Deterministic-Embedded Monte Carlo Approach to Find out an Objective Item in a Large Number of Data Sets

Xingbo Wang[1, 2 *], Jianxiang Guo[1]

[1] Department of Mechatronic Engineering, Foshan University, Foshan, China.
[2] Guangdong Engineering Center of Information Security for Intelligent Manufacturing System, Foshan, China.

* Corresponding author. Tel.: +86075782988845; email: xbwang@fosu.edu.cn, dr.xbwang@qq.com

Manuscript submitted April 12, 2019; accepted July 13, 2019.
doi: 10.17706/ijapm.2019.9.4.173-181

Abstract: The paper investigates an approach to find out an objective integer in a large integer interval. It first puts forward an approach to subdivide a large integer interval into small ones that are available for the Monte Carlo randomized search algorithm, then selects a small interval by the Monte Carlo algorithm and applies a deterministic search algorithm on the selected one. In order to make the search in an expected computing time, the paper proposes certain regulations to set an initial length for the small interval and to update it in accordance with the expectation of the time complexity. Mathematical foundations for setting-up the initial value and updating it to an acceptable value are presented and proved in detail and a parallel computing strategy is introduced to realized it. Except for the availability in integer factorization, the approach is also applicable in big data searches.

Key words: Subdivision, randomized algorithm, parallel computing, big data, integer factorization.

1. Introduction

Computer search or computer searching has been a primary function of computers, as introduced in kinds of schoolbooks like the Goodman's early book [1], Steven S. Skiena's [2] or Anany Levitin's [3] classical book. It is a truth that, algorithm design related with the computer searching has been a topic in the field of computer science, as H R Richardson and T L Corwin overviewed early in 1980 [4]. In recent years since 2017 and in several literatures, for example, in papers [5]-[9], a problem has been investigated to search rapidly an integer that hides somewhere in a large interval, for example, to find an integer that has a common divisor with the RSA100 in the large interval [32556519888 3223146736258950657270975 3 9753475571713,390205718554012655122895733394843710189050069001 94], which contains 3232 02598353947541933183913687863673957576566424 1 large odd integers. The papers [5], [6] and [7], proposed approaches by means of subdividing the large interval and performing parallel brutal searches (BS) on each subinterval, the paper [8] put forward a probabilistic search, and the paper [9] gave some criterions to see if the objective integer hides in an interval. It seems that, those papers broke out a way in finding a satisfactory solution for the problem. However, summarizing the common essences of the approaches proposed in those papers, one can see that, such a subdivision-plus-parallel-computing approach will essentially costs large computing resources because there is merely one subinterval containing the objective integer, and consequently, only one process assigned by the parallel computing system can find out the objective integer whereas the rest processes merely waste their time. Facing such a dilemma and considering the success of the Monte Carlo approach applied in integer factorization, the probabilistic searching strategy that is of efficiency and simplicity, as Juraj Hromkovic declaimed in his book [10], is naturally worthy of exploring.

This paper makes an investigation on applying a random approach to searching a large interval. It puts forward an approach that randomly selects subinterval from the subintervals subdivided from the large interval and searches the selected subinterval with proper deterministic algorithm.

2. Preliminaries

This section introduces symbols, definitions and lemmas that are necessary in later sections.

2.1. Symbols and Notations

Throughout this paper, an odd interval $[a,b]$ is a set of consecutive odd numbers that take a as their lower bound and b as their upper bound. For example, $[3,11]=\{3,5,7,9,11\}$. An interval can also abstractly denoted with the capital letter *I*. Symbol $x \in I$ means number x is an odd integer in interval *I*. Symbols $\cup$ and $\cap$ are respectively union and intersection of set operations; two odd intervals, I_1 and I_2, are said to have intersection and denoted by $I_1 \cap I_2 \neq \varnothing$ if they contain some common items, for example, $[3,11]\cap[7,19]\neq\varnothing$. Symbol $\lfloor x \rfloor$ is to express *x*'s floor function defined by $x-1<\lfloor x \rfloor \le x$, where x is a real number. Symbol $O(x)$ is the big *O* symbol defined in calculus.

