Notes on Prime Juggling Patterns

Jack Boyce, jboyce@gmail.com

version dated March 30, 2025

Note: These notes assume a familiarity with siteswap notation, a popular system for describing juggling patterns. If needed consult [1] for an introduction. These notes are an updated and corrected version of a paper written in 1999 and published online [2].

Here we look at a particular kind of pattern, called a *prime* pattern. We'll start with some definitions and a look into the theory of prime patterns. Then we'll conclude with a discussion of `jprime`, a computer program designed to search for prime patterns, and results obtained to date.

1. Definitions and Theory

1.1. Prime patterns

A *prime pattern* is defined as a juggling pattern that contains no repeatable subsequences. For example the pattern `423` is not prime, since the subsequence `3` can be repeated indefinitely within the pattern:

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

as can the subsequence `42`:

```
... 4234234242424242423423 ...
```

An interesting fact is that if you restrict yourself to a certain number of balls and some maximum throw value, then there are infinitely many juggling patterns – but only finitely many of them are prime! For example, with 3 balls and no throw values greater than 5, the exhaustive list of prime patterns is:

```
3
42
51
441
522
531
450
4440
4530
5241
5340
5511
5520
45501
52440
52530
```

```
53502
55140
55500
55050
455040
525501
551502
5255040
5350530
55150530
```

Any other 3 ball, height 5 siteswap that you might write down – and there are an unlimited number of them – can be decomposed into a combination of these 26 patterns! In the case of `423` we can spot it immediately: `423` is a composition of prime patterns `3` and `42`.

In this sense the prime patterns are like the prime numbers in arithmetic, in that they cannot be decomposed into smaller parts.

From a practical juggling standpoint, we find that the "best" patterns (nicest to look at, most fun to juggle) are often prime patterns. The fact that they are not compositions of shorter patterns makes them aesthetically pleasing, and each one unique.

1.2. Juggling states

The easy way to tell if a pattern is prime is to look at the *juggling states* it traverses. A state is a record of the times in the future when the balls in the air will land. (The state in general changes from one throw to the next.) For mathematical simplicity we model the throwing process as happening instantaneously, so at any moment there are always b balls in the air coming down to land. When a siteswap throw t is made, that ball lands – and is instantly re-thrown – t time units later.

Figure 1 below illustrates how we determine the juggling state immediately after a throw. In this case we are continuously juggling the pattern `423`, and the first row in the figure shows the landing rhythm if we stop throwing after the `3`. Arrows and colored dots indicate the sources and landing times of the three balls in the air. Again the rule is simple: A siteswap throw t lands t time units after it is thrown. We can see that the landing balls fill the first three available slots, which we indicate with an `x` in those first three positions. We then say that the *juggling state* after throw `3` is `xxx---`. (Note there is an implied infinite number of empty slots off the right, which we truncate to some reasonable length.)

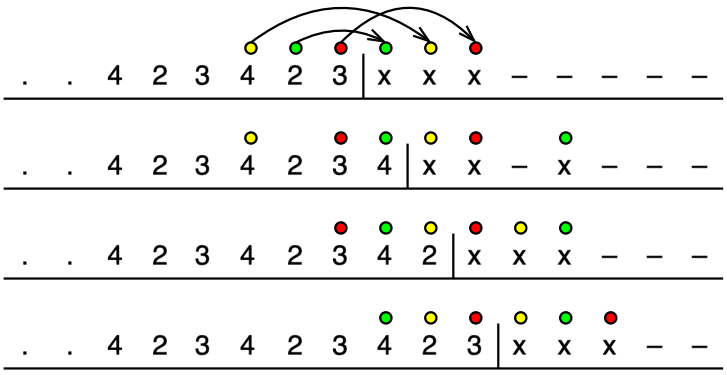

Figure 1: States traversed by juggling pattern `423`.

The other rows in the figure show what happens if we stop after each subsequent throw in the pattern. Following the same procedure, stopping after the next throw (`4`) yields the juggling state `xx-x--`. Stopping after the `2` yields `xxx---`, and stopping after the next `3` yields `xxx---` again.

We can summarize this situation succintly as follows:

```
3->  xxx---   4–>   xx-x--   2–>  xxx---   3–>  xxx---   (etc.)
```

Because `423` is a valid pattern, it returns to the same juggling state after each cycle through the pattern. Also, when we were constructing the states we never got a situation where two balls landed at the same time: Each of the `x`s fell into a separate time slot. These two conditions (returning to the initial state, and no colliding balls) are necessary and sufficient for a sequence of numbers to be a valid pattern.

A virtue of the juggling state concept is that there are simple rules telling us how the state evolves from one throw to the next, based on 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` (where the leftmost position corresponds to position 1). If we make throw `T`, then we convert position `T` to an `x` and we have our new state.
3. If a `-` shifted off the left, we must throw a value `0` for the current beat. In siteswap notation, throw value `0` is special and corresponds to no throw, i.e., an empty hand.

From these rules we can see that the total number of `x`s in the state is always b, the number of balls. Moreover we can truncate the number of positions in the state to h, the maximum throw value, since no slot to the right of h can ever be filled. This gives us:

Theorem. The number of juggling states with b balls and maximum throw value h, is $\binom{h}{b} = \frac{h!}{b!(h-b)!}$.

Proof. We are arranging b `x`s in the h slots of the juggling state, and ordering is not important. From basic combinatorics we have $\binom{h}{b}$ distinct ways of making this arrangement, each of which is a valid juggling state.

1.3. State graphs

Now that we have our states, we can join them together to form the *state graph* (b, h). The vertices of the graph consist of all possible states, which we join together by arrows representing all the allowed throw transitions between those states. In particular an arrow from S_1 to S_2 indicates there is an allowed single-beat transition (via the rules above) taking us from S_1 to S_2, which we label with the corresponding throw value. Figure 2 shows the state graph $(3, 5)$.

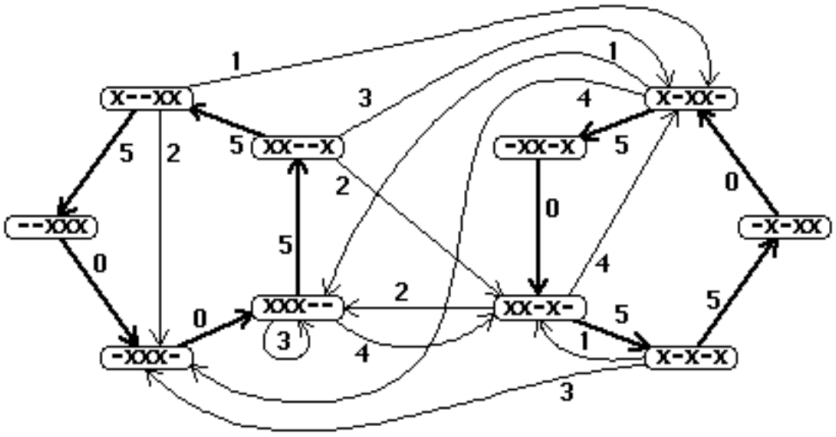

Figure 2: State graph $(3, 5)$ with throw values labeling arrows between juggling states. The significance of the heavy arrows is discussed below.

Once the graph is drawn, it's easy to find juggling patterns: We start at some state S and follow the arrows around the graph until we return to S, recording the throw values along the way. We then have a pattern since we could repeat indefinitely the same path through the graph.

