Original Research Article

Approach Derived from Lévy Flight to Identify a Divisor of an Odd Composite Integer

Abstract: In accordance with the distribution law of composite integers possessing a divisor of the odd semi-prime $N = pq$ with $2 < p < q$, and considering the characteristics of Lévy flight (LF), this paper presents an innovative approach that integrates LF with specific local search techniques (LS) to identify a divisor of N. This method exhibits minimal space complexity and, in theory, achieves logarithmic time complexity in optimal scenarios. A comprehensive framework for the approach is outlined, alongside the introduction of a simple-random-walk-blended LF (SRW-blended LF) algorithm. Key issues pertinent to the implementation of the algorithm are thoroughly analyzed and elucidated. Experiments conducted to factorize semi-primes demonstrate that the newly developed approach, implemented using Matlab software, yields highly satisfactory results

Key words: Integer factorization, randomized algorithm, Lévy flight, local search.

1. Introduction

The problem of integer factorization has been a research topic in the field of mathematics for centuries. Since the RSA cryptosystem came into being, it has gotten particular attention in cryptography because the RSA cryptosystem is highly dependent upon whether the problem is solved or not. Once the problem is settled down, the RSA cryptosystem is no longer safe. Accordingly, cryptanalysts and mathematicians keep finding efficient ways to solve the problem. From the books [1] - [6] and the overview papers [7]-[17], it is seen that people since 1970s have developed more than ten methods to factorize a composite integer, including the Pollard –ρ method, the continued fraction method, the quadratic sieve method, the elliptic curve method, the number field sieve method, and even the quantum method. Nevertheless, literatures of overviews and researches [18]-[26], which were published in the last 5 years, exhibit that new efficient methods for solving the problem are still in need and in development.

Analysis of current methods shows that searching process forms their core operations: searching a divisor of an composite integer in a certain set of integers. For example, the famous Pollard-ρ method is essentially a search on the Folyd circle. Therefore, there were researches applying various searches to solve the problems. Ahmed T Sadiq [27] and Ade Candra [28] applied Tabu search respectively in 2009 and 2017 , Choudhury Bhargab [29] and Wang X [30] used general search respectively in 2015 and 2017, Dash Avinash [31] in 2018 performed an exact quantum search, Rutkowski Emilia [32] and Mahadee Al Mobin[33] adopted respectively search of genetic algorithm(GA) in 2020 and 2024, Mohit Mishra[34] as well as Fagin Barry S [35] and Hittmeir Markus [36] utilized heuristic search respectively in 2016, 2021, and 2023. These researches indicate that it is being a trend to apply intelligent search in developing method of factoring integers.

Lévy flight (LF) [37] was originally introduced in 1937 by the French mathematician named Paul Lévy (1886-1971). It is a statistical description of motion attributed to random walk. The LF was applied to be a search method [38][39] in many fields including analysis of financial market, optimization of engineering problems, simulation of animal foraging, and description of molecule motion [37][40][41][42][43]. Literatures show that the LF is optimal for 'blind search' (i.e., uninformed search), which finds without any knowledge or information of the

stationary targets that are randomly and sparsely distributed, and can maximize the efficiency of the search process in uncertain environments. Yet, no report has been found to apply the LF on integer factorization although there were literatures applying random walks on the issue[44][45]. This paper hence makes an investigation on applying the LF on finding a divisor of a positive composite odd integer.

The paper consists of five parts. The first one is this introductory part, the second one introduces the preliminary foundation for later parts, the third one briefly introduces the mathematical foundations necessary for later sections, the fourth one gives research objective and main results as well as numerical experiments, and the last part in the conclusion.

2. Terminologies, Symbols and Notations

Here presents necessary symbols, notations, definitions for later investigation.

Symbol $A \Rightarrow B$ means conclusion B can be derived from condition A. $A \Leftrightarrow B$ means both $A \Rightarrow B$ and $B \Rightarrow A$. An integer interval $[a,b]$ means the set of all the integers bounded with integers a and b with $a < b$; for example, integer interval $[5,9] = \{5,6,7,8,9\}$. If an integer interval contains integer x, that integer interval is called a *host interval* of x. If a two-dimensioanl region (zone, area) contains integer point P, that region (zone, area) is called a *host region* (zone, area) of P. If d is a divisor of integer h, h is called a *host number* of d or a number hosting d and denoted by symbol h^d. For positive integers a and b, integer $g_a^b = |a-b|-1$ is called the gap between a and b. Symbol G^x means a gap taking value x. Symbols $\lfloor x \rfloor$ and $\lceil x \rceil$ are respectively the floor and ceil functions such that $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$ or $x = \lfloor x \rfloor + \{x\} = \lceil x \rceil - 1 + \{x\}$, where $\{x\}$ is the fractional part of x.

3. Math Foundations

This section summaries mathematical foundations for later sections, including lemmas to describe the distributions of the hosts hosting a divisor of another composite integer, principle to make the hosts more denser a distribution and the LF.

3.1. Lemmas

Lemma 1[46]. Let q be a positive odd number, $S = \{a_i \mid i \in Z^+\}$ be a set composed of consecutive odd numbers; if $a_\alpha \in S$ is a host of q, then so it is with $a_{\alpha+q} \in S$.

Lemma 2[47]. Given an odd integer $N = pq$ with divisors p and q satisfying $2 < p < q$; let $\omega = \left\lfloor \frac{\sqrt{N}}{p} \right\rfloor$. Then the integer interval $I_N = [\lfloor \sqrt{N} \rfloor, \quad N-1-\lfloor \sqrt{N} \rfloor]$ contains $p+q-2\omega-1$ hosts of N's divisors if $p \mid \lfloor \sqrt{N} \rfloor$; otherwise it contains $p+q-2\omega-2$ hosts of N's divisors.

Lemma 3[48]. Let $N = pq$ be an odd integer and integer interval $I_N = [3, N-1]$, where p and q are odd integers with $1 < p < q$ and $(p,q) = 1$; then for each pair of h^p and h^q in I_N satisfying $1 < h^p, h^q < \frac{N-1}{2}$, it holds

$$g_{h^p}^{h^q} = g_{N-h^p}^{N-h^q}$$

and there exists a pair, still say h^p and h^q satisfying $g_{h^p}^{h^q} = g_{N-h^p}^{N-h^q} = 0$.

Lemma 4[49]. Given an odd integer $N = pq$ whose divisors p and q are odd integers satisfying $(p,q) = 1$ and $q = p + r$ with $1 < r < p$; let $\omega = \left\lfloor \frac{p}{r} \right\rfloor, \varsigma = \left\lfloor \frac{p+1}{2\omega} \right\rfloor - 1$, and integer intervals be given by

$$
\begin{aligned}
&I_N = [1, N-1],\\
&I_0 = [q, (\omega-1)q],\\
&I_1 = [\omega q, (2\omega-1)q],\\
&\ldots,\\
&I_k = [k\omega q, ((k+1)\omega-1)q],\\
&\ldots,\\
&I_\varsigma = [\varsigma\omega q, ((\varsigma+1)\omega-1)q],\\
&I_{\varsigma+1} = [(\varsigma+1)\omega q, (\frac{p-1}{2})q].
\end{aligned}
$$

Then for $0 \le j \le \varsigma$, G^{p-1} exists between the end of I_j and the start of I_{j+1} except for the one near and out of the end of $I_{\varsigma+1}$. There are at least $2\varsigma+1$ such gaps distributed symmetrically in I_N.

3.2. Distribution Traits of the Hosts In Hosting Interval

Let $N = pq$ be an odd integer with divisors satisfying $2 < p < q$; then from 3 to *N*–1 there are $p-1$ hosts of *q* and $q-1$ hosts of *p*, totally $p+q-2$ hosts in the integer interval $I_N = [3, N-1]$. Referring to Lemma 3, the gaps between two hosts are distributed symmetrically in I_N and the minimal gap is 0. This property shows that all the hosts are distributed sparsely in a global scope while some are locally accumulated somewhere. Let P_{host} be the probability to pick randomly in I_N an integer that is a host of *N*'s divisors; then

$$P_{host} = \frac{p+q-2}{N-3}$$

It surely follows

$$P_{host} < \frac{p+q+1}{N} = \frac{1}{p}+\frac{1}{q}+\frac{1}{N} < \frac{2}{p}+\frac{1}{pq} < \frac{1}{p}(2+\frac{1}{q}) < \frac{1}{p}(2+\frac{1}{5}) = \frac{11}{5p} \approx \frac{2}{p}$$

indicating one successful pick for a big *p* is like finding proverbial needle in the haystack.