2.2. Definitions

Definition 1 (Subdivision of Interval). A subdivision of interval I is to have finite intervals $I_1, I_2, \dots, I_\alpha$ such that $I_i \cap I_{i+1} = \varnothing$ and $I_1 \cup I_2 \dots \cup I_\alpha = I$, where i and α are positive integers, and I_i means the i^{th} subinterval.

Definition 2 (Expected Computing Times (ECT)). The ECT is a number that an algorithm is expected to compute the objective result successfully.

Remark 1. Hromkovic Juraj made it a rule in [10] that the ECT is not an infinite number. Hence the definition 2 makes it a rule that the ECT is not an infinite number, neither a static number. It may be changed to reach the goal of the task to be computed. For example, an ECT might be initialized a value 2^{16}, and be updated to 2^{32} or 2^{8} depending on the realization of the computing goal.

2.3. Lemmas

Lemma 1 (See in [10], pp. 38-39) Let $A_1, A_2, \dots, A_n$ be a finite collection of deterministic strategies algorithms; a randomized algorithm A is a probability distribution over the collection.

Lemma 2 (See in [11]) Let $\{x_i\}$ be a sequence of non-negative integers generated by

$$x_i \equiv a x_{i-1} + c \pmod m$$

then the sequence has full period m provided that

1) c is relatively prime to m;
2) $a \equiv 1 \pmod p$ if p is a prime factor of m;
3) $a \equiv 1 \pmod 4$ if 4 is a factor of m.

Particularly, if m is a power of 2, it suffices to have $a \equiv 1 \pmod 4$ and c odd.

Lemma 3. The number of odd integers contained in the odd interval $[a,b]$ is $\frac{b-a}{2}+1$.

Lemma 4 Let M be a positive integer and $X = \sum_{i=0}^{M-1} c_i 2^i$, where $c_i = 0$ or 1; then $X < 2^M$

Proof. Simply see the fact that X reaches its maximal value when $c_0 = c_1 = \ldots = c_{M-1} = 1$ and $X_{\max} = 2^M - 1 < 2^M$.

3. Method and Its Mathematical Foundations

Suppose o is an odd integer lying somewhere in a large odd interval $I = [s, e]$ that contains n odd integers; this section presents a search approach to find out o efficiently. The approach is based on a subdivision that subdivides the large interval I into finite small subintervals that are suitable for both parallel computing and random computing. This section also shows the mathematical validity of the approach.

3.1. Rule of Subdivision

In order to meet the needs of the Monte Carlo algorithm, a subdivision rule, which is briefly called a randomization oriented subdivision (ROS), is made as follows.

(1) In accordance with the proposed ECT number, define an odd integer N_{ECT} that is in the form of $4x+1$, namely,

$$N_{ECT} \equiv 1 (\bmod 4) \tag{1}$$

(2) Express n by

$$n = 2^M N_{ECT} + R \tag{2}$$

(3) Subdivide the interval I into two parts by

$$I = I_P \cup I_R \tag{3}$$

where I_P contains $2^M N_{ECT}$ consecutive odd integers, I_R contains R consecutive odd integers and $I_P \cap I_R = \varnothing$.

(4) Subdivide the interval I_P into 2^M subintervals each of which contains N_{ECT} odd integers. Consequently, the large interval I is subdivided into $2^M + 1$ subintervals among which each of the first 2^M ones contains N_{ECT} consecutive odd integers and the last one contains R consecutive odd integers, as depicted with Fig. 1.

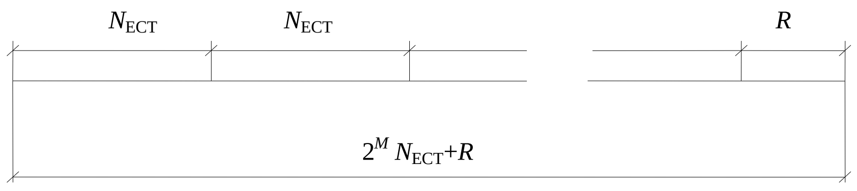

Fig. 1. Subdivision of a large interval.