As an aside, by convention we single out the unique state with all of the `x`s on the left (`xxx--` in Figure 2 above) as a special one called the *ground state*. A pattern that travels through this state is called a *ground state pattern*, while all others are called *excited state patterns*. This nomenclature is due to Paul Klimek, an original inventor (in 1981) of this numbers-based notation that he called *quantum juggling*. (I learned the notation from Bruce "Boppo" Tiemann at Caltech in 1988, who was an important popularizer [3] of the notation and coiner of the name *siteswap*. The ideas of juggling states, state graphs, and prime patterns came in 1990, as I was writing a program to find siteswap patterns (`j.c`) and needed a way to identify the "obvious compositions" in my pattern lists.)

More formally, by introducing the juggling graph (b, h) we have defined a new representation of siteswap patterns: The siteswap with throw values $t_1 t_2 ... t_n$ is uniquely identified with a list of states traversed in the graph, $S_1, S_2, ..., S_n, S_{n+1}$. Here $S_1 = S_{n+1}$ and the throw value t_i corresponds to transition $S_i \to S_{i+1}$. Given a list of throws $\{t_i\}$ we can generate the state traversal list $\{S_i\}$, and vice versa. They are equivalent representations of a siteswap pattern.

With these definitions in place, we can now connect back to the idea of a prime pattern introduced at the beginning of the paper.

Theorem. A pattern P is prime if and only if the states that P traverses, $S_1, S_2, ..., S_n, S_{n+1}$, are all distinct except for $S_1 = S_{n+1}$.

Proof. Assume P's list of states $S_1, S_2, ..., S_n, S_{n+1}$ are *not* all distinct, in other words, there are some j, k such that $1 \le j < k < n+1$ and $S_j = S_k$. Then by the rules described above, states $S_j, S_{j+1}, ..., S_k$ themselves form a valid pattern because $S_j = S_k$, and this pattern has a period $k - j$ that is less than n. Thus we have identified a subpattern within P, so P is not prime. This establishes the direction P prime $\rightarrow$ distinct S_i. Now assume that P is not prime, i.e., it has a repeatable subsequence $t_j t_{j+1} ... t_k$ of throws, such that $1 \le j < k < n$. For this sequence to be repeatable, its beginning and ending states must be the same, or $S_j = S_{k+1}$, or in other words the original set of states $S_1, S_2, ..., S_n$ are not all distinct. This establishes the direction distinct $S_i \rightarrow P$ prime.

In plainer language, a prime pattern touches upon no state more than once in its path through the graph. In the language of graph theory, a prime pattern is a *cycle* in the graph.

Corollary. Let P be a prime pattern with b balls, maximum throw value h, and period n. Then $n \le \binom{h}{b}$.

Proof. There are $\binom{h}{b}$ vertices in graph (b, h), and because P's states are all distinct, the upper bound follows.

Corollary. There are a finite number of prime patterns with b balls and maximum throw value h.

Proof. Prime patterns correspond to cycles in the juggling graph (b, h), and the cycles of a finite graph are finite in number.

1.4. Shift cycles

At the start of this paper we made note of the fact that `55150530` is the longest prime pattern in graph $(3, 5)$. This pattern has period 8, which is two smaller than the number of vertices in the graph. Is it possible to find a prime pattern that visits *every* state in its graph? (This is known to mathematicians as a "Hamiltonian cycle".)

Indeed it is not, except in certain edge cases, and to understand why we need to understand more about the structure of juggling graphs.

We'll begin with some empirical facts that were first discovered by Johannes Waldmann in 1998. Johannes noted that if we restrict ourselves to some maximum throw value h, then the longest prime patterns are comprised mostly of throws 0 and h. Johannes also noted that the 0 and h throws commonly occur in groupings of length $h - 2$. Using these observations he wrote an efficient computer program to generate long prime patterns, and conjectured that they were maximally-long (in most cases he was correct). In this and the next section we will understand why this form is so prevalent in long prime patterns.

Consider the juggling graph (b, h), and in particular some state S_1 in (b, h). Now S_1 always has either a 0 or h throw leading out of it, as determined by the contents of its leftmost position. Following this arrow, we get to a new state S_2 which is just S_1 shifted left by one position: whatever symbol dropped off the left reappears on the right. This process can be repeated starting at S_2, to generate a whole sequence $S_3, S_4, ...$ of shifted states.

As we make these shifting throws, it's clear that eventually we must return to our starting point S_1. In every case this happens after at most h throws, and sometimes sooner (for example from

state `x-x-x-` in $(3, 6)$ we return after just two shift throws). By returning to our starting point, we have in fact formed a valid pattern! Moreover it is easy to show that it must be prime, because all the states in its traversal list are distinct.

This special prime pattern is called a *shift cycle*, made using only throws 0 and h, and consisting of S_1 and its distinct rotations. Each state in (b, h) lies in exactly one shift cycle, and the shift cycles together form a set of prime patterns that completely cover the vertices in the graph.

As an example, take a moment to identify the shift cycles in Figure 2. (Hint: they are shown in heavy arrows.) In the case of $(3, 5)$ there are two shift cycles. Confirm that every state in the graph is in exactly one shift cycle.

1.5. Excluded states and the prime period bound

In constructing the shift cycles above we used only throws 0 and h, which for hopefully obvious reasons we will call *shift throws*.

The only way to get from one shift cycle to another, then, is to make a throw that isn't 0 or h: we must use one of $1, 2, ...,$ or $h - 1$. We will call these intermediate throw values *link throws*.

On the left side of Figure 3 we illustrate part of a prime pattern that link throws onto a shift cycle, follows it with $0/h$ throws, and then link throws off.

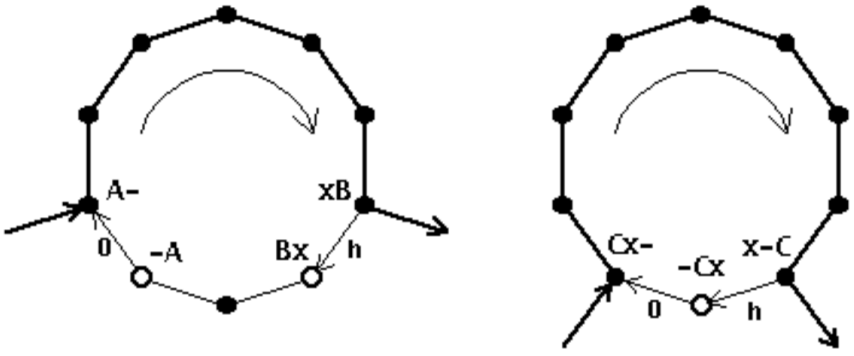

Figure 3: Portion of a prime juggling pattern joining a shift cycle, traversing it with shift throws, and exiting with a link throw. Open circles represent states excluded from the pattern.

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

1. State `-A` has not been visited by the pattern yet. If it had then `A-` would also have been visited before, since `-A` can only exit to `A-`. This would contradict our assumption that this is a prime pattern (`A-` would be visited twice).
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 from S to be missing from a prime pattern.** This is indicated in Figure 3 by the open circle for state `-A`.

A similar logic applies to a link throw *off* a shift cycle, which can only occur at states of the form `xB`. The next state downstream in the shift cycle is `Bx`, which can only be entered with a throw h from our exit state `xB`. So 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 Figure 3.

