Report: Testing conjectures on primes

Alex Ferreira Costa
Joel Costa

31 May 2015

Contents

1 Introduction

Within the frame of a university course, experimental mathematics, we tested some conjectures on prime numbers. The conjectures we tested were set up by the Chinese mathematician Zhi-Wei Sun, more precisely we decided to tackle his *60 open problems on combinatorial properties of primes.*

The aim of this project was on the one hand to gather evidence for his theories, on the other hand to approach mathematics in a different matter from how it's taught in school or university, i.e. in a more experimental way. As mathematics students, we were interested in mathematics research, and this project was a first booster to this domain, so far unknown to us.

Prime numbers have very interesting properties, be it algebraic or analytic ones. Around 300 BC Euclid demonstrated that there are infinitely many primes. We notice, that already the ancient Greeks were aware of the compelling nature of prime numbers.

Unfortunately, there is no known formula to set apart all prime numbers from composites (non-prime number). So there is still a lot to prove about prime numbers. However, the distribution of primes can be modelled. For instance there is the prime number theorem, proven in the late 19th century, stating that the probability of a randomly chosen number n to be prime is inversely proportional to $\log n$.

Still, many theories concerning prime number aren't proved yet. In Mathematics, we call these theories 'conjectures' (on prime numbers, in this case), two of the most famous being the Goldbach conjecture and the twin prime conjecture.

Goldbach conjecture: $\forall n \in 2\mathbb{N}_{\geqslant 2} : \exists p, q \in \mathbb{P} : p + q = n$
In other words, each even number $n \geqslant 4$ can be written as sum of two primes.

Twin prime conjecture: There are infinitely many primes p such that $p+2$ is prime. In other words, there are infinitely many couples of primes with a distance of 2 to each other.

Besides being very interesting, prime numbers are also quite useful. For instance it is largely used in public-key cryptography, and in information technology in general.

That's why we decided to work on this subject, testing conjectures on primes.

For the rest of this document, we'll use the following notations and definitions:

Définition 1.1. We define the function $\pi : \mathbb{N} \longrightarrow \mathbb{N}$, $x \longmapsto \pi(x)$ where $\pi(x)$ returns the number of primes not exceeding x. Examples:

$$\begin{aligned}
&\pi(1) = \#\{\} = 0\\
&\pi(2) = \#\{2\} = 1\\
&\pi(3) = \#\{2,3\} = 2\\
&\pi(4) = \#\{2,3\} = 2\\
&\pi(5) = \#\{2,3,5\} = 3\\
&\pi(7) = \#\{2,3,5,7\} = 4\\
&\pi(11) = \#\{2,3,5,7,11\} = 5\\
&\dots
\end{aligned}$$

Définition 1.2. We'll denote $\mathbb{P}$ be the set of all prime numbers.

$$\mathbb{P} = \{2,3,5,7,11,13,17,\dots\}$$

Définition 1.3. We define the function $p : \mathbb{N}^* \longrightarrow \mathbb{P}$, $n \longmapsto p_n$ where p_n returns the n^{th} prime.

$$\begin{aligned}
&p_n(1) = 2\\
&p_n(2) = 3\\
&p_n(3) = 5\\
&p_n(4) = 7\\
&p_n(5) = 11\\
&\dots
\end{aligned}$$

2 Testing conjectures: procedure

2.1 Example

We tested the conjectures by implementing a checking algorithm in the open-source mathematics software SageMath (`http://www.sagemath.org/`). In a first attempt we checked the Goldbach conjecture up to small number in order to familiarise with the Sage syntax.

We wrote a short code to check whether an integer is a Goldbach number. A Goldbach number is an even number $n \geqslant 4$ that can be written as the sum of two primes p and q. One the one hand we looked for the number of unordered prime couples (p, q), as well as the smallest prime p for which there is a prime q, such that $p + q = n$. A little further on, we will see that many conjectures can be treated the same way. Here is the code for the Goldbach conjecture:

```
def isGoldbach(n):
      c = 0
      m = 0
      aux = True
      p = 2
      while p<=n/2:
            if is_prime(n-p):
                  c+=1
                  if aux:
                      m = p
                      aux = False
            p = next_prime(p)
      return [c,m]
```

The next step consisted of implementing algorithms for some more conjectures. Therefore we took some conjectures of the mathematician Zhi-Wei Sun (`http://math.nju.edu.cn/~zwsun/`). We tried to establish a structure in our code, in order to change as little as possible when implementing a new conjecture, i.e. to only change the condition when the conjecture is true. It turns out that there are two recurring types of conjectures, in a way that the algorithms to check conjectures of a same type are very similar. We decided to treat those different types particularly and separately.