From now on, an ROS is by default composed of an I_P part and an I_R with I_R containing R items and I_P containing 2^M subintervals each of which contains N_{ECT} items.

3.2. Rule of Search

Based on the ROS and a deterministic algorithm A_d, a randomized algorithm that is embedded with A_d, briefly called deterministic plus randomized search (DPRS), can be accomplished as follows.

```
Step 1. Select $I_R$ and search it with $A_d$;
    If $o$ is found, output it and stop.
Step 2. Begin random search
    $x_0 = 1$;
    Loop
    Calculate $x_n = x_{n-1}N_{ECT} + 2^{M-1} - 1(\bmod 2^M)$;
    Select the subinterval $I_{x_n}$ in $I_P$ and search it with $A_d$.
    If $o$ is found, output it and stop.
    End loop
```

3.3. Mathematical Foundations

This subsection presents the mathematical foundations to show that the ROS is a universal subdivision, and thus the DPRS can always be performed. The main contents are 3 theorems and 1 corollary.

Theorem 1. The DPRS can always find out the objective in the large interval. The searching time in the best case is $O(N_{ECT})$ while in the worst case it is $O(2^M N_{ECT})$ provided that $O(R) = O(N_{ECT})$.

Proof. The condition $O(R) = O(N_{ECT})$ means that the Step 1 in the DPRS can be accomplished in $O(N_{ECT})$. Therefore, if $o \in I_R$, it is surely found out in $O(N_{ECT})$. If $o \in I_P$ and it happens to lie in the subinterval that is picked by the first random selection, the searching time is $O(N_{ECT})$. By Lemma 2 it knows that, it requires 2^M picks to have each of the 2^M subintervals picked once. Since each subinterval in I_P contains N_{ECT} items, it knows that $O(2^M N_{ECT})$ is the time cost in the worst case that searches all the items in I_P.

Theorem 2. There always exists an N_{ECT} that enables $O(R) = O(N_{ECT})$.

Proof. First give an N_1 satisfying $N_1 \equiv 1(\bmod 4)$ and express n in terms of the Euclidean division by

$$n = aN_1 + r$$

where $0 \le r < N_1 a$.

Then express a by

$$a = 2^m + r_a \tag{4}$$

and then

$$n = 2^m N_1 + r_a N_1 + r \tag{5}$$

which is also

$$n = 2^{m+1} N_1 + (r_a - 2^m) N_1 + r \tag{6}$$

Let in (5) $R_0 = r_a N_1 + r$ and in (6) $R_1 = (r_a - 2^m)N_1 + r$; it can see

$$r_a N_1 \le R_0 < (r_a + 1) N_1 \tag{7}$$

Since the maximal value of r_a is $2^m - 1$, e.g., in the case $a = 2^m + 2^{m-1} + 2^{m-2} + \ldots + 2 + 1$, and $0 \le r < N_1$, it knows

$$R_1 < 0 \tag{8}$$

Obviously, if $R_0 \approx N_1$, then the formula (5) can be the basis of choosing N_{ECT} and R. Actually N_{ECT} and R can be chosen by $N_{ECT} = N_1$ and $R = R_0$, and it yields $O(R) = O(N_{ECT})$. If R_0 is much bigger than N_1, then the simplest choice of N_{ECT} and R is to take $N_{ECT} = N_1$ and $R = R_1$ in terms of the formula (6). The inequality $R < 0$ means that there is not an item in I_R, thereby it costs no computing time, which is $R = O(1) = O(N_{ECT})$.