Theorem. Let P be a prime juggling pattern that visits more than one shift cycle. Then P must miss at least one state on each shift cycle it visits.

Proof. In visiting multiple shift cycles, P must link throw on and off each shift cycle it visits. By the argument above, at least one state must be missed on any shift cycle visited in this way.

We can now understand why long prime patterns are usually composed of long blocks of shift throws: Each link throw excludes at least one state, so the longest patterns will tend to stay on a shift cycle for as long as possible before link throwing to another shift cycle.

As an aside, when we write down long prime patterns it is often convenient to use a special notation called *block form*, introduced by Waldmann. Here we substitute '`-`' for 0, '`+`' for h, and use numbers for the link throws. For example the long pattern `777170077307707170770607077400` in $(4, 7)$ is written in block form as `+++1+--++3-++-+1+-++-6-+-++4--`, which more clearly highlights the link throws involved.

Corollary (1999 conjecture by Johannes Waldmann). Let P be a prime juggling pattern of period n in juggling graph $G = (b, h)$, where G contains more than one shift cycle. Then $n \le n_{\text{bound}}$, where:

$$n_{\text{bound}} = \binom{h}{b} - [\#\text{ of shift cycles in } (b, h)]. \tag{1}$$

Proof. P must miss at least one state on each shift cycle it visits, so in the extremal case it can visit at most every state in the graph, minus one per shift cycle.

1.6. Complete prime patterns

A *complete prime pattern* is one that is as long as possible, i.e., it has period $n = n_{\text{bound}}$ as defined above. A complete prime pattern misses exactly one state on each shift cycle in the graph. (By definition a complete prime pattern can only exist if there is more than one shift cycle in the graph.)

Which graphs (b, h) have complete prime patterns? Empirically we find that some do, and some do not. One positive example is the graph $(3, 5)$ discussed above. It contains the prime pattern `55150530` which we can now see is a complete prime pattern since $n = n_{\text{bound}} = 10 - 2 = 8$. Conversely, some graphs such as $(3, 11)$ have no complete prime patterns.

The only general result known to date is the following.

Theorem (Dietrich Kuske 1999). There is no $b > 2$ such that $(b, 2b)$ contains a complete prime pattern.

Proof. Assume you have a complete prime pattern P in $(b, 2b)$. Then consider the shift cycle containing the state `(x-)^b` (i.e., the state you obtain by repeating the sequence '`x-`' b times). This shift cycle consists of just two states, the other one being `(-x)^b`. Thus, the only possible entry (exit) point of this shift cycle is `(x-)^b`, and because P is a complete prime pattern it must traverse this state.

Now the predecessor of `(x-)^b` in P's traversal list must be `x-(-x)^{b-1}`, as follows: It has to be of the form `x-A`, as discussed already (see Figure 3). Now the only possibility to reach a state starting with `x` from `x-A` is to throw a 1. Hence `A=(-x)^{b-1}`.

Similarly, the successor of `(x-)^b` in P's traversal list must be `(-x)^{b-1}x-`, as follows: It has to be of the form `Bx-`, as shown in Figure 3. But P can only get to this state with a throw $h-1$, which implies that `B=(-x)^{b-1}`.

To summarize, we have shown that P enters the shift cycle in question from state `x-(-x)^{b-1}`, and exits to state `(-x)^{b-1}x-`. However by inspection these two states belong to the same shift cycle. Hence, the graph $(b, 2b)$ contains two shift cycles only. The only cases where this holds are $b = 1$ and $b = 2$. qed

1.7. Pattern inverses

Now let P be a complete prime pattern, as defined above ($n = n_{\text{bound}}$). Is there any relationship between the states that are *missed* by P?

In fact there is, and to demonstrate consider the complete prime pattern `+++1+--++3-++-+1+-+ +-6-+-++4--` in $(4, 7)$. This pattern excludes the following five states (one per shift cycle) as it traverses through the graph, in this order:

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

Now if we *reverse* these states, i.e. flip each one right to left, we find that they form a traversal list for a valid pattern!

```
xxxx---   6–>
xxx--x-   4–>
xx-xx--   6–>
x-xx-x-   1–>
xxx-x--   3–>
```

Also, we can see that the numbers in this new pattern `64613` are just the link throws in the original pattern, subtracted from h!

What is going on here? This apparent magic is due to the following result. (We introduce here the notation `A'` to denote the reverse of the `x/–` sequence `A`.)

Theorem. Assume a link throw t_i on prime pattern P connects state S_i to state S_{i+1} on different shift cycles in juggling graph (b, h). Then:

1. S_i must be of the form `xB`, and S_{i+1} must be of the form `A-`, where `A` and `B` are `x/-` sequences of length $h-1$.

2. States `Bx` and `-A` are excluded from P.
3. The reverses of these excluded states, `xB'` and `A'-`, are connected by link throw $h - t_i$.

Proof. Parts 1 and 2 follow from the discussion above on link throws. For part 3, we note that t_i connects `xB` to `A-`, so sequence `A` consists of sequence `B` with position t_i filled in. Reversing these, sequence `A'` then consists of sequence `B'` with position $h - t_i$ filled in. Thus there is a throw $h - t_i$ connecting `xB'` $\rightarrow$ `A'-`, as claimed.

Figure 4 below shows schematically a complete prime pattern P in a graph with three shift cycles, missing exactly one state in each shift cycle. (The states traversed by P are indicated by the heavy line, and the missed states by open circles.) By the theorem above, corresponding to each of the link throws a, b, c there are throws $h - a, h - b, h - c$ between the reversed missing states of P.

This pattern traversing the reversed missing states of P is called the *inverse* of P.

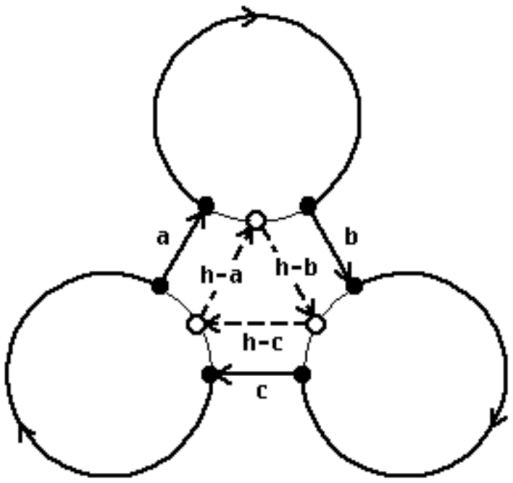

Figure 4: The inverse of a complete prime pattern P, shown with dashed lines.

This diagram may be misleading because the reversed state need not lie on the same shift cycle as its corresponding missing state. For example if state `--x-xx` is our missing state, its reversal `xx-x--` lies on a different shift cycle in $(3, 6)$. However we do have the following:

Theorem. Let S_1 and S_2 be two states in a juggling graph, and let S_1' and S_2' be their reverses. Then S_1' and S_2' lie on the same shift cycle if and only S_1 and S_2 are on the same shift cycle.

Proof. Assume S_1 and S_2 lie on the same shift cycle. Then S_1 and S_2 are shifted versions of one another. When we reverse each, it's clear the reversed versions will also be shifted versions of one another, so S_1' and S_2' lie on the same shift cycle. The opposite direction is immediate because $(S')' = S$.