2.2 Conjecture type 1

The first type was the most frequent one. Those conjectures had each time the same structure, similar to this one:
Conjecture: $\forall n \in \mathbb{N}_{\geqslant m} : \exists k \in \mathbb{N} : a \leqslant k \leqslant b : \ldots$
We notice that there are several cases (k from a to b) for which this conjecture may be true for a given integer n. One case is enough to prove it for this certain n. However it is also interesting to know for how many cases it is true. Therefore, we decided to look for two values. Once the number of ways c for which the conjecture is true for a given n, and for the smallest k for which the conjecture is true for n, which we denote m in the code. Here's an example of such a conjecture.

Conjecture 2.1.1: $\forall n \in N_{\geqslant 1} : \exists k \in \mathbb{N} : 1 \leqslant k \leqslant n : \pi(kn)$ *is prime.*

```
def check(n):
      aux = True
      c = 0
      m = 0
      for k in range(1,n):
            if is_prime(prime_pi(k*n)):
                  c+=1
                  if aux:
                  m = k
                  aux = False
      return [c,m]

arrayOfC = []      # stores the number of ways for which
                   # the conjecture is true
arrayOfM = []      # stores the smallest case for which
                   # the conjecture is true
bound = 1000       # sets up to which number the conjecture is
                   # tested
for i in range(1,bound+1):
      temp = check(i)
      arrayOfC.append((i,temp[0]))
      arrayOfM.append((i,temp[1]))

def plot(array):
      s = scatter_plot(array, marker=".", facecolor='black')
      s.show()

plot(arrayOfC)
plot(arrayOfM)
```

2.3 Conjecture type 2

The second most common type was about stating that there are infinitely many positive integers, or primes, for which a conjecture is true. Naturally one can see, that this kind of conjecture can't be treated the same way as the first one. It is more difficult to support it, because we can't prove it up to a certain number. However, we can look for the density in which the integers satisfying the conjecture appear. For this reason we looked for two things in this kind of conjecture. First of all we made a list with all the integers fulfilling the conjecture up to a certain bound. Moreover we created a map $\pi_c : \mathbb{N} \longrightarrow \mathbb{N}$, that to a given n associates the number of integers not exceeding n satisfying the conjecture, similar to the π-function. Here's an example

Conjecture 2.15.2: There are infinitely many primes p such that $\pi(p)$, $\pi(\pi(p))$ and $\pi(p^2)$ are all prime.

```
def check(n):
      if is_prime(n) and is_prime(prime_pi(p)) and
                        is_prime(prime_pi(p*p)):
            return True
      else:
            return False

def prime_c(b):
      l = []       # list of integers fulfilling the conjecture
      c = 0        # counts number of integers fulfilling the
                   # conjecture so far
      piMap = []   # defines the map p_c by storing the
                   # couples (i,c) such that pi_c(i) = c
      for i in range(1,b+1):
            if check(i):
                  l.append(i)
                  c+=1
            piMap.append((i,c))
      return [l,piMap]

b = 1000
P = prime_c(b)

def pi_c(n):
      return P[0][n-1][1] # [0] return the list, [n-1] the n-th
                          # element and since the n-th element is
                          # the couple (n,c) where c is what we need,
                          # we have the [1]
```

3 Optimisation of the algorithms

There are ways to optimise certain algorithms. Since our goal was to check each conjecture for bigger numbers than shown in the examples above, this was necessary, as many conjectures simply take too long to check.