3.3. Densification of Hosts

Paper [50] proposed a method to make the hosts a denser distribution by Cartesian subtraction and obtained a sequence of the form

$$
T_N^g = \left\{
\begin{array}{cccccc}
\underbrace{p_b+1, \cdots, p_b+1}_{g\ times}, & & & & & \\
\cdots & \cdots & & & & \\
\underbrace{e, \cdots, e}_{g\ times}, & \cdots & \underbrace{p_b+1, \cdots, p_b+1}_{g\ times}, & & & \\
\cdots & \cdots & \cdots & \cdots & & \\
\underbrace{N-1, \cdots, N-1}_{g\ times}, & \cdots & \underbrace{e, \cdots, e}_{g\ times}, & \cdots & \underbrace{p_b+1, \cdots, p_b+1}_{g\ times} &
\end{array}
\right\}
$$

where $p_b < p$ is a lower bound of *p*, and g is the multiplicity of the terms.

By this means, a host of *N*'s divisors accumulates together to *g* times every time it occurs and the total number of the hosts is increased by *g* times. Hence it increases the probability of a successful pick. Paper [47] investigated the case *g*=1 in detail. It demonstrated that taking $p_b = \lfloor \sqrt{N} \rfloor$ and $p_u = N-1-\lfloor \sqrt{N} \rfloor$ led to a *pq*-band Ξ that has a more concentrated distribution of the hosts of *N*'s divisors, where Ξ is given by

$$\Xi=\left\{\begin{array}{cccccccccc}
p_b+1, & & & & & & & & & \\
p_b+2, & p_b+1, & & & & & & & & \\
\vdots & p_b+2, & p_b+1, & & & & & & & \\
\vdots & \vdots & p_b+2, & p_b+1, & & & & & & \\
p_u-1, & \vdots & \vdots & p_b+2, & \ddots & & & & & \\
p_u, & p_u-1, & \vdots & \vdots & \ddots & p_b+1, & & & & \\
 & p_u, & p_u-1, & \vdots & \vdots & p_b+2, & p_b+1, & & & \\
 & & \ddots & p_u-1, & \vdots & \vdots & p_b+2, & p_b+1, & & \\
 & & & p_u, & p_u-1, & \vdots & \vdots & p_b+2, & p_b+1, & \\
 & & & & p_u, & p_u-1, & \vdots & \vdots & p_b+2, & p_b+1
\end{array}\right\}$$

It is seen that, in Ξ each ridge (defined in paper [47]) is identified with a unique integer. Hence Ξ is said to be a uniform set.

3.4. Simple Random Walk, Brownian Motion and the LF

Let $X_1, X_2, ...,$ and X_n be identically distributed (*iid*) random variables, a random walk S_n is defined by

$$S_n = \sum_{i=1}^{n} X_i = S_{n-1} + X_n$$

The *idd* random variables are alternatively called random steps that can be intuitively shown with their 'footprints'. In the case that the random steps are drawn from the uniform distribution, S_n is a simple random walk (SRW), whereas, it is the LF in the case that the random steps are drawn from the isotropic symmetric Lévy stable distribution. Referring to [51][52], the characteristic function of the LF is given by

$$F(k) = e^{-\alpha|k|^{\beta}}, 0 < \beta \le 2$$

where real α is a scale coefficient and real β is the Lévy index.

With inverse Fourier transform of $F(k)$, the probability density function (pdf) of the Lévy stable distribution is calculated by

$$L(s) = \frac{1}{\pi}\int_0^{\infty} \cos(ks)e^{-\alpha k^{\beta}} dk$$

Let $\Gamma(z)$ be the Gamma function $\Gamma(z) = \int_0^{\infty} t^{z-1}e^{-t}dt$; when s is large, $L(s)$ can be estimated by

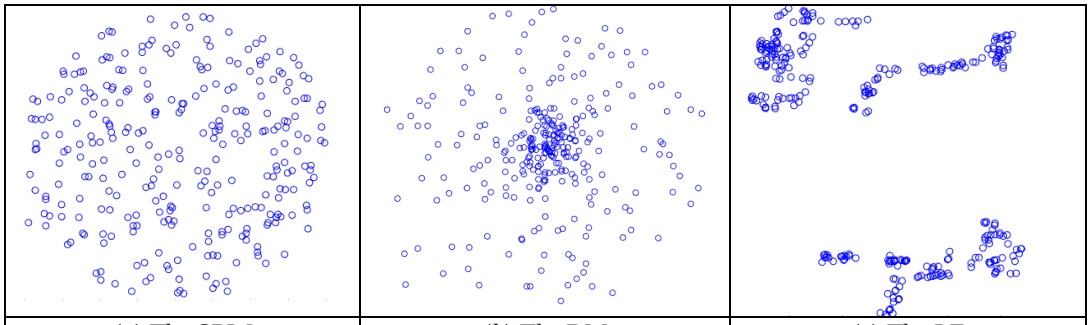
$$L(s) = \frac{\alpha\beta\Gamma(\beta)\sin(\pi\beta/2)}{\pi s^{1+\beta}}, s \to \infty$$

which results in a heavy-tailed power-law distribution

$$L(s) \square |s|^{-1-\beta}$$

When $\beta = 2$, the LF turns to be the Brownian motion (BM) whose iid random steps are drawn from the Gaussian distribution owning Fourier transform $F(k) = e^{-\alpha|k|^2}$.

The difference in the distributions of the random steps brings difference in the distributions of the their footprints of the three random walks. Seen in Figure 1, the footprints of the SRW are nearly-uniformly distributed, those of the BM are more concentrated near the center, and those of the LF are alternated between frequent short-distance jumps and occasional long-distance jumps. Therefore, the LF is suitable for global-local blending search, whereas, the SRW and the BM are suitable for local searches. The BM is particularly good for the case that there is some reference (clue) to locate searched target while SRW is more suitable for the local blind search. The Mantegna algorithm [53], whose implementation is found in [53] or [54], is always utilized to compute the LF due to its simplicity and efficiency.



(*a*) The SRW | (*b*) The BM | (*c*) The LF

Fig. 1. Footprints of the SRW, the BM, and the LF

4. Research Objective and Main Results

Given an odd composite integer $N = pq$ with $1 < p < q$ being odd integers, the purpose of this paper is to establish a way to find out p or q quickly. Since p or q can be found out by calculating the greatest common divisor (gcd) of N with an arbitrary host of N's divisors, the key is to find out one of the hosts as fast as possible. This section designs an efficient search algorithm to find a host in the sequence T_N^g by means of an SRW-blended LF. First a sequence Δ_N^g is drawn from T_N^g to reduce the searched candidates; then an Ω-zone, which is with a relatively bigger probability to pick randomly a host, is constructed to search Δ_N^g with better efficiency; in the end, the algorithm and its key issues are presented and investigated.

4.1. Δ_N^g and Its Terms

T_N^g contains $\frac{1}{2}(N-p_b-1)(N-p_b)g$ terms. The terms of its rightmost g columns of each row take the same value p_b+1, totally $(N-p_b-1)g$ ones. Without loss of generality, assume $d = \gcd(N, p_b+1) = 1$ because, otherwise, d is a divisor of N and the searching work is done. Cutting off $g-1$ ones of the repeated terms p_b+1 on each row forms Δ_N^g, described in rows and columns as follows :

row 1: $d_{1,1} = p_b + 1$;

row 2: $\underbrace{d_{2,1} = d_{2,2} = \ldots = d_{2,g} = p_b + 2}_{g \text{ items}}, d_{2,g+1} = p_b + 1$;

row 3: $\underbrace{d_{3,1} = d_{3,2} = \ldots = d_{3,g} = p_b + 3}_{g \text{ items}}, \underbrace{d_{3,g+1} = \ldots = d_{3,2g} = p_b + 3 - 1}_{g \text{ items}}, d_{3,2g+1} = p_b + 3 - 2$;

row 4: $\underbrace{d_{4,1} = \ldots = d_{4,g} = p_b + 4}_{g \text{ items}}, \underbrace{d_{4,g+1} = \ldots = d_{4,2g} = p_b + 4 - 1}_{g \text{ items}}, \underbrace{d_{4,2g+1} = \ldots = d_{4,3g} = p_b + 4 - 2}_{g \text{ items}}, d_{4,3g+1} = p_b + 4 - 3$;

.....;

row k:

$$\underbrace{d_{k,1} = \ldots = d_{k,g} = p_b + k}_{g \text{ items}}, \ldots, \underbrace{d_{k,(j-1)g+1} = \ldots = d_{k,jg} = p_b + k - (j-1)}_{g \text{ items}}, \ldots, \underbrace{d_{k,(k-2)g} = \ldots = d_{k,(k-1)g} = p_b + 2}_{g \text{ items}},$$

$$d_{k,(k-1)g+1} = p_b + k - (k-1) = p_b + 1;$$

where $1 \le k \le N - p_b - 1$ and $1 \le j \le k-1$.

