Generating New Prime Numbers via Symmetric Triplets: 20 New 300-Digit Primes in 40 Seconds

Author: Dobri Bozhilov

Abstract:

This paper presents a novel method for discovering large prime numbers using symmetric triplets, based on an extension of the Goldbach hypothesis. In a previous paper (DOI: 10.5281/zenodo.15198172, https://ai.vixra.org/abs/2504.0029), we demonstrated that every positive integer is surrounded by a symmetric pair of prime numbers. In this paper, we examine a specific case of that symmetry—where the center of symmetry is itself a prime number. Thus, prime numbers can be grouped into "triplets": two primes equidistant from a third one. The key idea is to analyze the most common differences between primes and to scale these differences based on the distribution density, enabling efficient discovery of large primes with far fewer checks compared to brute-force methods.

1. Introduction

Prime numbers are fundamental in number theory and essential to modern cryptography. Generating large primes is crucial for RSA and other encryption systems, but traditional methods are slow and computationally intensive. This work presents a new approach that significantly reduces the search space using symmetry and statistical inference.

2. Theoretical Basis

The method builds on an extension of Goldbach's conjecture, analyzing the symmetry of primes around a number n. It assumes that for any integer n, including very large ones, there exists a symmetric pair of primes p1 and p2 such that:

$p1 + p2 = 2n \rightarrow n = (p1 + p2)/2 \rightarrow p = 2n - s$

where s is a known prime. This allows the generation of candidate primes through symmetric transformation around n. But this is the general principle that applies to every positive integer. Here, we focus on applying it specifically to known prime numbers, with the goal of finding the "symmetric" ones around them. This reduces the number of required checks.

3. Algorithm and Implementation

The algorithm consists of:

1. Selecting a large base prime n (e.g., verified using FactorDB).

2. Using the most frequent prime gaps up to 1 million: 6, 12, 30, 60, 90, 210, 420, 630, 840, 1050.

3. Scaling each gap to match the size of n using the logarithmic approximation based on Riemann:

coefficient = li(n) / li(10^6)

4. For each scaled gap d, computing values in a 1% extended interval d ± 1%, and checking up to 200 million candidates (100 million in each direction). These are not full interval scans but samples spread uniformly across the range. These values are chosen because the scaling, even when based on the Riemann Hypothesis, cannot predict a new prime number with absolute precision. There is an inevitable element of inaccuracy, but as demonstrated in this work, a ±1% interval is sufficient to discover a new prime. Moreover, it is not necessary to check the entire interval—only 200 million values evenly distributed within it.

5. Checking whether p1 = n - d and p2 = n + d are both prime.
6. Running the process in parallel and stopping after discovering a predefined number of new primes.

4. Experiment and Results

Using a 300-digit base prime n, and 10 scaled gaps with 100 million values per gap, we obtained:

- ~2 billion planned checks

- Runtime: 40 seconds (on an 8-core Google Cloud server), stopped early after 10 prime pairs were found. That is, in reality, 2 billion checks were not actually performed.

- New primes discovered: 20 (in 10 symmetric pairs)

- Confirmed using gmpy2.is_prime and FactorDB

5. Discussion

Results show that even with modest hardware, proper selection of symmetric candidates and statistical scaling can yield large new primes efficiently. The method is suitable for practical cryptography using primes with 500–1000 digits. Unlike traditional probabilistic methods, this approach offers guaranteed prime numbers in seconds. Although initial screening uses Miller-Rabin, the final numbers were confirmed as primes by independent databases. This opens the door for generating deterministic cryptographic keys in real applications.

6. Scaling Up

The idea of "symmetric primes" would undoubtedly lead to faster discovery of prime numbers across any range when compared to brute-force methods. However, in the higher ranges (with millions of digits), the method becomes impractical due to the immense computational cost of verifying each candidate. Even with a preliminary Miller-Rabin test, checking candidates with millions of digits would take an extremely long time. Additionally, the number of checks must be scaled. For instance, while a 100-million-value interval may be sufficient around each candidate at the 300-digit level, applying a density-based scaling factor means that around 130 trillion checks would be needed at the 20-million-digit level. This alone is not unreasonable, but the main issue is the time required for each primality test. At such digit lengths, each test would take a prohibitive amount of time. Even with supercomputers, it would require thousands or millions of years. Again, this is still faster than brute-force, but in practice, it remains infeasible. Therefore, the method is most effective for practical cryptographic purposes, where primes with 500–1000 digits are used. In that range, truly prime numbers can be generated within seconds.