As another example, the longest prime pattern in $(3, 9)$ is the complete pattern `++5----+-+1+----+-8-+----+7--+---+1+--+---8-8--+--+-8---+--+6----++-8-----` ($n = n_{\text{bound}} = 74$). We can read off its inverse `4812811131` directly from the link throws.

The pattern inverse idea is more general than what we've discussed so far. Consider Figure 5 below, which shows schematically a pattern P that is nearly complete, but misses two neighboring states on one shift cycle ($n = n_{\text{bound}} - 1$).

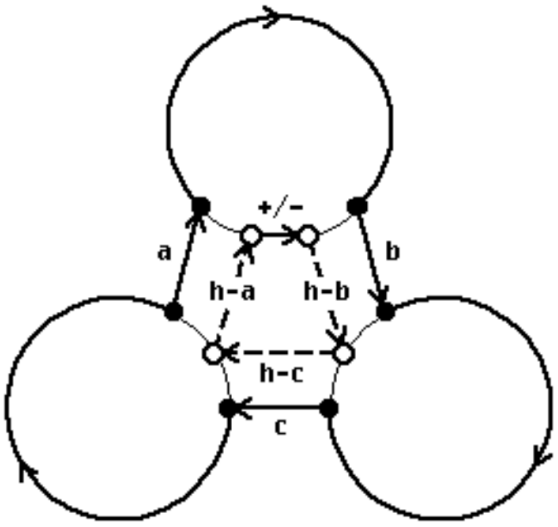

Figure 5: The inverse of a prime pattern that is not complete.

The inverse pattern can still be defined, as shown in the figure. Here the inverse includes a shift throw, to traverse between the two adjacent reversed missing states of P. Note that when we apply shift throws to the reversed states, we effectively move in the opposite direction on the shift cycle.

Additionally, there is no reason that P must visit *all* shift cycles of its graph in order to have an inverse. We can define this inversion operation as acting on whatever subset of shift cycles P visits, so long as it visits at least two.

Conversely, consider what happens when P *revisits* a shift cycle, as shown in Figure 6. Here the middle shift cycle is exited with a link throw, and then revisted later in the pattern.

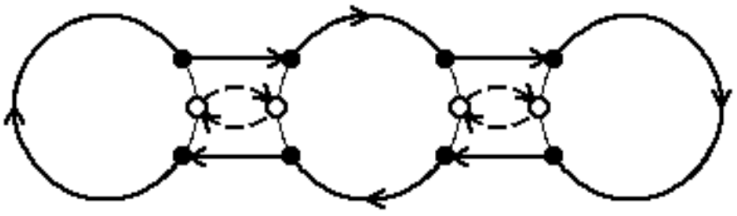

Figure 6: A pattern that revisits a shift cycle does not have a well-defined inverse.

In this case the link throws between the reversed missing states of P do not stitch together into a single pattern of their own; we end up with multiple disjointed patterns. In such a case we say that the inverse does not exist.

We even find unusual cases where a pattern is its own inverse. Examples are `6330` in $(3, 6)$ and `88488040` in $(5, 8)$.

1.8. Superprime patterns

A prime pattern P is called *superprime in graph* G if two conditions hold:
1. P visits more than one shift cycle in G.
2. P contains exactly k link throws, where k is the number of shift cycles visited by P.

Observations. A superprime pattern in G never revisits a shift cycle, or link throws within a shift cycle. Moreover, for the shift cycles visited by the pattern, all states missed by the pattern are contiguous in the shift cycle.

Proof. Recall that the only way to leave a shift cycle is via a link throw, so we must have at least as many link throws as cycles visited. Thus the second condition on superprime patterns has some teeth: It means we get exactly one link throw per cycle visited. This implies a shift cycle cannot be revisited, as that would require multiple link throws off that cycle. It also implies that link throws *within* a shift cycle (e.g., `xx-x-`→`x-xx-`) are not allowed, as this would also entail multiple link throws for that cycle. Since no shift cycles are revisited, the missed states on each must be contiguous.

Corollary. Let P be a complete prime pattern in graph G. Then P is superprime in G.

Proof. The first condition for superprimality is immediate from the definition of a complete prime pattern. Now, let k be the number of shift cycles in our graph G, so that P excludes exactly k states. Every link throw in P excludes at least one state, so we know there can be at most k link throws in P. However we also know there must be at least k link throws, since exiting any shift cycle requires a link throw. So there are exactly k link throws in P, as required.

Observations (stated here without proof). Let P be superprime in graph G. Then:
1. inverse(P) exists and is also superprime in G.
2. inverse(inverse(P)) = P.

The pattern inverse idea helps us greatly in a computer search for complete prime patterns. These patterns are long and generally difficult to find, but they are all superprime, and thus each one has an inverse that is much shorter. Moreover this inverse has some special characteristics: It too is superprime, it contains no shift throws, and it has a period equal to the number of shift cycles in G. So our search strategy is: Find these short superprime patterns, and then invert them to get our complete prime patterns.

Corollary. Let P be a complete prime pattern in graph G. Then there exists a pattern Q that is superprime in G, contains no shift throws, has a period equal to the number of shift cycles in G, and satisfies $P = \text{inverse}(Q)$.

Proof. Consider the pattern $Q = \text{inverse}(P)$, where the inverse is defined as shown in Figure 4. It is clear that Q has no shift throws, and has a period equal to the number of shift cycles in the graph (which is greater than one from the definition of a complete prime pattern). Because Q has no shift throws, all its throws are link throws and thus the second condition for superprimality is satisfied. Hence Q is superprime in G, and by our observation above, $P = \text{inverse}(\text{inverse}(P)) = \text{inverse}(Q)$.

As an example of this strategy, consider again graph $(3, 5)$ in Figure 2. There are two shift cycles, and so we seek a prime pattern with period 2 that visits both. By inspection we can see there is just one such pattern, `42`, which cycles between states `xxx--` and `xx-x-`, and we want to invert this (superprime) pattern. Reversing these states, we know our complete prime pattern P misses states `--xxx` and `-x-xx`, which are at the far ends of the figure. Visually we can read off the inverse by following each shift cycle around its missing state, and stitching the throw sequences together with link throws. So starting from `-xxx-` we have `055` for the left shift cycle, then link throw a `1`, then on the right shift cycle we have `505`, then link throw back to the start with `3`: That is `05515053` in total, or `55150530` if we start the cycle from the ground state.

What about prime patterns with $n < n_{\text{bound}}$? Here we find that not all of them are superprime, unlike the complete prime patterns where it is guaranteed. The superprime ones are generally easier to find because we can use a search strategy similar to the above. Figure 5 shows an example with $n = n_{\text{bound}} - 1$, where the inverse Q is superprime, period 4, and contains a single shift throw. A computer search for period-4 superprime patterns is much faster than a search for period-$(n_{\text{bound}} - 1)$ prime patterns!

As a computational note, superprime patterns are significantly faster to find than prime patterns in general. The reason is that during a search, after each link throw we can exclude from future consideration all other unused states on the departed shift cycle, because that cycle cannot be revisited. This constraint can be applied as the search proceeds and it greatly reduces the size of the search space.

1.9. Dual patterns and dual graphs

We end this section with a curiosity, not very related to the development so far but often useful in computer searches.

There exists a *duality transform* for juggling patterns, defined as follows: Take any pattern P in (b, h), reverse it throw by throw, then subtract each throw from h. This process always results in another valid pattern, but with $(h - b)$ balls!