The j^{th} row contains $(j-1)g+1$ terms and the total number of terms in Δ_N^g is given by

$$\chi = \Lambda + \frac{1}{2}\Lambda(\Lambda - 1)g$$

where $\Lambda = N - p_b - 1$ is the number of rows in Δ_N^g.

Using X and Y to express respectively the indices of the row and the column, it follows

$$1 \le X \le N - p_b - 1, 1 \le Y \le (X-1)g + 1$$

and the three borders of Δ_N^g are lattices corresponding to

$$\text{line } Y=1, \text{ line } X=N-p_b-1, \text{ and line } Y=(X-1)g+1.$$

The term $\omega_{X,Y}$ at row X and column Y is

$$\omega_{X,Y}=p_b+X-\left\lfloor\frac{Y-1}{g}\right\rfloor \tag{1}$$

By the way, $\gcd(N,p_b+1)=1$ means Δ_N^g has the same number of the terms hosting N's divisors as T_N^g has, saying Δ_N^g has a bigger density of the terms hosting N's divisors than T_N^g because total number of the terms of Δ_N^g is less than that of T_N^g.

4.2. The Ω-Zone

Taking $p_b=\lfloor\sqrt{N}\rfloor$ and $p_u=N-1-\lfloor\sqrt{N}\rfloor$, the Ω-zone can be obtained from Δ_N^g as the following form.

$$\Omega=\left\{\begin{array}{ccccccccc}
p_b+1, & & & & & & & & \\
\underbrace{p_b+2,\ldots,p_b+2}_{g\ ones}, & p_b+1, & & & & & & & \\
\vdots & \underbrace{p_b+2,\ldots,p_b+2}_{g\ ones}, & p_b+1, & & & & & & \\
\vdots & \vdots & \underbrace{p_b+2,\ldots,p_b+2}_{g\ ones}, & p_b+1, & & & & & \\
\underbrace{p_u-2,\ldots,p_u-2}_{g\ ones}, & \vdots & \vdots & \underbrace{p_b+2,\ldots,p_b+2}_{g\ ones}, & \ddots & & & & \\
\underbrace{p_u-1,\ldots,p_u-1}_{g\ ones}, & \underbrace{p_u-2,\ldots,p_u-2}_{g\ ones}, & \vdots & \vdots & \ddots & p_b+1, & & & \\
\underbrace{p_u,\ldots,p_u}_{g\ ones}, & \underbrace{p_u-1,\ldots,p_u-1}_{g\ ones}, & \ddots & \vdots & \vdots & \underbrace{p_b+2,\ldots,p_b+2}_{g\ ones}, & p_b+1, & & \\
 & \underbrace{p_u,\ldots,p_u}_{g\ ones}, & \ddots & \underbrace{p_u-2,\ldots,p_u-2}_{g\ ones}, & \vdots & \vdots & \underbrace{p_b+2,\ldots,p_b+2}_{g\ ones}, & p_b+1, & \\
 & & \ddots & \underbrace{p_u-1,\ldots,p_u-1}_{g\ ones}, & \underbrace{p_u-2,\ldots,p_u-2}_{g\ ones}, & \vdots & \vdots & \underbrace{p_b+2,\ldots,p_b+2}_{g\ ones}, & p_b+1, \\
 & & & \underbrace{p_u,\ldots,p_u}_{g\ ones}, & \underbrace{p_u-1,\ldots,p_u-1}_{g\ ones}, & \underbrace{p_u-2,\ldots,p_u-2}_{g\ ones}, & \cdots & \cdots & \underbrace{p_b+2,\ldots,p_b+2}_{g\ ones}, \quad p_b+1
\end{array}\right\}$$

whose four borders are given by

$$\text{line } Y=1, \text{ line } X=N-p_b-1, \text{ line } X=\left\lfloor\frac{Y-1}{g}\right\rfloor+p_u-p_b, \text{ and line } Y=(X-1)g+1.$$

The terms in the zone are easy to calculate, still by (1). For example, taking N=15 and g=2 yields the outputs as Figure 2 shows.

```
4,
5, 5, 4,
6, 6, 5, 5, 4,
7, 7, 6, 6, 5, 5, 4,
8, 8, 7, 7, 6, 6, 5, 5, 4,
9, 9, 8, 8, 7, 7, 6, 6, 5, 5, 4,
10, 10, 9, 9, 8, 8, 7, 7, 6, 6, 5, 5, 4,
      10, 10, 9, 9, 8, 8, 7, 7, 6, 6, 5, 5, 4,
            10, 10, 9, 9, 8, 8, 7, 7, 6, 6, 5, 5, 4,
                  10, 10, 9, 9, 8, 8, 7, 7, 6, 6, 5, 5, 4,
                        10, 10, 9, 9, 8, 8, 7, 7, 6, 6, 5, 5, 4,
```

Fig. 2. The Ω-zone calculated by N=15 and g=2

Compared to the *pq*-band Ξ, Ω makes each term of Ξ accumulated by g times in the Y-direction except for the boundary term p_b+1. Therefore, the Ω-zone is said to be a non-uniform one, g is also called the multiplicity of it, and the two sets, Ξ and Ω, can be described by

$$\Omega=\Xi^g$$

4.3. Distribution Traits of Integer's Divisors in the Ω-zone

In the Ω-zone, there are *ridges* that accumulate the hosts of N's divisors. As previously mentioned, every host accumulates together by g times when it appears. Referring to the distribution traits of the hosts in the interval, as described in Lemmas 3 and 4, the distribution of the hosts in the Ω-zone also demonstrates the global-sparse-with-local-accumulation. The difference between the distribution in the interval and that in the Ω-zone is that the latter enhances the accumulation by densifying the hosts.

4.4. Framework of the LF-based Algorithm

The characteristic of the global-sparse-with-local-accumulation is beneficial for designing the algorithm of global-local blending searches, as proposed in [48]. This naturally arises the idea to utilize the LF to be the framework to design such an algorithm because its movements alternate between frequent short-distance jumps and occasional long-distance jumps. This section designs an algorithm of the kind. By preserving the long-distance jumps that jump globally from one place to another in the entire searched zone, the algorithm utilizes certain local search (LS) method to search the objective (a host) at each local zone where the short-distance jumps happen. So it is called an LS-blended LF. The LS-blended LF, as intuitively illustrated in Figure 3, does not concern the 'frequent short-distance jumps' of the classical LF but cares about the local zone that the 'frequent short-distance jumps' cover because the taking-off and landing points of those short-distance jumps might be very sparse in a local perspective and miss the objective even if they are very close to one another. In addition, the long jump is required not to be parallel to the ridges owning to the structure of the Ω zone.

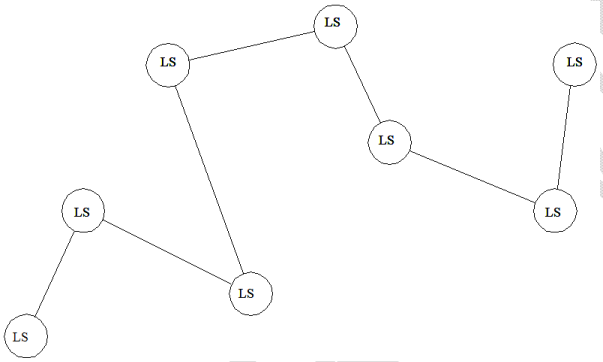


Fig. 3. The LS-blended LF

4.4.1. Algorithm of SRW-blended LF

As previously mentioned, the SRW is suitable for local blind searches. Therefore this subsection presents an SRW-blended LF to be an implementation of the LS-blended LF. It includes an LF main routine and an SRW-search subroutine.

The LF main routine contains a stage of initialization and a stage of dynamical computation. In the stage of initialization, it calculates a value of the multiplicity g, sets necessary parameters for the initial local zone, estimates the maximal permissive numbers of the long jumps (or steps) of the LF (The routine is forced to quit if the number of the long jumps exceeds the maximal permissive number), randomly chooses a start point in the Ω-zone, computes the restart points, and call the SRW-search subroutine to search the local zone. If the search fails to find the objective, it then begins the dynamical stage. The dynamical stage consists of jumping long distances in the case the SRW-search fails to find the objective, calculating dynamical local zones around the landing points of the long jumps, calling the SRW-search subroutine to search the dynamical local zones, and starting the necessary restart rounds. The dynamical stage is kept looping until the objective (a host) is found or the number of the jumps reaches the maximal permissive number. With Table 1, the LF main routine is described clearly and intuitively.