3.1 Checking the validity of the conjecture only

Since the goal of this project is to support the veracity of the conjectures, for conjectures of type 1, we eventually decided to omit the computation of the number of ways for which the conjecture was true for a certain n. This accelerated the algorithm drastically. We just looked for the smallest k for which it works, and kept this value. The final code looked like follows:

Conjecture 2.4: $\forall n \in N_{\geqslant 1} : \exists k \in \mathbb{N} : 0 < k < p_n : \pi(kn)$ *is square.*

```
def checkM(n):
      aux = True
      k = 1
      m = 0
      while k<nth_prime(n) and aux:
            if prime_pi(k*n).is_square():
                  m = k
                  aux = False
            k+=1
      return m
```

3.2 Checking the conjecture in intervals

Instead of calculating a conjecture up to a certain number, it's evident to come up with the idea to split the calculations into several intervals. By saving the lists that contain the data supporting (or not) the conjecture during the calculations, we can resume them later. Saving a list L can be done with the following command in Sage:

```
s = Sequence(L)
s.save("nameOfFile")
```

The list will be saved to a file called "nameOfFile.sobj" and can be loaded to a list l with the command:

```
l = load("nameOfFile.sobj")
```

The code can then be changed to save such an object each time the list reaches a certain length. This is useful, because we didn't always know how the speed of the calculations would evolve. It might calculate to 100000 in

a minute, but then slow down quickly. Since those calculations often take hours, and one can't watch them continuously, it is useful, in case the computer may crash, causing the risk to lose all collected data.

3.3 Mathematical optimisation

Since this is a mathematical work, it is clear that at some point, we would try to find a way to improve the algorithm mathematically. This part is not as obvious. Whether there is a mathematical trick to accelerate the calculations is entirely dependent on each conjecture itself. It was hard to find such optimisations and often, as far as we knew, impossible. But when we found them, they produced a high increase of speed. The following example shows such an optimisation.

Conjecture 2.7.2: Any integer n can be written as $k+m$ with $0 < k < m$ such that $\binom{2k}{k} + p_m$ is prime.
In other words: $\forall n \in \mathbb{N}^* : \exists k \in \mathbb{N}^* \, with \, 0 < k < \frac{n}{2} : \binom{2k}{k} + p_{n-k} \, is \, prime.$

Instead of calculating $\binom{2k}{k}$ for every k, which takes a lot of computation, we may find a way to calculate this term using the previous one.
We found that: $\forall k \in \mathbb{N}^* : \binom{2k}{k} = \frac{4k-2}{k} \cdot \binom{2(k-1)}{k-1}$

Proof.

$$\begin{aligned}
\binom{2k}{k} &= \prod_{i=1}^{k} \frac{2k-i+1}{i} \\
&= \frac{2k \cdot (2k-1) \cdot \ldots \cdot (k+1)}{1 \cdot 2 \cdot \ldots \cdot k} \\
&= \frac{2k(2k-1)}{k^2} \cdot \frac{(2k-2) \cdot (2k-3) \cdot \ldots \cdot k}{1 \cdot 2 \cdot \ldots \cdot (k-1)} \\
&= \frac{4k-2}{k} \cdot \prod_{i=1}^{k-1} \frac{2(k-1)-i+1}{i} \\
&= \frac{4k-2}{k} \cdot \binom{2(k-1)}{k-1}
\end{aligned}$$

□

That way, we can highly increase the speed of the algorithm, by getting the next binomial term by multiplying the previous binomial term by a fraction. The result is the following code:

```
def countM(n):
      aux = True
      k = 1
      m = 0
      c = 2
      while k<(n/2) and aux:
            if is_prime(c+nth_prime(n-k)):
                  m = k
                  aux = False
            k+=1
            c = ZZ((4-2/k)*c) # = (4k-2)/k*c
      return m
```

Conjecture 2.14.3: $\forall n \in \mathbb{N}^* : \exists p \in \mathbb{P} : p \in [\pi(n^2), \pi((n+1)^2)]$.
For this conjecture, there is a fast way of calculating the number of primes in the concerning interval, that is $\pi(\pi((n+1)^2)) - \pi(\pi(n^2) - 1)$. The -1 is important, so the interval is closed, and includes $\pi(n^2)$.

```
def checkC(n):
      c = prime_pi(prime_pi((n+1)**2)) - prime_pi(prime_pi(n**2)-1)
      return c
```

3.4 Informatical optimisation

Another natural optimisation is the avoiding of repetition of computation of already computed data. For instance, we have the following conjecture:

Conjecture 2.7.2: $\forall n \in \mathbb{N}^* : \exists k \in \mathbb{N}, 0 \leqslant k \leqslant n-1 : \pi((k+1)n) - \pi(kn)$ *is prime.*

