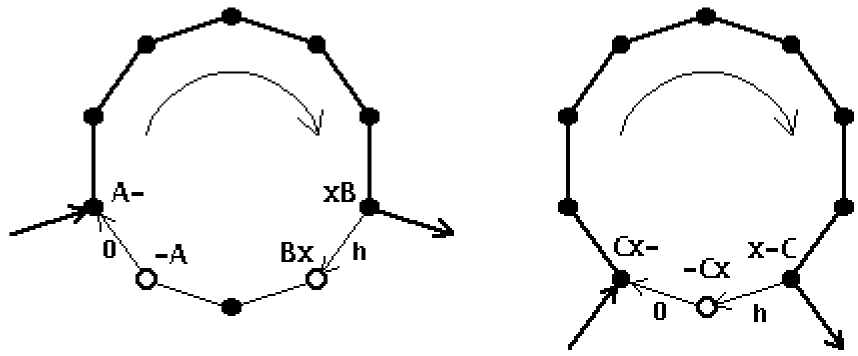
Shift Cycles

We now turn our attention toward understanding the structure of the juggling graph `(b,h)`. This will explain the empirical facts noted above and will also provide powerful means of calculating long prime patterns.

Let `S` be a state in the graph. Then it is always the case that `S` has either a `0` or `h` throw leading out of it (the contents of the leftmost position in the state determines which). If we follow this arrow, we get to a new state which is just `S` *shifted left* by one position. After no more than `h` of these `+/-` throws we will return to `S`, and in this way we form a prime pattern (called a **shift cycle**) containing `S` and its distinct rotations. Find the shift cycles in the `(3,5)` graph drawn above (they are the heavy arrows).

It is clear that each state in `(b,h)` is in exactly one shift cycle. The shift cycles are not always of length `h`; for example, `x-x-x-` in `(3,6)` lies on a shift cycle of length 2. When `b` and `h` are coprime all shift cycles are length `h`; otherwise there are shorter shift cycles. It is a relatively straightforward exercise to calculate how many shift cycles a particular graph has, and of what lengths.

Now consider what happens when we make a link throw (not `0` or `h`), the only way to switch from one shift cycle to another. On the left side of the figure below we illustrate part of a prime pattern which link throws onto a shift cycle, follows it with `+/-` throws, and then link throws off.



The pattern must enter the shift cycle on a state of the form `A-` (`A` is some `x/-` sequence of length `h-1`), since states `Ax` can only be entered with a `+` throw. The state immediately prior to state `A-` in the shift cycle is `-A`. But `-A` has only a single throw out of it, 0. This implies that:

1. State `-A` has not been visited by the pattern yet. If it had then `A-` would also have been already visited, contradicting our assumption that this is a prime pattern.
2. State `-A` will not be visited later in the pattern. If it were, the only way to get out is through the (already visited) state `A-`.

Taken together, these imply that **a link throw onto state `S` forces the next state "upstream" in `S`'s shift cycle to be missing from the pattern.** This is indicated in the figure by the open circle for state `-A`.

A similar logic applies to link throws off a shift cycle, which can only occur at states of the form `xB`. The next state in the shift cycle is `Bx`, which is only entered with a `+` throw from our exit state `xB`. Clearly state `Bx` must also be missing from the (prime) pattern. We exclude a single state overall, rather than two, by combining these two missed states as shown on the right side of the figure above.

Complete and Incomplete Prime Patterns

We can now understand why maximal patterns are usually composed of long blocks: each link throw excludes at least one state, so the longest patterns will tend to stay on a shift cycle as long as possible before link throwing to another cycle. It is also clear from our discussion above that a prime pattern must miss at least one state on each shift cycle, yielding the following upper bound on the length of a prime pattern (this was discovered by Johannes):

L_{bound} = (Total # of States) - (Total # of Shift Cycles)

Prime patterns that satisfy the upper bound ($L=L_{bound}$) are called **complete** patterns, and are missing exactly one state from each shift cycle. In the next section we will discover that these missing states have an interesting relation to one another.

Prime patterns shorter than L_{bound} are called **incomplete**, and at least one shift cycle is missing more than one state. It is useful to classify incomplete patterns into the following categories:

- **Type I incomplete** -- for every shift cycle, all missing states on the shift cycle are contiguous.
- **Type II incomplete** -- some shift cycle has missing states that aren't contiguous.

A similar terminology is applied to blocks within prime patterns. A block of length (Cycle Length - 1) is a **complete block**, and a shorter one is an **incomplete block**. A complete pattern is composed entirely of complete blocks.