For example, in $(3, 5)$ the pattern `423` that we saw at the beginning of the paper transforms into `231`, which is a two ball pattern.

It's clear from the definition that applying the duality transform twice gets us back to where we started. In other words, $\text{dual}(\text{dual}(P)) = P$. Because of this, there is a one-to-one mapping between the patterns in (b, h) and the patterns in $(h - b, h)$! This implies that the graphs have the same number of patterns, the same distribution of pattern periods, and so on. (That they have the same number of vertices can be readily shown from the symmetry properties of binomial coefficients.)

Observations (stated here without proof). Let P be a juggling pattern in (b, h), and $\text{dual}(P)$ be its dual. Then:

1. P is prime if and only if $\text{dual}(P)$ is prime.
2. P is superprime in (b, h) if and only if $\text{dual}(P)$ is superprime in $(h - b, h)$.
3. P is a complete prime pattern in (b, h) if and only if $\text{dual}(P)$ is a complete prime pattern in $(h - b, h)$.

So the duality transform preserves these pattern properties we care about.

Using these ideas, it is possible to take our entire graph (b, h) and apply some surgical rules to transform it in place into the graph $(h - b, h)$ (again, stated here without proof):

1. Replace each state S with its inverted and reversed version (by "inverted" we mean to swap `-` and `x`). For example `xx-x-` becomes `x-x--`.
2. Reverse the direction of each arrow.
3. Replace the throw value t_i labeling each arrow with $h - t_i$.

From these actions we can see intuitively what the dual pattern is: It traverses the same loop as the original pattern, but in the opposite direction since the arrows have been reversed. We can also see why the shift cycle structure is unchanged, and why primality and superprimality are preserved under duality.

To be clear however, the graphs (b, h) and $(h - b, h)$ are not isomorphic! They have a different distribution of vertex outdegrees, for example. Nevertheless their patterns (closed walks) are in one-to-one correspondence, and with many properties conserved.

As a fun observation note that graph $(b, 2b)$ is *self-dual* and we can find dual pairs of patterns, for example `661600550606400` and `662060611660500` in (3,6). And the occasional pattern is its own dual, such as `531` or `4400` or `62340`.

The duality transform helps us in computer searches, because depending on the algorithm one or the other graph may be faster to search over. If we search on the dual graph we can use the duality transform to translate back as needed.

2. Counting Prime Patterns

How many prime patterns are there? In this section we consider a few different versions of this question.

We start with counting juggling patterns in general. In 1994 Buhler et al [4] proved the first counting theorem related to juggling patterns. They show that the number of patterns with b objects and period n is $(b+1)^n - b^n$. In this count the throws are allowed to take on all values up to including $h = bn$, the maximum allowed by the siteswap average rule.

This formula treats cyclic shifts of the same pattern as distinct, e.g., `531`, `315`, and `153` are counted separately. However if we let $M(n, b)$ be the number of patterns with exact period n and b objects, where cyclic shifts are not counted as distinct, then

$$(b+1)^n - b^n = \sum_{d|n} dM(d, b) \tag{2}$$

and by Möbius inversion we have

$$M(n, b) = \frac{1}{n} \sum_{d|n} \mu(n/d) \left[(b+1)^d - b^d \right], \tag{3}$$

where $\mu(n)$ is the Möbius function.

How this relates to prime patterns: Note that $M(n,b)$ includes all prime patterns of period n, since all such prime patterns have exact period n by virtue of having no repeatable subsequences. It follows that an upper bound on the number of prime patterns of period n is

$$P(n,b) \leq M(n,b). \tag{4}$$

Regarding counting prime patterns specifically, one set of results comes from Banaian et al [5]. They find an exact formula for $P(n,b)$ for the case $b = 2$, and also establish the general lower bound

$$P(n,b) \geq b^{n-1}. \tag{5}$$

As an example, for 3 objects there are 11 prime patterns of period 3: `531`, `441`, `522`, `630`, `450`, `612`, `720`, `360`, `711`, `801`, and `900`. So $P(3,3) = 11$. In this case the above upper and lower bounds are 12 and 9 respectively.

For ground state prime patterns we can get a stronger upper bound from the idea of *primitive juggling sequences*. Chung and Graham [6] define these as sequences of states $\{S_1, S_2, ..., S_n, S_{n+1}\}$ where as usual $S_1 = S_{n+1}$, and in addition none of the intermediate states $S_2, S_3, ..., S_n$ equals S_1. This is a weaker condition than primality, and from our definitions above it's clear that all prime patterns are primitive juggling sequences, but not vice versa.

Define $\text{Prim}(n,b)$ as the number of primitive ground state juggling sequences with b objects and period n. Chung and Graham show that $\text{Prim}(n,b)$ equals the coefficient of x^n in the formal power series expansion of the generating function

$$g_b(x) = 1 - \frac{1-(b+1)x}{1 - x\sum_{k=0}^{b-1}(b-k)k!x^k}. \tag{6}$$

For example, for $b = 3$ we have

$$\begin{aligned} g_3(x) &= 1 - \frac{1-4x}{1-x(3+2x+2x^2)} \\ &= \frac{x-2x^2-2x^3}{1-3x-2x^2-2x^3} \\ &= x + x^2 + 3x^3 + 13x^4 + 47x^5 + 173x^6 + 639x^7 + ... \end{aligned} \tag{7}$$

and we have 13 primitive ground state patterns with period 4. They are: `4440`, `4512`, `4530`, `5241`, `5340`, `5511`, `5520`, `6222`, `6231`, `6312`, `6330`, `6411`, and `6420`. All of these are prime with the exception of `4512`. (Note that the siteswap generator in Juggling Lab [7] produces primitive patterns by default, so that is an easy way to enumerate them.) From Equation 6 we can also derive the useful general recurrence relation:

$$\text{Prim}(n,b) = \sum_{k=0}^{b-1}(b-k)k!\ \text{Prim}(n-k-1,b). \tag{8}$$

It is clear then that an upper bound on the number of ground state prime patterns is:

$$P_{\text{gs}}(n,b) \leq \text{Prim}(n,b). \tag{9}$$

We can also consider the counting problem when the throw value h is limited. For example we listed the prime patterns in graph $(3, 5)$ and saw there are 26 in total. The only known results in the height-limited case are from computer searches.

3. Computer Searches

The remainder of this paper discusses computer searches for prime juggling patterns. Computationally this amounts to a graph search problem: Finding cycles in the graph (b, h).

`jprime` [8] is a program that finds prime juggling patterns by doing depth-first search of the juggling graph. It is written in optimized C++ and can execute the search in parallel on multiple processing cores. A work-stealing scheme is used to subdivide the search tree and distribute portions among the search workers, resulting in a nearly linear speedup when multiple cores are used.

As a performance benchmark, `jprime` finds the 30.5 billion prime patterns in graph $(3, 9)$ in 86 seconds on eight cores of a MacBook Pro (Apple M2 Pro CPU), for a total rate of about 1.6 billion search tree nodes per second.

`jprime` can also perform optimized searches for superprime patterns, which as discussed above is a useful tactic when looking for complete prime patterns. The software can also analyze arbitrary siteswap patterns provided by the user, and compute pattern inverses where they exist.

Below we report computer results on three questions:

- How many prime patterns are there with b objects and period n? I.e., what is the function $P(n, b)$ above?
- How many prime patterns are there in graph (b, h)?
- What are the highest-period patterns in graph (b, h)? Which graphs (b, h) have complete prime patterns ($n = n_{\text{bound}}$)?