In this example, one can easily see that the term $\pi((k+1)n)$ calculated in one iteration can be used for the following one, replacing there the term $\pi(kn)$, since k increments by 1. This avoids the double computation of a single term. This optimisation is not an individual case and can often be used to make the algorithm faster.

```
def checkM(n):
      aux = True
      m = 0
      k = 1
      p = prime_pi(n)          # pi((k+1)n) for k = 0
      q = 0              # pi(kn) for k = 0
      while k<n and aux:
            if (p-q).is_square():
                  aux = False
                  m = k
            k+=1
            q = p        # the pi(kn) of the next iteration is the
                         # pi((k+1)n) of this one
            p = prime_pi((k+1)*n)
      return m
```

We notice that there is a structure similar to the one in conjecture 2.14.3, where we could save a value to use it in another iteration. The problem here is, it isn't exactly the same term that can be used afterwards. With some fiddling, it's possible though. However, we won't check it for a specific n any more, but for all n up to a certain bound b.

```
def checkC(b):
      arrayOfC = []
      b = 0                          # pi(pi(n)-1) = 0 for n=1
      for n in range(1,b+1):
            a = b                    # contains pi(pi(n)) now
            b = prime_pi(prime_pi((n+1)**2))
            if is_prime(pi(n**2)):
                  a=a-1      # if we don't do this, the
                             # considered inverval will be
                             # open on the left, instead of closed.
                             # contains pi(pi(n)-1) now
            c = b-a
            arrayOfC.append((n,c))
```

4 Plotting the conjectures

The conjectures that were tested can be represented by a graph. The way the graph evolves helps us to understand the conjecture better and support its validity. Depending on the type of conjectures there are several graphs that we can consider.

4.1 Conjectures of type 1

For this kind of conjecture, let's plot the following conjecture:

Conjecture 2.1.1: $\forall n \in N_{\geqslant 1} : \exists k \in \mathbb{N} : 1 \leqslant k \leqslant n : \pi(kn)$ *is prime.*

Let's remember, that for this kind of conjectures, we can look for a given positive integer n for the number of ways for which the conjecture is true, as well as the smallest case for which it is, which we will denote $c(n)$ and $m(n)$ respectively.

Graph 1 The first way of plotting this type of conjecture is to draw the graph of the function that to a given n associates $c(n)$.

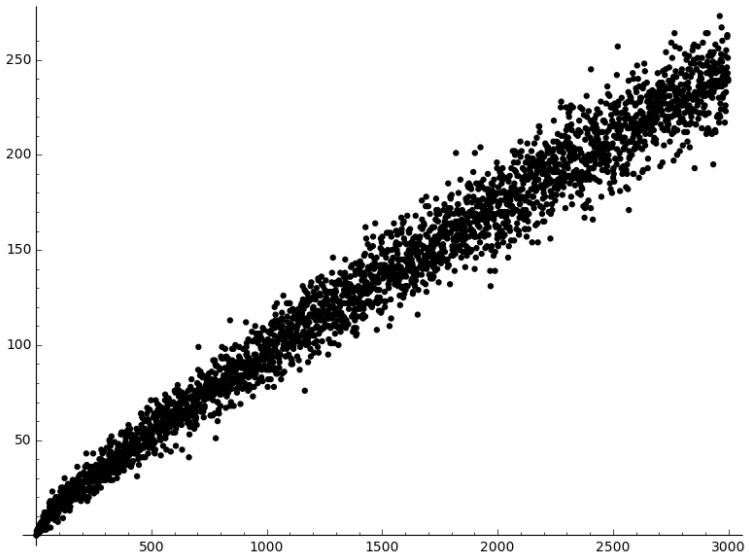

Figure 1: Graph $\{(n, c(n))|n \in \mathbb{N}^*_{\leqslant 3000}\}$ where $c(n) = \#\{k \in \mathbb{N}^*_{\leqslant n} \,|\, \pi(kn)\, is\, prime\}$

Graph 2 The second way of plotting this type of conjecture is to draw the graph of the function that to a given n associates $m(n)$.