Table 1. Description of the LF main routine

Initial stage	Calculate a g.
	Calculate parameters of the initial local zone.
	Estimate the maximal permissive number of steps.
	Chooses randomly a start point.
	Calculate the restart points.
	Search the initial zone with the SRW-search.

Dynamical stage	Loop	Jump a long distance if the search fails to find a host.
		Calculate a dynamical local zone at the landing point.
		Search the dynamical local zone with the SRW-search.
		Return the found objective if the search successes.
		Restart a next round if necessary.

The SRW-search subroutine searches the objective locally with the SRW in an area; it has four input arguments passed from the main routine: *N*, g, the start-point at which the SRW begins, and the parameter to calculate a local zone within which the SRW-search searches. It first calculate the maximal permissive number of the steps, then calculate the local zone, and then carries out the search by the SRW until the objective is found or the number of the steps reaches maximal permissive number of the steps. It returns the found objective and the number of the moved steps during the search. If the search fails, it returns an information of failure. Table 2 describes SRW-search subroutine clearly and intuitively.

Table 2. Description of the SRW-search subroutine

Calculate the maximal permissive number of the steps.
Calculate the host search zone.
Search the local zone until the objective is found or the number of the steps is overflow.
Return the found objective or information of failure.

4.4.2. Key Issues of the Algorithm

Seen from Tables 1 and 2, the multiplicity g, the start-point, restart-point, the local search zone, and the maximal permissive number of steps are key quantities to determine the chance of success and efficiency of the search. Here present strategies to calculate them.

1. ***The start point***. In order to find the objective in the least number of steps , the start point, say S_0=(X_0, Y_0), is better set near a host. Because the hosts are unknown before one of them is found, an initial start point can be chosen by a rough estimation. Considering $p \le \sqrt{N} \le q \Rightarrow ip \le i\sqrt{N} \le iq$ for an arbitrary positive integer *i* and $\lfloor \sqrt{N} \rfloor$ is the biggest *i* satisfying $i\sqrt{N} \le N$, X_0 can be chosen by

$$X_0 = \alpha \lfloor \sqrt{N} \rfloor \tag{2}$$

where $1 < \alpha \le \lfloor \sqrt{N} \rfloor$ is an integer.

Because S_0 must be inside the Ω-zone, the center *C* of Δ_N is naturally a good reference, as illustrated in Figure 4, in which *M* is end point of the median *OM*.

Referring to Lemma 4, the start point is better not to be far away from ridge $\lfloor \sqrt{N} \rfloor$. That is, S_0 is not far away from *M*. Since *C* is $\frac{1}{3}(N-1)+\frac{2}{3}\lfloor \sqrt{N} \rfloor$, letting

$$\alpha_c \lfloor \sqrt{N} \rfloor = \frac{1}{3}(N-1)+\frac{2}{3}\lfloor \sqrt{N} \rfloor$$

yields

$$\alpha_c = \left\lfloor \frac{2\sqrt{N}+N-1}{3\sqrt{N}} \right\rfloor \approx \left\lfloor \frac{2}{3}+\frac{\sqrt{N}}{3} \right\rfloor$$

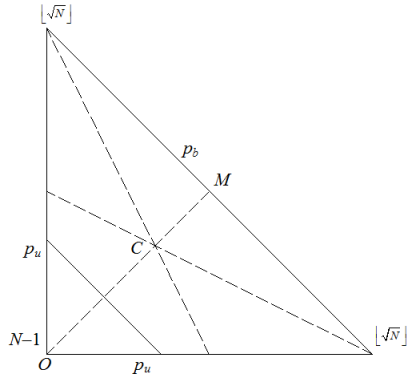


Fig. 4. The initial start point S_0 in Δ_N

Hence X_0 is randomly chosen by

$$\left\lfloor \sqrt{N} \right\rfloor < X_0 \le \alpha_c \left\lfloor \sqrt{N} \right\rfloor \tag{3}$$

and Y_0 is chosen in the Ω-zone by

$$Y_0 = gX_0 \tag{4}$$

2. ***The Local Zone***. Although the local zone can be any shape, it is suggested to be a square region described with its inscribed circle, as shown in Fig. 5, for ease of computation. Hence it is positioned by the center of the circle and measured by the radius *r* of the circle. The zone with center S_0 is simply called *zone* S_0 for convenience. By default, the radius of a zone is an integer.

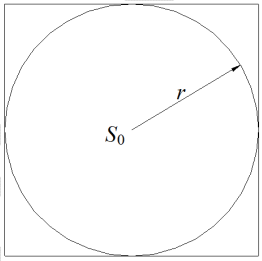


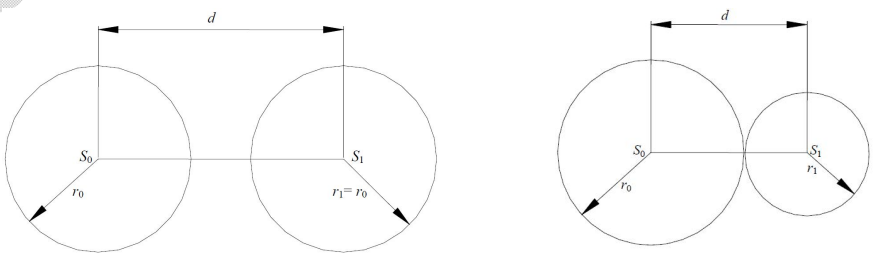
Fig. 5. Local zone and its inscribed circle

The initial local zone is set with a fixed radius *r* relevant to the multiplicity g. In order to cover as many hosts as possible, *r* is suggested to satisfy.

$$r > g \tag{5}$$

Since the SRW search takes a longer time when *r* gets bigger, $r = \left\lfloor \frac{3}{2} g \right\rfloor$ or $r = 2g$ can be practicable.

The calculation of the dynamical local zone depends on the distance between the taking-off and landing points of the LF. Assume zone S_0 with radius r_0 is the local zone of the taking-off point S_0, S_1 with radius r_1 is that of the landing-point following S_0, and *d* is the integer distance between S_0 and S_1, as depicted in Fig. 6; then set $r_1 = r_0$ in the case $d \ge 2r_0$ and calculate $r_1 = d - r_0$ in the case $d < 2r_0$.



(*a*) $d \ge 2r_0$ (*b*) $d < 2r_0$

Fig. 6. The SRW-search area

Remark 1. The SRW-blended LF or the LS- blended LF obviously becomes the classical LF if the radius of the local zone is 0.

3. ***The restart point***. The LF should be carried out inside the Ω-zone. Once it goes outside, a restart point is necessary to restart a next round. The restart point is better near but different from initial start point S_0. Obviously, points surrounding the initial zone can meet the needs of this situation. Assume the initial zone is of side $2r$ such that $r \geq 9$ and $r = uv$ is a composite integer with u being odd. Then each side of the initial zone contains $2v$ line segments of length u. Constructing another square with center S_0 and side $2(r+u)$, as illustrated with Fig. 7, obtains $4(v+1)$ restart-points from the center points of the small square regions whose sides are length u. The small square region, each of which contains u^2 integer points, are surely the local zones of the restart points.

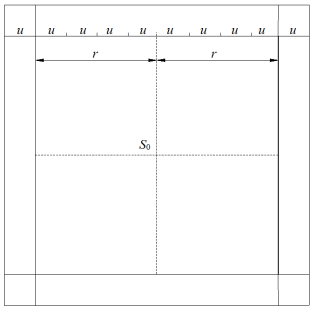


Fig. 7. Square initial zone to generate restart-points

If $r < 9$, construct 12 squares around the initial zone, each of which is of side r, as depicted in Fig.8. The restart points are taken from the centers of these squares.

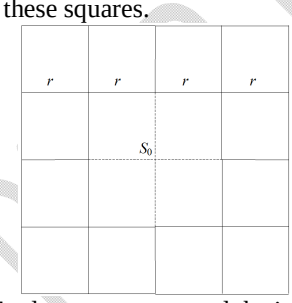


Fig.8. Twelve squares around the initial zone

Since there is a small number of restart points, they can therefore be stored in a heap, say,

$$H_{rst} = Rst(S_0, r) \tag{6}$$

When a restart round begins, a restart point is popped up from the heap

$$P = Popup(H_{rst}) \tag{7}$$

The whole search stops if H_{rst} is turned to be empty.