3.1. Counting prime patterns, by period

Using `jprime` we have calculated the counting function $P(n, b)$ for selected values of period n and number of objects b. Results are tabulated below. For the cases $b = 2, 3, 4, 5$ these are OEIS sequences A260744, A260745, A260746, and A260752 respectively.

n	$P(n,2)$	$P(n,3)$	$P(n,4)$	$P(n,5)$
1	1	1	1	1
2	2	3	4	5
3	5	11	19	29
4	10	36	83	157
5	23	127	391	901
6	48	405	1663	4822
7	105	1409	7739	27447
8	216	4561	33812	149393
9	467	15559	153575	836527
10	958	50294	677901	4610088
11	2021	169537	3075879	25846123
12	4146	551001	13586581	142296551
13	8631	1835073	61458267	799268609
14	17604	5947516	272367077	4426204933
15	36377	19717181	1228519987	24808065829
16	73876	63697526	5456053443	137945151360
17	151379	209422033	24547516939	773962487261
18	306882	676831026	109153816505	4310815784117
19	625149	2208923853	490067180301	24208263855765
20	1263294	7112963260	2180061001275	
21	2563895	23127536979	9772927018285	
22	5169544	74225466424	43467641569472	
23	10454105	239962004807		
24	21046800	768695233371		
25	42451179	2473092566267		
26	85334982	7896286237030		
27	171799853	25316008015581		
28	344952010	80572339461372		
29	693368423	257264254717163		
30	1391049900			
31	2792734257			
32	5598733260			
33	11230441259			
34	22501784458			
35	45103949933			
36	90335055318			
37	180975948735			
38	362339965776			
39	725616088097			
40	1452416238568			

Table 1: Number of prime patterns $P(n,b)$ with period n and b objects, $2 \leq b \leq 5$.

n	$P(n,6)$	$P(n,7)$	$P(n,8)$	$P(n,9)$
1	1	1	1	1
2	6	7	8	9
3	41	55	71	89
4	264	410	601	843
5	1777	3163	5227	8161
6	11378	23511	44181	77248
7	76191	180477	382221	743823
8	493550	1353935	3254240	7081367
9	3263843	10297769	27976325	67880511
10	21373408	77849603	239491149	648866014
11	141617885	593258483	2061545183	6225810713
12	926949128	4486556303	17664073336	59574064361
13	6157491321	34267633327	152326783983	572427402861
14	40536148951	260349728028	1309746576182	
15	268893316363	1987331381633		
16	1777319903383			

Table 2: Number of prime patterns $P(n,b)$ with period n and b objects, $6 \le b \le 9$.

3.2. Counting prime patterns, by height

For a real juggler there is a physical limit to how high a person can throw. In fact many of the patterns found above (prime patterns of given period n) are not readily juggleable because they contain very large throw values.

Here we ask a different question: If we restrict ourselves to siteswap throws no greater than some value h, how many prime juggling patterns are there? Let $Q(b,h)$ be the count of prime patterns in graph (b,h), where as above we do not count cyclic permutations as distinct.

Table 3 and Table 4 tabulate counts for the number of prime and superprime patterns in state graph (b,h), found using `jprime`.

h	$Q(2,h)$	$Q(3,h)$	$Q(4,h)$
2	1		
3	3	1	
4	8	4	1
5	26	26	5
6	79	349	79
7	337	29693	29693
8	1398	11906414	1505718865
9	7848	30513071763	
10	42749	587709292126267	
11	297887		
12	2009956		
13	16660918		
14	133895979		
15	1284371565		
16	11970256082		
17	130310396228		
18	1381323285721		
19	16817079193531		

Table 3: Number of prime patterns in graph (b,h), with b objects and maximum throw h.

h	$S(2,h)$	$S(3,h)$	$S(4,h)$
2	0		
3	0	0	
4	4	0	0
5	16	16	0
6	56	195	56
7	176	5304	5304
8	668	376245	17393712
9	2144	133410514	3051666331193
10	10032	91844340627	
11	33696		
12	191476		
13	664816		
14	4451560		
15	15811792		
16	121840652		
17	439927232		
18	3832850784		
19	14011621440		
20	136156869348		

Table 4: Number of superprime patterns in graph (b,h), with b objects and maximum throw h.

How do these total counts break down in terms of pattern periods? When we find the 30,513,071,763 prime patterns in $(3, 9)$ we discover their periods are distributed like so:

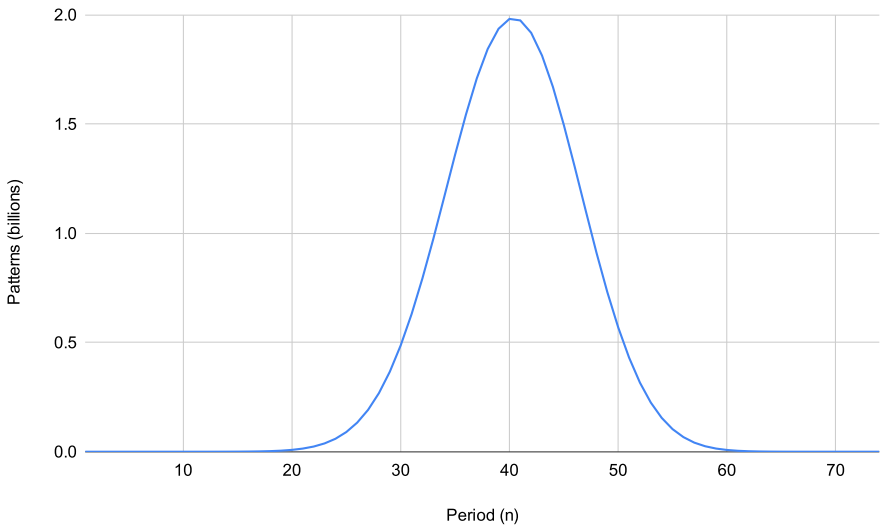

Figure 7: Distribution of periods for the 30,513,071,763 prime patterns in graph $(3, 9)$.

We see that the distribution of pattern periods is close to a normal distribution, with the most patterns at period 40 in this case. In general there are relatively few very short or very long prime patterns, and for graph (b, h) most are clustered around a period slightly less than half the number of vertices in the graph, e.g., 84 vertices in the case of $(3, 9)$.

At the extremes, for $(3, 9)$ there is a single pattern of period 1 `(3)` and a single prime pattern of period 74 `(995000090919000090809000097009000919009000808009009080009009600009908000000)`. We will have more to say about such maximally-long patterns in the next section.

3.3. Finding the longest prime patterns in (b, h)

As a final investigation, we calculate the longest prime patterns for a given number of objects b and maximum throw value h. Earlier we showed that every graph (b, h) has an upper bound n_{bound} on the period of prime patterns it contains. Some graphs contain complete prime patterns $(n = n_{\text{bound}})$, and some do not.

Table 5 below summarizes everything known about the longest prime siteswap patterns. Column n gives the period of the longest prime pattern for the given (b, h), while n_{bound} is the theoretical upper bound on that period, as given by Equation 1.

Table notes:

- The patterns are too numerous to list here; instead we tabulate pattern periods and counts. Pattern listings are maintained at [8].
- When $n < n_{\text{bound}}$, this means there are no complete prime patterns for that case. The pattern count is then shown as {superprime, non-superprime} patterns. The superprime patterns are faster to find than the non-superprime ones, and in some cases only the former have been tabulated.