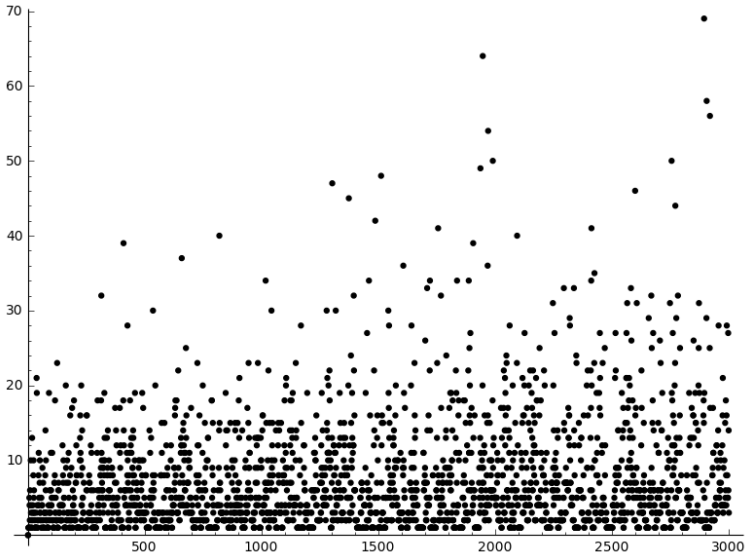

Figure 2: Graph $\{(n, m(n))|n \in \mathbb{N}^*_{\leqslant 3000}\}$ where $m(n) = min\{k \in \mathbb{N}^*_{\leqslant n} \,|\, \pi(kn)\, is\, prime\}$

4.2 Conjectures of type 2

For this kind of conjecture, let's plot the following conjecture:

Conjecture 2.15.2 There are infinitely many primes p with $\pi(p)$, $\pi(\pi(p))$ and $\pi(p^2)$ all prime.

For this type of conjectures we looked for each positive integer n up to a certain bound that satisfy the conjecture, but in the same time created a map $\pi_c : \mathbb{N}^* \longrightarrow \mathbb{N}$, $n \longmapsto \pi_c(n)$, where $\pi_c(n)$ returns the number of positive integers not exceeding n satisfying the conjecture.

We plotted this kind of conjecture, by drawing the graph of the function π_c. That way one can see the density of the numbers satisfying the conjecture in $\mathbb{N}$.

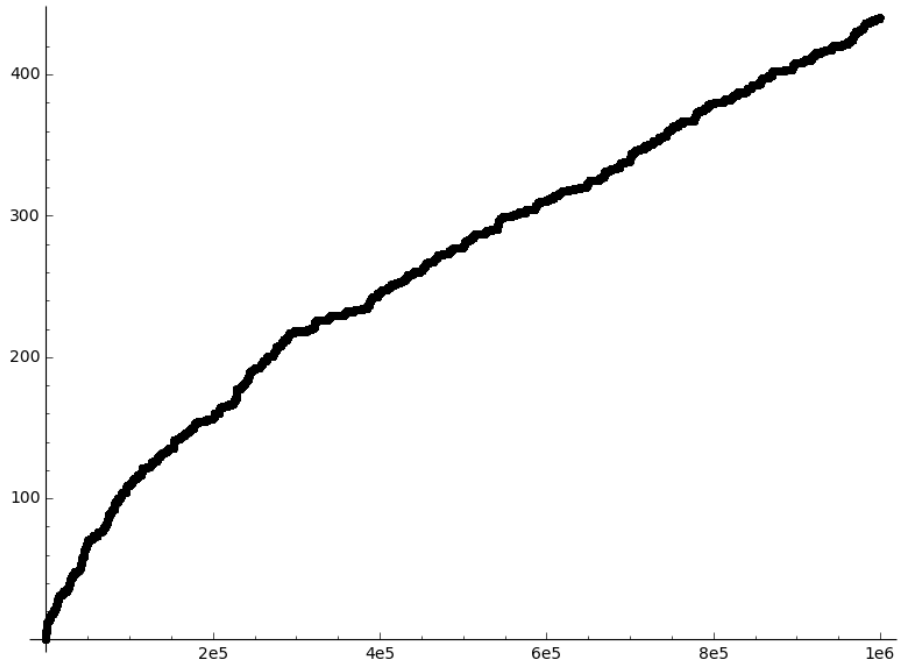

Figure 3: Graph $\{(n, \pi_c(n))|n \in \mathbb{N}^*_{\leqslant 10^6}\}$ where $\pi_c(n) = \#\{p \in \mathbb{P}_{\leqslant n} \,|\, \pi(p),\, \pi(\pi(p))\, and\, \pi(p^2)\, are\, prime\}$

5 Conclusion

We checked the conjectures as far as possible. And all our results clearly support Zhi-Wei Sun's claims. None of our collected data contradicts his conjectures, except for one minor detail, that we'll explain further down.