Remark 2. For the classical LF, the restart points can be randomly chosen by (2) and (4).

4. ***The multiplicity*** **g**. Theoretically, the bigger g is, the more hosts are accumulated somewhere. Nevertheless the relation (5) shows that a large g also increases the search time of the SRW. It is surely critical to make the two balance. Let $r = \alpha g$ be the radius of the local zone A and n_s be the number of the steps that the SRW moves in A, where $\alpha > 1$ is an integer; then n_s is a random variable satisfying

$$0 \leq n_s \leq 4r^2 = 4\alpha^2 g^2 \tag{8}$$

and the expectation $E(n_s)$ is calculated by

$$E(n_s) = \frac{1}{2r}\int_0^{2r} 4(\frac{x}{2})^2 dx = \frac{4}{3}r^2 = \frac{4}{3}\alpha^2 g^2$$

Now is to determine a proper value for g. Consider there is a host in *A*. Referring to the efficiency of the deterministic algorithms[55][56], assume the best case that the SRW finds out the host is $c_1 \cdot \log_2 N$ and the worst case is $c_2 \cdot \sqrt[\lambda]{N}$, where $c_1 > 0$, $c_2 > 0$ are real numbers, and $\eta > 1$ is an integer. That is

$$c_1 \cdot \log_2 N \le \frac{4}{3}\alpha^2 g^2 \le c_2 \cdot \sqrt[\lambda]{N}$$

or

$$\frac{1}{2\alpha}\sqrt{3c_1 \log_2 N} \le g \le \frac{1}{2\alpha}\sqrt{3c_2} N^{\frac{1}{2\lambda}}$$

Taking $\alpha = 1.5$ yields

$$0.5773\sqrt{c_1 \cdot \log_2 N} \le g \le 0.5773 N^{\frac{1}{2\lambda}}\sqrt{c_2}$$

while taking $\alpha = 2$ yields

$$0.433\sqrt{c_1 \cdot \log_2 N} \le g \le 0.433 N^{\frac{1}{2\lambda}}\sqrt{c_2}$$

Either case yields approximately

$$\frac{\sqrt{c_1 \cdot \log_2 N}}{2} \le g \le \frac{\sqrt[2\lambda]{N}\sqrt{c_2}}{2}$$

indicating

$$\left\lceil \frac{1}{2}\sqrt{c_1 \cdot \log_2 N} \right\rceil \le g \le \left\lfloor \frac{1}{2} N^{\frac{1}{2\lambda}}\sqrt{c_2} \right\rfloor$$

Taking $c_1 = c_2 = 1$ yields*

$$\left\lceil \frac{1}{2}\sqrt{\log_2 N} \right\rceil \le g \le \left\lfloor \frac{1}{2} N^{\frac{1}{2\lambda}} \right\rfloor$$

Since $\lambda = \frac{\log_2 N}{\log_2 \log_2 N}$ leads to $\log_2 N > N^{\frac{1}{\lambda+1}}$ and $\log_2 N = N^{\frac{1}{\lambda}} \Leftrightarrow \sqrt{\log_2 N} = N^{\frac{1}{2\lambda}}$, it is known that

$$\eta = \left\lfloor \frac{\log_2 N}{\log_2 \log_2 N} \right\rfloor \tag{9}$$

is the maximal integer such that $\log_2 N < N^{\frac{1}{\eta}}$. Accordingly, g can be chosen randomly by

$$\left\lceil \frac{1}{2}\sqrt{\log_2 N} \right\rceil \le g \le \left\lfloor \frac{1}{2} N^{\frac{1}{2\eta}} \right\rfloor \tag{10}$$

5. ***The maximal permissive number of steps***. Once the multiplicity g is chosen, the maximal permissive number of the steps for the SRW to move in the initial local zone is $4\alpha^2 g^2$, as seen in (8), whereas, that to move in the dynamical local zone is calculated with the dynamical radius of the dynamical area, as stated previously. If the total search steps are within $\sqrt[\eta]{N}$, the maximal permissive number of the long jumps for the LF to flight in the Ω-zone is estimated by

$$n_{\max}^{LF} = \left\lfloor \frac{\sqrt[\eta]{N}}{4\alpha^2 g^2 (n_{rst} + 1)} \right\rfloor$$

where n_{rst} is the number of the restart points and η is defined in (7).

Taking $\alpha = 2$ yields

$$n_{\max}^{LF} \le \left\lfloor \frac{\sqrt[\eta]{N}}{16 g^2 (n_{rst} + 1)} \right\rfloor \tag{11}$$

* Just for determining a value for g.

6. ***The Steps of the Jumps***. For a two-dimensional LF, each move is formed from a step in *x*-direction and a step in *y*-direction. The step length is surely a critical quantity to determines the search efficiency and effectiveness. Owning to the structure of the Ω-zone, the step in *y*-direction is properly bigger than that in *x*-direction. For the SRW, *y*-step is proposed not to be bigger than $1.5g$.

4.4.3. The Algorithm in Detail

After all the key issues settled down, the LF main routine and SRW-search subroutine are designed and presented below in detail; their N-S flow charts are depicted in figures 9 and 10 respectively. In the routines, *N* is the odd composite integer to be factorized, g is the multiplicity, Obj_{found} is a state parameter to record if the object is found or not, $obj(P,g)$ is a function to calculate (1) with point *P* and g, $Y = LF(X)$ expresses the LF from point *X* to point *Y*, $Y = SRW(X)$ is the SRW from point *X* to point *Y* , $d = |Y - X|$ is the integer distance from *X* to *Y*, $H = Rst(X,Y)$ calculates the heap of the restarting points with *X* and *Y* , and $Y = Popup(X)$ to indicate *Y* is obtained by popping up the heap *X*.

Subroutine : **SRW-search**

Inputs: *N*, g, the start point P_0 , radius *r* of the local zone.

Step 1. Calculate the maximal permissive number $n_{SRW} = 4r^2$.

Step 2. Initialize the step-counter c_{SRW} and state-parameter Obj_{found} :

$c_{SRW} = 0$; $Obj_{found} = 1$.

Step 3. Set the moving point P : $P = P_0$.

Step 4. Loop while ($Obj_{found} = 1$ and $c_{SRW} \le n_{SRW}$)

(i) $P_1 = SRW(P)$;

(ii) $Obj_{found} = (obj(P_1, g), N)$;

(iii) If $Obj_{found} > 1$ then return Obj_{found} and c_{SRW} ;

else

$P = P_1$;

$c_{SRW} = c_{SRW} + 1$;

End if.

End loop.

Step 5. return Obj_{found} and c_{SRW} .

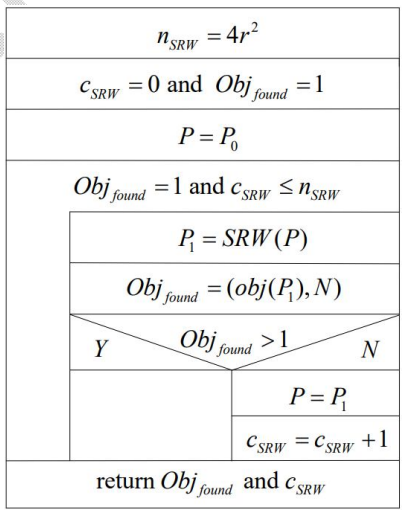


Fig.9. N-S flow chart of SRW subroutine

The LF main routine

Inputs: N.

Step 1. Initialize the counters c_{LF}^{s}, c_{LF}^{l}, and c_h:

$c_{LF}^{s}=0$, $c_{LF}^{l}=0$, and $c_h=0$.

Step 2. Initialize the state-parameter Obj_{found}: $Obj_{found}=1$.

Step 3. Initialize the number of the restart points: $n_{rst}=12$.

Step 4. Calculate η by (9).

Step 5. Choose randomly an odd composite $g=uv$ by (10).

Step 6. Calculate $n_{\max}^{LF}$ by (11).

Step 7. Calculate point $P_0 \in \Omega$ by (3) and (4).

Step 8. Set radius of the initial local zone $r_0=2g$.

Step 9. Set initial radius of the dynamical zone $r=r_0$.

Step 10. Calculate the heap of restart points: $H_{rst}=Rst(P_0,r_0)$ by (6).

Step 11. Record n_{rst} and the radius r_{rst} of the restart zone.

Step 12. $[Obj_{found},c_{SRW}]=\text{SRW-search}(N,g,P_0,r_0)$.