Inverses

Now let `P` be a complete prime pattern ($L=L_{bound}$). Is there any structure to the states which are *missed* by `P`? In fact there is, and to demonstrate consider again the complete prime pattern `+++1+--++3-++-+1+-++-6-+-++4--` in `(4,7)`. This pattern misses the following 5 states in the graph:

```
---xxxx
-x--xxx
--xx-xx
-x-xx-x
--x-xxx
```

Reversing these states, we find that they form the list of states for a valid pattern:

```
          xxxx---  <--
--6-->  xxx--x-     |
--4-->  xx-xx--     3
--6-->  x-xx-x-     |
--1-->  xxx-x--  ---
```

The numbers in this pattern are just the link throws in the original pattern, subtracted from `h`!

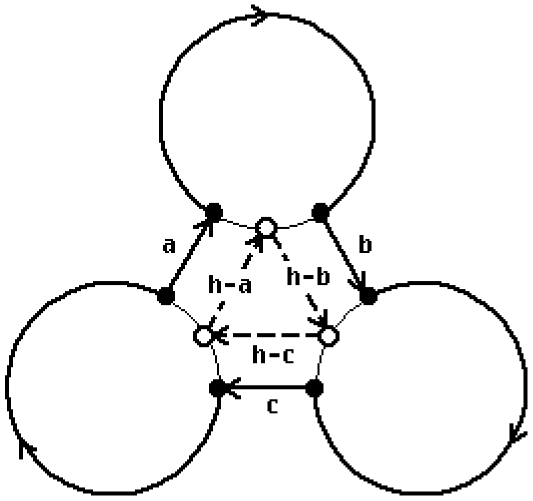
This apparent magic is due to the following simple result. (We introduce here the notation `A'` to denote the **reverse** of the `x/-` sequence `A`.)

> **Theorem** Assume a link throw `T` connects state `xB` to state `A-`, where `A` and `B` are `x/-` sequences of length `h-1`. Then state `xB'` is connected to state `A'-` with the link throw `h-T`.

(The link throw is out of state `xB` and so the state `Bx` is missed by the pattern. The theorem shows that the reverse of this missed state, `xB'`, is traversed by the pattern derived from the link throws.)

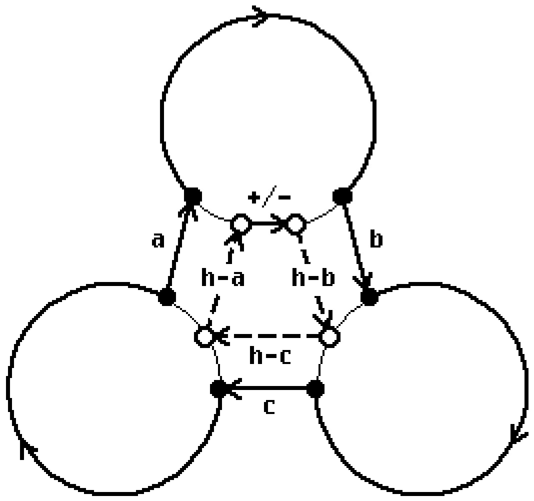
In fact the theorem shows that for any complete prime pattern, the link throws when subtracted from `h` form a valid pattern. I call this the **inverse** of the original pattern. For example, the longest prime pattern in `(3,9)` is the complete pattern `++5----+-+1+----+-8-+----+7--+---+1+--+---8-8--+--+-8---+--+6----++-8-----` ($L = L_{bound} = 74$); the inverse `4812811131` is read off from the link throws and is also a valid pattern in `(3,9)`.

Below I show schematically what is going on, in a case with 3 shift cycles in the graph (recall that for complete prime patterns there is exactly one state missing from each shift cycle). Open circles indicate states missing from the pattern, and dashed arrows indicate throws between these (reversed) states forming the inverse pattern.



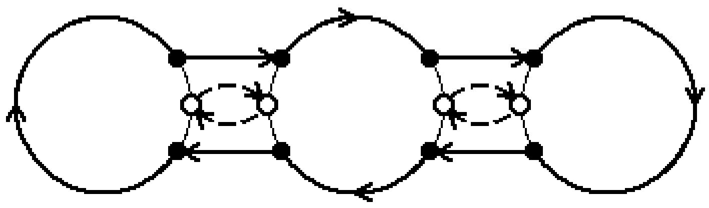
The inverse patterns have a much stronger property than just being valid, however. The inverse of a complete prime pattern has the interesting property that each of its states lies on a different shift cycle (states `A` and `B` are on the same shift cycle if and only if `A'` and `B'` are). I call patterns with this property **superprime**. They provide a very efficient way to find complete prime patterns.

What can we say about incomplete prime patterns? A Type I incomplete pattern has a well-defined inverse, shown schematically in the following figure, where there is an extra missing state on one of the shift cycles ($L = L_{bound}$-1). The inverse follows along the (contiguous) reversed missing states with shift (`+`/`-`) throws and is not superprime.