7. Empirical Results

The following 20 new 300+ digit prime numbers were discovered in 40 seconds on an 8-core cloud instance. All have been submitted and verified as prime by FactorDB:

Base prime number:

p =
10888867115841589312034362781372485550827926615711719697703545903484132043015623721673322604207052785909750803572004491817501408260167607773788073677923712749770404015724196960297257347124688307337323887078012467815974678594992662455133016707902302248690343823354571922270211042685572248211792573079822078133

Found symmetric primes:

p1 =
10888865939750240395484783144961623599374394243114974322856044975284319294130471251521233782691042001304666190109216203204066004803628252097597973879356649536351615234430501974476982337349386990758500074520283654451294249522168119685433020477800436160102798566007928137624308835075640690108414338431583682987 7

p2 =
10888868291932938228583942417783347502281458988308465072551046831683944791900776191825411425723063570514835417034792780430936811716706963449159640856228093013602465568714337417582469132075550670708964702895318839111900686196830412805593582863780024289635270198662986246829733373461473759533944176184380732638 9

……..

p1 =
10888865948548072097312704589152609480866538748070220435796763716842800967112074102214126066399330673234122912449871473703720702357035695333026562654510858919603074418443638794667615311196347070190893568743272846212161655839528984509640517084931199270181245265222424489272637330225507380309600929472317915563 7

p2 =
10888868283135106526756020973592361620789314483353218959610328090125463118919173341132519142014774898585378694694137509931282114163299520213731052081073883630351006384701200597391836158228590591276571208672329647351033279879469547981386086256649261179556823499448489895181404878311607069332757585143646500062 9

……..

p1 =
10888865954394048221794281294296935185628901867991722780300086504434814117103791987315999817543934421795689536945902168115945251333828857172146480112183484366476525610077142458068911206572902459275775302357871816202654315244239238866587069350534483241274498203029953804996411175478739849874319618719137300907 7

p2 =
10888868277289130402274444268448035916026951363431716615107005302533449968927455456030645390870171150023812070198106815519057565186506358374611134623401258183477555193067696933990540262852035202191689475057730677360540620474759293624439533991045977208463570561640960579457631033058374599768038895896827114718 9

……..

p1 =
10888865948503100256294955200906501332659088941314652047095441484798093223582699086756524794801225356299031828218858683360062239302727104497193851660599553418499715699354914444413717106506638259331395134758864215439292969465134402892912904891420538047531682874102012919297086992442914642145598541737952182699 7

p2 =
10888868283180078367773770361838469768996764290108787348311650322170170862448548356590120413612880215520469778925150300274940577217608111049563763074985189131454365103789924947645734362918299402136069642656738278123901966253864129598113698450159922402206385890568901465156955216094199807496759972878012232926 9

……..

p1 =
10888865948500214537158081811609460768894062411860645843197339364469257262817167143790708576101538700986493868794612639514530792770456457106628514720299282157793603200801134897663774534969112215427258101400600205367677761047005416661658433359912604151976653693218425808883833451120048301544435976475660157355 7

p2 =
10888868283182964086910643751135510332761790819562793552209752442499006823214080299555936632312566870833007738349396344120472023749878758440129100015285460392160477602343704494395676934455825446040206676015002288195517174671993115829368169981667856297761415071452488575570208757417066148097922538140304258270 9

……..

p1 =
10888865945568000573447026560735328469816517181921545132758135178544185955586982414869702895112791416993727442126644544840709475566835853237552064863145243490491839980325820256709823309191986627754969893290318022874980878258304352922128689119262684812938162273533065988526770453005874297998726343495236826334 9

p2 =
10888868286115178050621699002009642631839336049501894262648956628424078130444265028476942313301314154825774165017364438794293340953499362309205549872439499059462240822819019135349628160232951033712494884125284470688214057460694179568897914222317775636799906491137848395927271755531240151643632171120727589291 7

……..

p1 =
10888865933835472049146744706911768774393749161426445012740551290189091522704297721114501491409311891055409090948240527209402364332507814030374024962437914844746006427017478677913167232450994856484762401604652684897611168889793223935430828967758503286800947400186226516666547909460508501308108652962146530526 9

p2 =
10888868297847706574921980855833202327262104069996994382666540516779172563326949722232143717004793680764092516195768456425600452187827401516383589773146827705208074376127360714146284236973942804982702375810949 8

086655837668292053085555957743738219571629371213644846878678674942990766059483342498616538178850997

……..

p1 = 1088886593380809138680738648485185502186708766613422020745150252858775113594088902874451311506026180262235730168429695608776827585019550812671210136936978077588589989290609290908778549792657880574960621695524451972852981155593570441565009247859565873320478217389590181174095222369527426522943002253107093219 09

p2 = 1088886829787508723726133907789311607978876556528921918795558927838051295009035841460213209335384376919714430545971202754723454067013970742004551336621496177406818091023874648297166597149835885571785856046035797383466512416306282807537651086298480171653328659077501257271308998484184018441292849208489348343 57

……..

p1 = 1088886593967194798923537696976972294199500676051319108043467250323851262437622184183374759090345293005585099073787202375468919117736459370912160315450610821424528450259862812656225559740156303770851156833923538482393792298888662552465869702839215917656705045640969204426218752177534313449605769202209716348 69

p2 = 1088886829201123063483334859297524815966084647091024831497241930372975146165502560151289761751065264176365061640613695988031362534297062183763601158107863433570879630054621126549719587202337462375895320907636710873925701273011190696636790631318830127317101830826122234019185468676177131514630082259386725213 97