Step 13. $c_{LF}^{s}=c_{LF}^{s}+c_{SRW}$.

Step 14. If $Obj_{found}>1$ then return Obj_{found}, c_{LF}^{s} and c_{LF}^{l}; End if.

Step 15. Loop while($Obj_{found}=1$ and $c_h<n_{rst}$ and $c_{LF}\le n_{\max}^{LF}$)

(i) $P_1=P_0$.

(*ii*) $P_2=LF(P_1)$.

(*iii*) $c_{LF}^{s}=c_{LF}^{s}+1$.

(*iv*) $d=|P_2-P_0|$.

(*v*) Loop while ($d\le r$)

$P_1=P_2$.

$P_2=LF(P_1)$.

$c_{LF}^{s}=c_{LF}^{s}+1$.

$d=|P_2-P_0|$.

End loop // jump out of the zone P_0

(*vi*) If $P_2\in\Omega$ then

$P_0=P_2$.

If $d<2r$ then $r=d-r$; End if.

else

$P_0=Popup(H_{rst})$.

$r=r_{rst}$.

$c_h=c_h+1$.

End if.

(*vii*) $[Obj_{found},c_{SRW}]=\text{SRW-search}(N,g,P_0,r)$;

(*viii*) $c_{LF}^{s}=c_{LF}^{s}+c_{SRW}$

(*ix*) if $Obj_{found}>1$ then return Obj_{found}, c_{LF}^{s} and c_{LF}^{l}; End if.

(*x*) $c_{LF}^{l}=c_{LF}^{l}+1$;

End loop.

Step 16. return Obj_{found}, c_{LF}^{s} and c_{LF}^{l}.

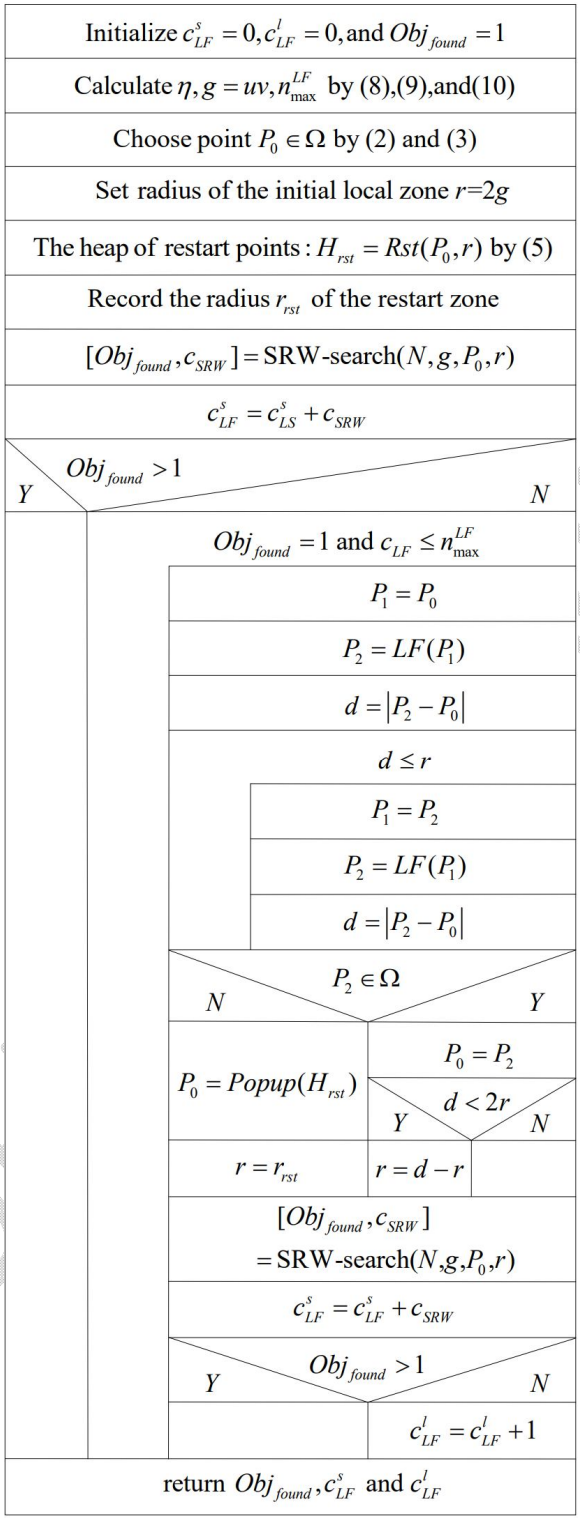


Fig.10. N-S flow chart of the LF main routine

Remark 3. The subroutine SRW-search actually performs the local search, which is depicted in Figure 3. It hence can be a routine of a randomized search or a brutal search.

Remark 4. In the case of a small *N*, g could be a small number between 1 and 8, which would result in no solution in Step 4. This time the restart-points calculated in Step 8 should be treated in the case mentioned in Fig.8.

4.4.4. Time Complexity and Space Complexity

The time complexity is g dependent. Consider a successful search. The subroutine SRW-search takes merely one step to find out the objective in the best case while it moves $n_{SRW} = 16g^2 \le 8N^{\frac{1}{\eta}} < 8N^{\frac{\log_2 \log_2 N}{\log_2 N - 1}}$ steps in the worst case because $g \le \left\lfloor \frac{1}{2} N^{\frac{1}{2\eta}} \right\rfloor \le \frac{1}{2} N^{\frac{1}{2\eta}}$ and $\frac{\log_2 N}{\log_2 \log_2 N} - 1 < \eta \le \frac{\log_2 N}{\log_2 \log_2 N}$. Since it takes $O(\log_2 Nm)$ times to compute the greatest common divisor of N with another odd integer m smaller than N, the total time complexity of the subroutine SRW-search is $O(\log_2 Nm) < O(2\log_2 N)$ in the best case and $O(16N^{\frac{\log_2 \log_2 N}{\log_2 N - 1}} \log_2 N)$ in the worst case.

The LF main routine takes merely one jump in the best case and $n_{\max}^{LF} \le \left\lfloor \frac{\sqrt[\eta]{N}}{16g^2(n_{rst}+1)} \right\rfloor \le \frac{\sqrt[\eta]{N}}{16g^2(n_{rst}+1)}$ $\le \frac{N^{\frac{\log_2 \log_2 N}{\log_2 N}}}{16(\log_2 N)^2 (n_{rst}+1)}$ in the worst case. Because $n_{rst} \ge 12$, the total search process takes $O(\log_2 N)$ in the best case and $O(\frac{N^{\frac{\log_2 \log_2 N}{\log_2 N} + \frac{\log_2 \log_2 N}{\log_2 N - 1}}}{13\log_2 N}) \approx O(\frac{N^{\frac{2\log_2 \log_2 N}{\log_2 N}}}{13\log_2 N})$ in the worst case. Using T_{best} and T_{worst} to denote the time complexity of the total search process, it follows

$$T_{best} = O(2(\log_2 N)^2)$$

and

$$T_{worst} = O(\frac{N^{\frac{2\log_2 \log_2 N}{\log_2 N}}}{13\log_2 N})$$

The whole process have a heap to store the restart points, hence it takes $O(n_{rst}) \approx O(1)$ space complexity.

4.5. Numerical Experiments

Numerical experiments are performed with Matlab on a Lenovo P70 (Laptop) with Intel(R) Xeon(R) E3-1505M v5@2.80GHz, 64GB memory, and Windows 8 OS.

4.5.1. Description and Data

Four local searches, an SRW in a local circle zone (SRWc), a brutal search in a circle zone (BSc), a brutal search in a square (BSs), and an SRW in a square zone (SRWs) are engaged respectively in the main routine. Due to that Matlab's *gcd* function can merely compute integers not exceeding 2^{53}, the experiments are performed on semi-primes whose decimal digits are no more than 16. To have a comparison, Tables 3, 4, and 5 list the experimental results of factoring semi-primes taken partly from Table B of paper [33]. Table 3 lists the results obtained from LF+SRWc, Table 4 lists the results obtained from LF+BSc, Table 5 lists the results from LF+BSs, and Table 6 is for LF+ SRWs. In the tables, the column 'Time' is counted with Matlab function *toc*. Readers can check the source codes of Matlab programs as well as the original records of the experiments in [58].