5.1 Checking the validity

For the conjectures of type 1, we saved for each n, as described before, the smallest case m. Afterwards we went through that list to check whether there are any 0's. That means we searched all n for which the conjecture was not verified. If L was the list, the code would be the following:

```
for i in range(1,len(L)+1):
    if L[i-1] == 0:
       print i
```

We did this for all conjectures of type 1 and found no dissent with the conjectures.

5.1.1 Exception: conjecture 3.21.1

For this conjecture we found a minor deviation. The conjecture is the following:

Conjecture 3.21.1: $\forall n \in \mathbb{N}_{>5} : \exists k \in \mathbb{N}^*_{<n} : 2k+1 \in \mathbb{P} \wedge p_{kn} + kn \in \mathbb{P}$

Here are our results, where $m(n) = min\{k \in \mathbb{N}^*_{<n} \,|\, 2k+1 \in \mathbb{P} \wedge p_{kn}+kn \in \mathbb{P}\}$

n	$m(n)$	n	$m(n)$
1	0	11	2
2	1	12	2
3	2	13	2
4	1	14	3
5	0	15	0
6	1	16	2
7	6	17	2
8	3	18	1
9	2	19	6
10	0	20	5

As we can see, the conjecture is wrong for the numbers 10 and 15, while the conjecture states that it should be true for all $n > 5$. We checked again and the conjecture hasn't been changed yet. However the conjecture seems to be

true for all $n > 15$. We think that it's probably a typing error. There is no reason to say the conjecture is wrong, just because there is a small number for which it's wrong.

5.2 Analysing the plot

Previously we plotted the conjectures, and although they don't serve as proof for the conjectures, they strongly support them. Here's again one of the graphs for conjecture 2.1.1.:

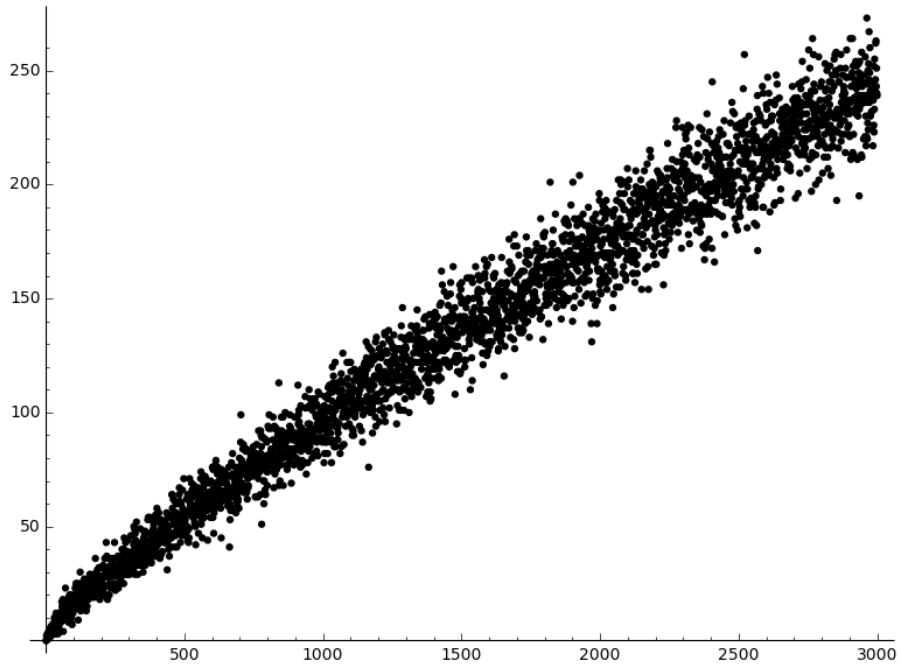

Figure 4: Graph $\{(n, c(n))|n \in \mathbb{N}^*_{\leqslant 3000}\}$ where $c(n) = \#\{k \in \mathbb{N}^*_{\leqslant n} \,|\, \pi(kn)\, is\, prime\}$

We observe that the number of ways increase more and more. It gives us the impression, that the bigger we choose n, the bigger $c(n)$ probably is. It doesn't prove the conjecture, but still gives it a good support. This is the case for most of the conjectures we treated of this type.