- Because of the duality transform discussed above, graphs (b, h) and $(h - b, h)$ have identical results in the table.
- The table for $b = 2$ is truncated; the observed pattern appears to continue.
- Inequalities shown in the table are upper or lower bounds established from partial calculations.

The current record holders are $(3, 28)$ and $(25, 28)$, which have prime patterns containing 3158 throws. If juggled at a normal pace it would take over 10 minutes to complete a single cycle of these patterns! (Perhaps even more impressive, these 26 patterns took 2.39 core-years of processing time to find.)

b	h	n_{bound}	n	pattern count
2	3	3	3	1
	4	4	4	2
	5	8	8	1
	6	12	12	1
	7	18	18	1
	8	24	24	1
	.	.	.	.
3	4	4	4	1
	5	8	8	1
	6	16	15	{6, 0}
	7	30	30	1
	8	49	49	3
	9	74	74	1
	10	108	108	1
	11	150	149	{18, 0}
	12	201	200	{28, 2}
	13	264	263	{4, 4}
	14	338	337	{38, 0}
	15	424	424	1
	16	525	524	{20, 10}
	17	640	639	{34, 4}
	18	770	769	{50, 7}
	19	918	917	{0, 4}
	20	1083	1082	{92, 4}
	21	1266	1266	1
	22	1470	1469	{18, 4}
	23	1694	1693	{56, 4}
	24	1939	1938	{44, 3}
	25	2208	2207	{0, 4}
	26	2500	2499	{180, ?}
	27	2816	2816	1
	28	3159	3158	{26, ?}
	29	3528	<3528	{?, ?}

b	h	n_{bound}	n	pattern count
4	5	5	5	1
	6	12	12	1
	7	30	30	1
	8	60	58	{28, 16}
	9	112	112	5
	10	188	188	9
	11	300	300	144
	12	452	452	45
	13	660	660	16317
	14	928	928	63606
5	6	6	6	1
	7	18	18	1
	8	49	49	3
	9	112	112	5
	10	226	225	{752, 86}
	11	420	420	59346
	12	726	726	>=309585
6	7	7	7	1
	8	24	24	1
	9	74	74	1
	10	188	188	9
	11	420	420	59346
	12	844	843	{>=(2726+263), ?}
7	8	8	8	1
	9	32	32	1
	10	108	108	1
	11	300	300	144
	12	726	726	>=309585
8	9	9	9	1
	10	40	40	1
	11	150	149	{18, 0}
	12	452	452	45
9	10	10	10	1
	11	50	50	1
	12	201	200	{28, 2}
	13	660	660	16317

Table 5: Summary of the longest prime patterns in (b, h).

Note for case $(6, 12)$**:** We showed above that there cannot be a pattern $n = n_{\text{bound}}$ in graph $(b, 2b)$. In the case $(6, 12)$ we do find patterns of period 843, which are one shorter than n_{bound}. The superprime ones are the inverses of two different kinds of (shorter) superprime patterns: (a)

patterns that miss the `(x-)^b` shift cycle entirely, and use exactly one state on each of the other shift cycles, and (b) patterns that use state `(x-)^b` and every other shift cycle, and includes exactly one shift throw on one of these other cycles. The numbers shown are the quantities found of each type, respectively. In similar fashion the 752 maximal superprime patterns in $(5, 10)$ can be found by combining results from `jprime 5 10 25 -super 0 -inverse` (308 patterns) and `jprime 5 10 27 -super 1 -inverse` (444 patterns).

3.4. Algorithms

We conclude this section with a description of the techniques used by `jprime` to efficiently search for prime and superprime patterns. These techniques developed over various iterations of programs designed to find and count juggling patterns. In fact most of the theoretical ideas discussed in these notes came about while trying to make these searches more efficient. The algorithms evolved and led to a dramatic speedup in some cases.

In combinatorial algorithms there are two sorts of tasks: *Finding* is to generate all instances of an object, and *counting* is to discover how many there are. Sometimes the latter problem is easier than the former, as in Equation 3 which counts a particular kind of juggling pattern without finding them. However we know of no general counting algorithm for prime and superprime patterns that doesn't involve finding, so finding is our goal.

3.4.1. Basic approach

The most efficient published algorithm for finding all the cycles in a general directed graph is due to Johnson [9]. Johnson's algorithm is depth first search (DFS) that begins at a chosen starting vertex and recursively visits each outgoing link in turn. Vertices less than the starting vertex (in some ordering) are skipped to avoid double counting, and visited vertices are marked as "blocked" except for the starting vertex which completes a cycle. Johnson's algorithm also incorporates a clever scheme to not immediately unblock vertices when the search backtracks, but records the "pending" unblocks for later use. This scheme reduces the amount of fruitless searching when there is no way to return to the starting vertex without intersecting the partial path. Johnson's algorithm finds all cycles in time bounded by $O((n + e)(c + 1))$ and space bounded by $O(n + e)$, where there are n vertices, e edges, and c cycles in the graph.

In experiments with juggling graphs, we find that relative to DFS search with blocking, Johnson's algorithm reduces the size of the search tree by $28 - 32\%$ in terms of the number of search nodes explored. However, because of the additional complexity of its unblocking algorithm, Johnson has a ~25% longer run time than DFS. So `jprime` uses simple DFS with blocking as the basis of its search.

3.4.2. Tracking excluded states

In DFS the biggest efficiency gains come from finding ways to cut the search short. The search space grows exponentially with search depth, so when we can prune branches from the search tree we often see a large speedup.

The most dramatic optimization like this in `jprime` is to track excluded states when searching for prime patterns with a period close to n_{bound}. As we discussed in Section 1.5, every link throw in a prime pattern excludes at least one (and sometimes more) states from the pattern. By

accounting for excluded states as the search proceeds, we can draw inferences about the *longest possible pattern* (LPP) we can make from the current partial path.

For example, if our tracking indicates two or more excluded states on a single shift cycle, then we know that $n = n_{\text{bound}}$ is not possible. (Recall that $n = n_{\text{bound}}$ implies exactly one state missing from each shift cycle.) If we are searching for complete prime patterns then all searches from the current position will be fruitless.

The algorithm is: During partial path construction let E_i be the number of excluded states implied by link throws so far, that lie on shift cycle i. Then from the current partial path the longest possible pattern is

$$n_{\text{LPP}} = \binom{h}{b} - \sum_i \max(E_i, 1). \tag{10}$$

The maximum in this expression is because we know a priori there must be at least one excluded state on each shift cycle, so we always have $n_{\text{LPP}} \le n_{\text{bound}}$. Then as the search proceeds, whenever n_{LPP} is smaller than the pattern periods we're searching for – or more precisely, the lower end of the range of pattern periods we're searching for – we can immediately backtrack.

As an illustration of the speedup this produces, consider that `jprime` takes 661 seconds to find all prime patterns in $(3, 9)$, up to and including the single longest pattern with $n = n_{\text{bound}} = 74$ (single core of a MacBook Pro, Apple M2 Pro CPU). Using the excluded states tracking algorithm `jprime` finds those patterns with periods 69 and higher (5476 in total) in just 66 milliseconds! Until superprime patterns were discovered this was the fastest way to search for complete prime patterns.