Table 3. Factorization with LF+SRWc

Integer N	Digits	Found Divisor	Searching Steps	Time
10909343	8	4051	994	0.2817
29835457	8	7457	374	0.0900
392913607	9	21937	1399	0.4497
5325280633	10	57731	3922	0.7980
42336478013	11	174169	2988	1.1181
272903119607	12	374989	30974	9.9060
11683458677563	14	2595899	1415616	202.3835
51790308404911	14	5581897	805214	305.5239
115137038087959	15	10037141	2894944	490.4322

8335465900089539	16	90745723	12612030	2424.0374

Table 4. Factorization with LF+BSc

Integer *N*	Digits	Found Divisor	Searching Steps	Time
10909343	8	2693	928	0.0924
29835457	8	4001	270	0.0479
392913607	9	17911	280	0.0370
5325280633	10	57731	7967	1.4962
42336478013	11	174169	33298	6.7620
272903119607	12	374989	21524	4.6436
11683458677563	14	4500737	1122654	238.0093
51790308404911	14	5581897	474291	109.6567
115137038087959	15	10037141	5372793	894.4002
8335465900089539	16	91855193	2307434	581.0664

Table 5. Factorization with LF+BSs

Integer *N*	Digits	Found Divisor	Searching Steps	Time
10909343	8	2693	667	0.0605
29835457	8	4001	231	0.0230
392913607	9	17911	917	0.0938
5325280633	10	57731	10678	0.9446
42336478013	11	174169	138363	15.2974
272903119607	12	727763	262057	36.1139
11683458677563	14	2595899	6733357	832.3340
51790308404911	14	5581897	10547824	1167.6061
115137038087959	15	10037141	5377783	739.3921
8335465900089539	16	90745723	5642844	797.1368

Table 6. Factorization with LF+SRWs

Integer *N*	Digits	Found Divisor	Searching Steps	Time
10909343	8	2693	328	0.0948
29835457	8	7457	247	0.0647
392913607	9	17911	283	0.0736
5325280633	10	92243	2033	0.7131
42336478013	11	243077	1537	0.4975
272903119607	12	374989	4279	1.3427
11683458677563	14	4500737	598758	238.0979
51790308404911	14	5581897	823085	369.1591
115137038087959	15	10037141	1021821	192.0498
8335465900089539	16		out of range	

4.5.2. Comments and Existed Problems

The experiments test many factorizations each of which has been tested more than 3 times. The results are very well satisfactory because every test is successful. Being a pure randomized computation, the time each case consumes varies a lot. The records in the tables here are taken from those that spend the least time. Readers are surely aware that the computing time here is incomparable with those reported in other literatures due to the variation of computing configurations, not to speak of a truly randomized computation.

Nevertheless, the experiments also found that there were things to be improved and investigated. Firstly, the start point and the step-length affect the computing time. With the guidance of section 4.4.2, the start point was chosen randomly and the step-length of the LF was fixed. Experiments found that variation of the step-length could reduce the searching time rapidly. How to change the step-length remains to investigate further. Secondly, the designed programs are very rough, as readers can see in the source codes; they can simply describe the principle of how the LF-based algorithm works. They are expected optimized by veteran programmers. In the end, although it is easy to program with Matlab to test the designed algorithms for its rich resources in dealing with the random numbers and stochastic processes, such as the uniform distribution, the normal distribution, and so on, its limitation to compute *gcd* within integers of no more than 16 decimal digits becomes a fetal bottle-neck to apply the research for factoring big integers. Better platform with resources as many as Matlab's is urgently in need for computing big numbers.

Remark 5. Tools like GMP , NTL, and MIRACL are capable of calculating big integers but lack of enough

function to compute the random number with respect to specific distribution. For example, the GMP can calculate a random number from *uniform distribution* but cannot calculate the one from *normal distribution*. Accordingly, nowadays they cannot be applied for the computation related with the LF. I have tested the Maple software, at which I am quite skill, but it failed in calculate the *gcd*, either. The situation is just like that mentioned in [57] four year ago.

5. Conclusion and Future Work

Applying the intelligent algorithms for factoring integers has been more and more paid attention this years. For an odd semi-prime $N=pq$, previous studies show that integers having a divisor p or q distribute sparsely in the whole interval [1, N -1] and accumulatively in some local sub-interval. This trait is suitable to apply the LF to search an integer hosting a divisor of N because the LF moves alternately between frequent short-distance jumps and occasional long-distance jumps. The LF-based algorithm raised in this paper is an attempt of such practice. Experiments show that the algorithm is successful although it needs improvement and further optimization. As commented in section 4.5.2, future work is to focus on investigating the choice of the start point, the step-length, and the related issues. Hope more younger people join the work.

Appendix

Source code as well as records of the numerical experiments can be downloaded at:
https://drive.mathworks.com/sharing/6e4cf414-f286-4655-a2bc-cc1af222f0f8

References

[1] Hans Riesel. (1994). Prime Numbers and Computer Methods for Factorization(2nd Edition), Birkhauser.

[2] Crandall R, Pomerance C. (2005). Prime Numbers: A Computational Perspective(2nd Edition). New York: Springer-Verlag.

[3] Shi Bai. (2006). Computer Methods for Integer Factorization and Discrete Logarithm: A Cryptographic Perspective. Mater Thesis, Australian National University.

[4] Yan S Y. (2013). Computational number theory and modern cryptography. Wiley & Higher Education Press.

[5] Alfred J Menezes, Paul C van Oorschot, Scott A Vanstone. (2018). Handbook of Applied Cryptography. CRC press. DOI:10.1201/9781439821916

[6] Joppe Bos, Martijn Stam . (2021). Computational Cryptography: Algorithmic Aspects of Cryptology. Cambridge University Press.

[7] Wunderlich M C. (1988). Computational methods for factoring large integers. *Abacus*, 5(2), 19-33.

[8] Peter L Montgomery. (1994). A Survey of Modern Integer Factorization Algorithms. *CWI*, 7(4), 337-365

[9] Pomerance C. (1996). A Tale of Two Sieves . *Notice of The AMS*,1996, 43(12), 1473–1485.

[10] Arjen K Lenstra. (2000). Integer Factoring . *Designs, Codes and Cryptography*, 19, 101-128.

[11] Kefa R. (2006). Review of Methods for Integer Factorization Applied to Cryptography. *Journal of Applied Sciences*, 6(2), 458-581.

[12] Wanambisi A W, Aywa S, Maende C, Muchiri Muketha G. (2013). Advances in Composite Integer Factorization, *Methematical theory and Modeling*, 13(2), 86-90.

[13] Sonal Sarnaik, Dinesh Gadekarand Umesh Gaikwad. (2014). An overview to Integer factorization and RSA in Cryptography . *International Journal for Advance Research in Engineering and technoloy*, 2(9):21-26

[14] Duta C L, Gheorghe L, Tapus N, (2016). Framework for evaluation and comparison of integer factorization algorithms, *2016 SAI Computing Conference* (pp.1047-1053), London, UK,.
DOI: 10.1109/SAI.2016.7556107.

[15] Zhang X, Li M, Jiang Y, Sun Y. (2019). A Review of the Factorization Problem of Large Integers . In: Sun X,

Pan Z, Bertino E (eds) Artificial Intelligence and Security. ICAIS 2019. Lecture Notes in Computer Science, 11635:202-213. DOI:10.1007/978-3-030 -24268-8 _19

[16] Majid Mumtaz, Luo Ping. (2019). Forty years of attacks on the RSA cryptosystem: A brief survey. *Journal of Discrete Mathematical Sciences and Cryptography*, 22(1),9-29. DOI: 10.1080/09720529.2018. 1564201

[17] Patro Subhasree, Alvaro Piedrafita. (2020). An Overview of Quantum Algorithms: From Quantum Supremacy to Shor Factorization. 2020 IEEE International Symposium on Circuits and Systems (ISCAS) (pp. 1-5). DOI:10.1109/ISCAS45731.2020.9180793

[18] Wu Liangshun, et al. (2019). The Integer Factorization Algorithm With Pisano Period. IEEE Access 7 (pp. 167250-167259). DOI:10.1109/ACCESS.2019.2953755

[19] Wang Baonan, He Feng, Yao Haonan, Wang Chao. (2020). Prime factorization algorithm based on parameter optimization of Ising model. *Scientific Reports*, 10:7106. DOI:10.1038/s41598-020-62802-5

[20] Kritsanapong Somsuk. (2020). The new integer factorization algorithm based on fermat's factorization algorithm and euler's theorem. *International Journal of Electrical and Computer Engineering*, 10(2), 1469-1476. DOI: 10.11591/ijece.v10i2.pp1469-1476

[21] Overmars Anthony, Sitalakshmi Venkatraman. (2021). New Semi-Prime Factorization and Application in Large RSA Key Attacks. *Journal of Cybersecurity and Privacy*. 1, 660-674. DOI:10.3390/jcp1040033

[22] Murthy D H R. (2022). A Comprehensive study on RSA Prime Factorization Algorithms, *Journal of Engineering & Management*, 6(1),101-104. DOI: 10.37314/JJEM.2022.060114

[23] Omollo Richard, Arnold Okoth. (2022). Large Semi Primes Factorization with Its Implications to RSA Cryptosystems. *BOHR International Journal of Smart Computing and Information Technology* (3),1-8. DOI:10.54646/bijscit.011

[24] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, et al. (2022). The State of the Art in Integer Factoring and Breaking Public-Key Cryptography. *IEEE Security and Privacy Magazine*, 20 (2), 80-86. DOI: 10.1109/MSEC.2022.3141918.