……..

p1 = 1088886593379772001978526492027352539893066694319267208369337430595134602611913375079991929762502818667941800759070614293554636993071412356504310924242130481257399876455940375901651271057288993864853362429688346204461128864637972487986956824499070728691711580588849478889702109578708604778421312276793561622 29

p2 = 1088886829788545860428346064247144570272518628823076731171371750101691

80599121136925467259107890773851400835995533028406994564465896210919817145054931634377373800820385854356330429387588520477228189311531187190315185836470726188076111570350965897531628209529587824195955570211127500284018581453918480287994037

8. Code Used

The following Python code was used for the purposes of the experiment:

```
import gmpy2

from sympy import li

from time import time, sleep

from multiprocessing import Process, Queue, cpu_count, Value, Lock

from numpy import linspace


# === SETTINGS ===

BASE_PRIME =
gmpy2.mpz("10888867115841589312034362781372485550827926615711719697703545903484132043015623721673322604207052785909750803572004491817501408260167607773378807367792371274977040401572419696029725734712468830733732388707801246781597467859499266245513301670790230224869034382335457192227021104268557224821179257307982207813 3")

BASE_SAMPLE = gmpy2.mpz(10**6)

li_now = li(BASE_PRIME)

li_sample = li(BASE_SAMPLE)

scaling_factor = float(li_now / li_sample)

COMMON_DIFFS = [6, 12, 30, 60, 90, 210, 420, 630, 840, 1050]

scaled_diffs = [int(d * scaling_factor) for d in COMMON_DIFFS]

INTERVAL_PERCENT = 0.01

SAMPLES_PER_DIFF = 100_000_000  # 100 million per difference

print("Safe Scan mode: Scanning 100 million values per gap using parallel cores.")
```

```
# ===WORKER FUNCTIONS ===
def check_segment(diff_value, start_idx, end_idx, steps, queue, counter, lock):
    delta = int(diff_value * INTERVAL_PERCENT)
    start_d = diff_value - delta
    end_d = diff_value + delta
    for i in range(start_idx, end_idx):
        with lock:
            if counter.value >= 10:
                return
        interp_d = int(start_d + i * (end_d - start_d) / steps)
        p1 = BASE_PRIME - interp_d
        p2 = BASE_PRIME + interp_d
        if gmpy2.is_prime(p1) and gmpy2.is_prime(p2):
            with lock:
                if counter.value < 10:
                    counter.value += 1
                    queue.put((int(p1), int(BASE_PRIME), int(p2)))

# === WRITER PROCESS ===
def writer(queue):
    with open("found_primes.txt", "a") as f:
        while True:
            item = queue.get()
            if item == "DONE":
                break
            p1, p, p2 = item
```

```
            f.write(f"p1 = {p1}\np  = {p}\np2 = {p2}\n\n")


    # === START ===
    if __name__ == "__main__":
        start_time = time()
        cpu_cores = cpu_count()

        q = Queue()
        counter = Value('i', 0)
        lock = Lock()
        writer_process = Process(target=writer, args=(q,))
        writer_process.start()

        for diff_index, d in enumerate(scaled_diffs):
            print(f"\n➡ Starting difference #{diff_index + 1} ({d})")
            segment_size = SAMPLES_PER_DIFF // cpu_cores
            workers = []

            for core in range(cpu_cores):
                start_idx = core * segment_size
                end_idx = (core + 1) * segment_size if core < cpu_cores - 1 else SAMPLES_PER_DIFF
                p = Process(target=check_segment, args=(d, start_idx, end_idx,
SAMPLES_PER_DIFF, q, counter, lock))
                workers.append(p)
                p.start()
```

```
    for p in workers:
        p.join()

    with lock:
        if counter.value >= 10:
            print("⛔ 10 pairs found, stopping the search.")
            break

q.put("DONE")
writer_process.join()

end_time = time()
print("\n--- RESULTS ---")
print(f" Completed in {end_time - start_time:.2f} seconds ")
```

9. Future Work

1. Testing on more powerful supercomputers to validate scalability to larger ranges.

2. Exploring application of the method to Mersenne numbers, which are easier to verify and may enable breakthroughs at higher levels without massive infrastructure.

10. Conclusion

The presented method enables discovery of large prime numbers using symmetric triplets and scaled prime gaps. It demonstrates practical efficiency and has potential to extend beyond 10^500 or 10^1000.

11. Acknowledgments

The author expresses gratitude to GPT-4 (OpenAI ChatGPT), whose interactive assistance helped resolve many of the technical questions and computations.

12. References:

- Goldbach's conjecture

- Prime number theorem (logarithmic behavior)

- Riemann prime counting function (log integral approximation)

- Miller-Rabin primality test (probabilistic)

- gmpy2.is_prime — fast primality check in Python

- FactorDB for prime number confirmation

- Previous publication: https://ai.vixra.org/abs/2504.0029

(DOI: 10.5281/zenodo.15198172)

==========

Contact: zadrugata2019 (at) gmail (dot) com

Phone: ++359 887 467 818