Excluded state tracking imposes some overhead on the search, and in practice we find it is beneficial when the minimum period we're searching for is at least $\sim\frac{2}{3}n_{\text{bound}}$. In `jprime` it is automatically turned on or off accordingly.

3.4.3. Building a period-n subgraph

When we search for patterns with a fixed period, as in Table 1 and Table 2, then we can take advantage of another major optimization which is to observe that some states S cannot be part of a period-n pattern.

Theorem. Let state S be part of a period-n pattern, and let S be empty at position i (i.e., in our conventional way of notating states it has a '-' in that position). Then S must be empty at position $i + n$.

Proof. Let S be part of a period-n pattern, and empty at position i. Assume that S is filled at position $i + n$. Because S is in a period-n pattern, there is some sequence of n throws within the state graph that returns to S. In the course of doing so, the filled position $i + n$ shifts down into position i, which must then be filled. This contradicts our assumption that S is empty at position i.

We can apply this criterion before we start searching: Take the full graph (b, h) and remove all states (and the arrows to/from them) that don't meet this condition. This *period-n subgraph* is

guaranteed to have every period-n pattern in the original graph, and we can search over it instead.

For example, say we want to find all 4 ball prime patterns with period 15 (this is one of the elements shown in Table 1). A 4 ball pattern of period 15 could have throws up to and including $h = 4 * 15 = 60$, so we would need to search in graph $(4, 60)$ to be sure of finding them all. Now the graph $(4, 60)$ has 487635 states which is quite large, but its period-15 subgraph has only 3060 states and is much quicker to search over.

In fact for some of the cases in Table 1 the full graph (b, h) is impractically large. When searching for a single period, `jprime` works by generating the period-n subgraph directly, without ever building the full graph.

3.4.4. Finding superprime patterns

Superprime patterns are an important class of prime patterns that satisfy some additional properties, as discussed in Section 1.8. `jprime` includes two optimizations to speed up the search for superprime patterns.

First, we can use the fact that superprime patterns never revisit a shift cycle to extend our blocking approach as follows. When the pattern link throws onto a shift cycle C, then link throws off, all of the unused states on C are blocked, i.e., excluded from the pattern. This is a very effective procedure because each link throw removes so many states from consideration.

A second optimization for superprime patterns is specific to the case of searching for patterns with no shift throws. Here we keep track of *exit cycles* in the graph, which is defined as any shift cycle that contains a state with a transition to the starting vertex. A superprime pattern can only get back to the starting vertex by moving through an exit cycle. We track which exit cycles have been visited by the partial path, and if all have been visited then it's impossible to finish the pattern and we backtrack.

3.4.5. Searching in parallel

Depth first search (DFS) for prime patterns begins at a particular starting state S_1 and explores in turn each possible next state in the pattern, one for each link out of S_1 in the juggling graph. Each of these subsequent nodes is expanded into *its* possible next states, and so on into a tree of possibilities.

This *search tree* is shown on the left side of Figure 8. The tree is traversed from the starting node on the left to find all juggling patterns. In DFS this is done by moving down the tree using the topmost unexplored links until either the starting state S_1 is encountered again (forming a valid pattern), or we encounter a state that has already been visited. Once we have no more unexplored links from a given node, we backtrack to the closest prior node with an unexplored outgoing link, and resume.

`jprime` uses a *work stealing* approach to allow this tree traversal to be done in parallel using many independent workers. The central idea is the `split_work` operation shown in Figure 8. When a worker receives a `split_work` request, it identifies the node in its current path with unexplored links that is closest to the starting node. (This node is called the *root node* in `jprime`, and its depth in the search tree is called the *root position*.) On the left side of Figure 8

the root node is at position 0 and coincides with the starting node. The unexplored links at the root node are handed off to a second search tree as shown in the right side of Figure 8, which can be searched in parallel with the (pruned) original tree.

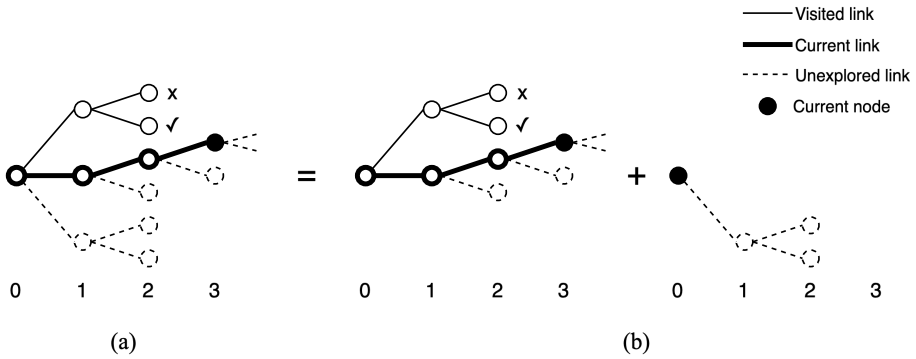

Figure 8: The `split_work` operation. The partially traversed search tree (a) is split into two disjoint parts (b) by moving the unexplored links at the root position into a second search tree. A second worker begins DFS starting at the original tree's root node, in this case at position 0.

Repeated `split_work` operations allow an arbitrarily large number of workers to cooperate in searching the tree. For example when running on a CUDA GPU over 50,000 concurrent threads may be used.

4. Open Questions

- Banaian et al. found an exact formula for $P(n, b)$ when $b = 2$. Is there a formula for any $b > 2$?
- For $P(n, b)$ can we derive tighter bounds than Equation 4 and Equation 5?
- Are there any explanations for the regularities evident in Table 5, especially for $b = 3$?
- Is there an explanation for why $(4, 8)$ has no prime patterns with $n = n_{\text{bound}} - 1$? Are there other graphs (b, h) with this same property?
- Why are the periods of prime patterns in (b, h) normally distributed, as seen in Figure 7? (Superprime patterns show a similar, but slightly broader, distribution.) Can we establish an analytical estimate for the mean or standard deviation of this distribution?

Bibliography

[1] "Siteswap." [Online]. Available: https://en.wikipedia.org/wiki/Siteswap

[2] "The Longest Prime Siteswap Patterns." [Online]. Available: https://github.com/jkboyce/jprime/blob/main/history/longest_prime_siteswaps_1999.pdf

[3] B. Magnusson and B. Tiemann, "The Physics of Juggling," *The Physics Teacher*, vol. 27, no. 8, pp. 584–589, Nov. 1989.

[4] J. Buhler, D. Eisenbud, R. Graham, and C. Wright, "Juggling Drops and Descents," *The American Mathematical Monthly*, vol. 101, no. 6, pp. 507–519, Jun. 1994.

[5] E. Banaian, S. Butler, C. Cox, J. David, J. Landgraf, and S. Ponce, "Counting Prime Juggling Patterns," *Graphs and Combinatorics*, vol. 32, no. 5, pp. 1675–1688, May 2016.

[6] F. Chung and R. Graham, "Primitive Juggling Sequences," *The American Mathematical Monthly*, vol. 115, no. 3, pp. 185–194, Mar. 2008.

[7] "jugglinglab software." [Online]. Available: https://jugglinglab.org/

[8] "jprime software." [Online]. Available: https://github.com/jkboyce/jprime

[9] D. B. Johnson, "Finding All the Elementary Circuits of a Directed Graph," *SIAM Journal on Computing*, vol. 4, no. 1, pp. 77–84, Mar. 1975.