[25] Ruslan Skuratovskii. (2022). Optimal Method of Integer Factorization. *Wseas Transactions on Information Science and Applications*,19,23-29. DOI: 10.37394/23209.2022.19.3

[26] Balasubramanian, Kannan and Mohana Priya Pitchai. (2023). A Survey of Fermat Factorization Algorithms for Factoring RSA Composite Numbers. *Multidisciplinary Science Journal. 6,* 2024ss0101 DOI:10.31893/multiscience.2024ss0101.

[27] Ahmed T Sadiq, Tayseer S Atia, Abd-alsattar S Awad. (2009). Integer Factorization Problem Solving Using Tabu Search. *Journal of Wasit for Science and Medicine*,(1),1-10. DOI:10.31185/jwsm.20

[28] Ade Candra, Mohammad Andri Budiman, Dian Rachmawati. (2017). On Factoring The RSA Modulus Using Tabu Search. *Journal of Computing and Applied Informatics*, 01(1),30-37. DOI:10.32734/ JOCAI.V1.I1-65

[29] Choudhury Bhargab, Sangita Neog. (2015). Particle Swarm Optimization Algorithm for Integer Factorization Problem (IFP). *International Journal of Computer Applications*, 117,14-17. DOI:10.5120/20613-3276

[30] Wang X. (2017). Factorization of Odd Integers as Lattice Search Procedure. International Symposium on Intelligence Computation and Applications(pp.7-22). DOI:10.1007/978-981-13-1651-7_22

[31] Dash, Avinash et al. (2018). Exact search algorithm to factorize large biprimes and a triprime on IBM quantum computer., Quantum Physics, arXiv:1805.10478v2.

[32] Rutkowski Emilia, Sheridan K Houghten. (2020). Cryptanalysis of RSA: Integer Prime Factorization Using Genetic Algorithms. 2020 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-8). DOI:10.1109/CEC48606.2020.918572

[33] Mahadee Al Mobin, Md Kamrujjaman. Cryptanalysis of RSA Cryptosystem: Prime Factorization using Genetic Algorithm. arXiv:2407.05944v1 [math.GM] 24 Jun 2024

[34] Mohit Mishra, Utkarsh Chaturvedi, and Kaushal K Shukla. (2016). Heuristic algorithm based on molecules optimizing their geometry in a crystal to solve the problem of integer factorization. *Soft Computing*,

20(9),3363–3371. DOI: 10.1007/s00500-015-1772-8

[35] Fagin Barry S. (2021). Search Heuristics and Constructive Algorithms for Maximally Idempotent Integers. *Information*, 12, 305. DOI:10.3390/info12080305

[36] Hittmeir Markus. (2023). Smooth Subsum Search: A Heuristic for Practical Integer Factorization. International Journal of Foundations of Computer Science, arXiv:2301.10529v2. DOI:10.1142/s0129054123500296

[37] Chechkin A V, Metzler R, Klafter J, Gonchar V Y. (2008). Introduction to the theory of Lévy flights. Anomalous Transport(pp. 129–162). Wiley-VCH Verlag GmbH & Co. KGaA.

[38] Raposo E P, Buldyrev S V, da Luz M G E, Viswanathan G M, Stanley H E. (2009). Lévy flights and random searches. *Journal of Physics A: Mathematical and Theoretical*,42,1-23.DOI:10.1088/1751-8113/42/43/434003

[39] Reynolds A M, Rhodes C J. (2009). The Lévy flight paradigm: random search patterns and mechanisms. *Ecology*, 90: 877-887. DOI: 10.1890/08-0153.1

[40] Kamaruzaman, Anis Farhan, et al. (2013). Levy Flight Algorithm for Optimization Problems - A Literature Review. *Applied Mechanics and Materials*, 421,496 - 501. DOI:10.4028/www.scientific.net/ AMM.421.496

[41] Truyen Tran, Trung Thanh Nguyen, Hoang Linh Nguyen. (2014). Global optimization using Lévy flights. arXiv:1407.5739v1.

[42] Mridul Chawla , Manoj Duhan. (2018). Levy Flights in Metaheuristics Optimization Algorithms–A Review, *Applied Artificial Intelligence*, 32:9-10, 802-821. DOI:10.1080/08839514.2018.1508807

[43] Li J, An Q, Lei H, Deng Q, Wang G. G. (2022). Survey of Lévy Flight-Based Metaheuristics for Optimization. *Mathematics*, 10, 2785.DOI:10.3390/math10152785

[44] Edlyn Teske. (2000). On Random Walks for Pollard's Rho Method. Mathematics of Computation ,70(234), 809–825. DOI:10.1090/S0025-5718-00-01213-8

[45] Wang X, Luo J, Tian Y. (2021). Factorization of Odd Integers as Random Walks on Valuated Binary Trees. ICCAI21: Proceedings of the 2021 7th International Conference on Computing and Artificial Intelligence(pp.244-251). DOI:10.1145/3467707.3467744

[46] Wang X. (2016). Valuated Binary Tree: A New Approach in Study of Integers. *International Journal of Scientific and Innovative Mathematical Research (IJSIMR)*,4(3),63-67. DOI: DOI:10.20431/2347-3142.0403008

[47] Wang X. (2024). Distribution of Divisors of an Integer in a Triangle Integer Sequence, *JP Journal of Algebra, Number Theory and Applications*, 63(2),185-208. DOI:10.17654/0972555524011

[48] Wang X. (2024). Minimal Gap among Integers having a Common Divisor with an Odd Semi-prime. *Journal of Advances in Mathematics and Computer Science*, 39(6), 1–7. DOI:10.9734/jamcs/2024/ v39i61896

[49] Wang, X. (2024). Maximum Gap Among Integers Having a Common Divisor With an Odd Semi-Prime. *Journal of Advances in Mathematics and Computer Science*, 39 (10):51-61. DOI:10.9734/jamcs/2024/v39i101934.

[50] Wang, X. (2023). Densification of Witnesses for Randomized Algorithm Design. Journal of Advances in Mathematics and Computer Science, 38(10):44-69. DOI: 10.9734/JAMCS/2023/v38i101823.

[51] Xin-She Yang, Xingshi He. (2019). Mathematical Foundations of Nature-Inspired Algorithms. Springer Briefs in Optimization. DOI:10.1007/978-3-030-16936-7

[52] Sajad Ahmad Rather, Aybike Özyüksel Çiftçioğlu, P Shanthi Bala, (2023). Lévy flight and Chaos theory based metaheuristics for grayscale image thresholding, Chapter 12 of Comprehensive Metaheuristics Algorithms and Applications edited by Seyedali Mirjalili and Amir H Gandomi, Academic Press. DOI: 10.1016/B978-0-323-91781-0.00012-0.

[53] Mantegna Rosario N. (1994). Fast, Accurate Algorithm for Numerical Simulation of Lévy Stable Stochastic Processes. Physical review, 5(49), 4677-4683 .

[54] Xin-She Yang. (2020). Nature-Inspired Computation and Swarm Intelligence Algorithms, Theory and Applications, Academic Press.
[55] Brent R P .(2000). Recent Progress and Prospects for Integer Factorisation Algorithms. COCOON 2000, LNCS 1858, 3–22 .DOI:10.1007/3-540-44968-X_2.
[56] Carella N A. Deterministic Integer Factorization Algorithms. arXiv:1308.2891v3 [cs.DS], 2022.DOI: 10.48550/arXiv.1308.2891
[57] Wang X.(2020. Algorithm Available for Factoring Big Fermat Numbers. Journal of Software, 15(3),86-97.DOI:10.17706/jsw.15.3.86-97
[58] Wang X. Source Code along with Experimental data of this paper.
https://drive.mathworks.com/sharing/6e4cf414-f286-4655-a2bc-cc1af222f0f8