It was trickier to analyse the veracity of the second type of conjecture. Let's take for example the following conjecture.

Conjecture 2.15.2: There are infinitely many $p \in \mathbb{P}$ such that $\pi(p)$, $\pi(\pi(p))$ and $\pi(p^2)$ are all prime.

A method is to study the π_c-function we build. We tried to find a function f, such that:

$$\pi_c(x) \sim f(x) \; as \; x \to +\infty$$

$$and$$

$$\lim_{x \to +\infty} f(x) = +\infty$$

If we found such a function, we would have strong evidence for the conjecture being true. In most cases it's a logarithmic function. This is due to the fact that:

$$\pi(x) \sim \frac{x}{\log(x)} \; as \; x \to +\infty$$

But in this case, didn't find such a function. However we were able to major π_c. We have no proof on this, but computed it to 10^6, and it seems that:

$$\lim_{x \to +\infty} \frac{\pi_c(x)}{log(x)} = +\infty$$

x	$\frac{\pi_c(x)}{log(x)}$
10	0.000
100	0.217
1000	1.013
10000	2.280
100000	9.468
250000	14.367
500000	21.109
750000	26.686
800000	27.883
900000	29.759
1000000	31.848

This shows, that π_c grows faster than log. And since

$$\lim_{x \to +\infty} log(x) = +\infty$$

we have some evidence supporting the conjecture.

6 Table

Here are the conjectures we tested (the enumerations corresponding to Zhi-Wei Sun's in his *60 open problems on combinatorial properties of primes*) alongside the bounds until which we tested them.

Tested conjectures	Tested at number	Tested conjectures	Tested at number
Conjecture 2.1.1	5600000	Conjecture 3.6.2	167324
Conjecture 2.2.1	10000	Conjecture 3.7.1	8000000
Conjecture 2.3	29227	Conjecture 3.7.2	1000000
Conjecture 2.4	26971	Conjecture 3.8.1	616707
Conjecture 2.5.1	1000000	Conjecture 3.8.2	969007
Conjecture 2.5.2	100000	Conjecture 3.9	800000
Conjecture 2.6.1	1000000	Conjecture 3.10.1	700000
Conjecture 2.6.2	97475	Conjecture 3.10.2	600000
Conjecture 2.7.2	132000	Conjecture 3.11.1	1000000
Conjecture 2.8.1	1000000	Conjecture 3.11.2	650000
Conjecture 2.8.2	90000	Conjecture 3.12.1	150000
Conjecture 2.9.1	230000	Conjecture 3.12.2	100000
Conjecture 2.9.2	10000	Conjecture 3.13.1	200000
Conjecture 2.11	22016	Conjecture 3.15.1	100000
Conjecture 2.12.1	1000000	Conjecture 3.15.2	100000
Conjecture 2.12.2	16358	Conjecture 3.15.3	50000
Conjecture 2.14.3	415633	Conjecture 3.18	300000
Conjecture 2.15.1	100000	Conjecture 3.19	500000
Conjecture 2.15.2	1000000	Conjecture 3.21.1	150000
Conjecture 2.16.1	1000000	Conjecture 3.22.1	752669
Conjecture 2.16.2	1000000	Conjecture 3.23.1	1000000
Conjecture 2.16.3	120000	Conjecture 3.23.2	300000
Conjecture 2.17.1	1000000	Conjecture 4.1.1	43552
Conjecture 2.17.2	1000000	Conjecture 4.1.2	43497
Conjecture 2.18.2	1000000	Conjecture 4.1.3	26980
Conjecture 3.1	421745	Conjecture 4.2.1	51030
Conjecture 3.2	242325	Conjecture 4.3.1	100000
Conjecture 3.3	333000	Conjecture 4.3.2	120051
Conjecture 3.4	100000	Conjecture 4.4.1	40000
Conjecture 3.6.1	212446	Goldbach Conjecture	5000000