It should be graphically clear in the figure above how the inverse operation is defined. An important property is that the inverse of the inverse pattern is the original pattern (inverse^2 = 1). We can efficiently find complete and Type I incomplete prime patterns by finding their inverses and applying the inversion operation.

Type II incomplete prime patterns don't always have an inverse that is one continuous path through the graph. Sometimes the reversed missing states form several disjoint patterns, not just one, as seen in the following figure. Finding Type II incomplete patterns is more computationally difficult for this reason. A reasonably efficient algorithm can nevertheless find quite long Type II patterns (L > 1000).



The Challenge

Which graphs `(b,h)` contain complete prime patterns?

The answer is unknown, and even a partial answer would be impressive.

Dietrich Kuske has proven that there are no complete prime patterns in `(b,2b)` when `b>2`.

The List

For each pattern in `(b,h)` there is a **duality transform** to get a pattern in `(h-b,h)`: you reverse the pattern throw by throw, and then subtract all throws individually from `h`. For example, one of the patterns in `(3,5)` is `423`; applying the duality transform gives `231` in `(2,5)`. A nice property of this transform is that a pattern is prime if and only if its dual is as well (exercise: figure out the relationship between a pattern's list of traversed states and that of its dual). In particular, a maximally-long prime pattern in `(b,h)` will have a maximally-long dual in `(h-b,h)`, and so **we only need to calculate prime patterns for the cases where `h>=2b`.** An interesting case is when `h=2b`, and you find dual pairs, for example `661600550606400` and `662060611660500` from `(3,6)`. Some patterns, such as `4400` in `(2,4)`, are *self-dual*.

The list below summarizes everything I know about the longest prime patterns. It is organized according to number of balls and maximum throw value.

The **L, L_{bound}, # States** column shows length of the longest prime patterns, the upper bound on this length (discussed above), and number of states in the graph, respectively. These obviously must form a nondecreasing sequence.

The **Number of Patterns** column indicates how many patterns there are of the maximum length. For cases where the longest patterns are incomplete, a pair of numbers is given in brackets indicating the number of Type I and Type II patterns respectively. '?' entries indicate that complete calculations have not yet been done and there may be additional patterns of the maximum length. In all cases, however, it is known with certainty that there are no prime patterns longer than the ones presented here.

Lastly, the **Number of Patterns** entries link to another page listing the patterns themselves.

Note: These results were all obtained with my "jdeep" program, available on my juggling page (linked from the bottom of this one).

	Maximum Throw	**L, L_{bound}, # States**	**Number of Patterns**
2 balls	3	3, 3, 3	1
	4	4, 4, 6	2
	5	8, 8, 10	1
	6	12, 12, 15	1
	7	18, 18, 21	1
	8	24, 24, 28	1
	9	32, 32, 36	1
	10	40, 40, 45	1
	11	50, 50, 55	1
	12	60, 60, 66	1
		(pattern continues)	
3 balls	4	4, 4, 4	1
	5	8, 8, 10	1
	6	15, 16, 20	{6,0}

	7	30, 30, 35	1
	8	49, 49, 56	3
	9	74, 74, 84	1
	10	108, 108, 120	1
	11	149, 150, 165	{18,0}
	12	200, 201, 220	{28,2}
	13	263, 264, 286	{4,4}
	14	337, 338, 364	{38,0}
	15	424, 424, 455	1
	16	524, 525, 560	{20,10}
	17	639, 640, 680	{34,4}
	18	769, 770, 816	{50,7}
	19	917, 918, 969	{0,4}
	20	1082, 1083, 1140	{92,4}
	21	1266, 1266, 1330	1
	22	1469, 1470, 1540	{>=4,?}
4 balls	5	5, 5, 5	1
	6	12, 12, 15	1
	7	30, 30, 35	1
	8	58, 60, 70	{26,18}
	9	112, 112, 126	1
	10	188, 188, 210	9
	11	300, 300, 330	144
	12	452, 452, 495	45
	13	660, 660, 715	>=245
5 balls	6	6, 6, 6	1
	7	18, 18, 21	1
	8	49, 49, 56	3
	9	112, 112, 126	5
	10	225, 226, 252	{752,86}
	11	420, 420, 462	59346
6 balls	7	7, 7, 7	1
	8	24, 24, 28	1
	9	74, 74, 84	1
	10	188, 188, 210	9
	11	420, 420, 462	59346
7 balls	8	8, 8, 8	1
	9	32, 32, 36	1

	10	108, 108, 120	1
	11	300, 300, 330	144
8 balls	9	9, 9, 9	1
	10	40, 40, 45	1
	11	149, 150, 165	{18,0}
	12	452, 452, 495	45
9 balls	10	10, 10, 10	1
	11	50, 50, 55	1
	12	200, 201, 220	{28,2}
	13	660, 660, 715	>=245

Back to my juggling page.