Theorem 3. For a given positive integer N such that $N \equiv 1(\bmod 4)$, an arbitrary positive integer n satisfying $n > 2N$ can always be expressed by either

$$n = 2^M(2N-1) + R^* \tag{9}$$

or

$$n = 2^{M+1-\alpha}(2^\alpha(N-1)+1) + R \tag{10}$$

where $M = \left\lfloor \log_2 \left\lfloor \frac{n}{N} \right\rfloor \right\rfloor$, $0 \le R^* < 2^M$, $0 \le R < 2^{M-1}$ and $\alpha \ge 2$ are integers.

Proof. Here first presents a method to express n in the forms (5) and (6) as following 4 steps

Step 1. Take an initial N_1 such that $N_1 \equiv 1(\bmod 4)$;

Step 2. Calculate $a = \left\lfloor \frac{n}{N_1} \right\rfloor$ and $r = n - a \cdot N_1$;

Step 3. Calculate $m = \lfloor \log_2 a \rfloor$ and $r_a = a - 2^m$;

Step 4. Calculate $R_0 = r_a N_1 + r$ and $R_1 = (r_a - 2^m)N_1 + r$.

By now

$$n = 2^m N_1 + R_0 = 2^{m+1} N_1 + R_1$$

Since $R_1 < 0$, as proved in Theorem 2, let $N_2 = 2N_1 - 1$, which obviously satisfies $N_2 \equiv 1(\bmod 4)$; then it holds

$$\begin{aligned} n &= 2^m(2N_1-1) - 2^m N_1 + 2^m + r_a N_1 + r \\ &= 2^m N_2 - 2^m N_1 + 2^m + r_a N_1 + r \\ &= 2^m N_2 + (r_a - 2^m)N_1 + 2^m + r \\ &= 2^m N_2 + \underline{R_1 + 2^m} \end{aligned} \tag{11}$$

If $R_1 + 2^m \ge 0$, then take $N = N_2$ and $R = R_1 + 2^m$. Since $R_1 < 0$, it is sure $0 \le R < 2^m$. Referring to Theorem 2, it knows the case corresponding to (9) is true. Next is for the formula (10).

If $R_1 + 2^m < 0$, letting $N_3 = 2N_2 - 1$ and $R_2 = R_1 + 2^m$ yields

$$\begin{aligned} n &= 2^{m-1}(2N_2-1) + 2^{m-1} + R_1 + 2^m \\ &= 2^{m-1} N_3 + \underline{R_2 + 2^{m-1}} \end{aligned} \tag{12}$$

If $R_2 + 2^{m-1} \ge 0$, then take $N = N_3 = 2N_2 - 1 = 2(2N_1 - 1) - 1 = 2^2(N_1 - 1) + 1$ and $R = R_2 + 2^{m-1} = R_1 + 2^m + 2^{m-1}$ $= R_1 + 2^{m-1} \times 3$. Since $R_1 + 2^m < 0$, it is sure $0 \le R < 2^{m-1}$.

If $R_2 + 2^{m-1} < 0$, letting $N_4 = 2N_3 - 1$ and $R_3 = R_2 + 2^{m-1}$ yields

$$\begin{aligned} n &= 2^{m-2}(2N_3-1) + 2^{m-1} + R_2 + 2^{m-1} \\ &= 2^{m-2} N_4 + \underline{R_3 + 2^{m-1}} \end{aligned} \tag{13}$$

If $R_3 + 2^{m-1} \geq 0$, then take $N = N_4 = 2N_3 - 1 = 2^3(N_1 - 1) + 1$ and $2^{m-1} + 2^{m-1} = R_1 + 2^{m-1} \times 4$ $R = R_3 + 2^{m-1} = R_2 + 2^{m-1} + 2^{m-1} = R_1 + 2^m + 2^{m-1} + 2^{m-1} = R_1 + 2^{m-1} \times 4$. Since $R_2 + 2^{m-1} < 0$, it is sure $0 \leq R < 2^{m-1}$.

If $R_3 + 2^{m-1} < 0$, letting $N_5 = 2N_4 - 1$ and $R_3 = R_2 + 2^{m-1}$ yields

$$\begin{aligned} n &= 2^{m-3}(2N_4 - 1) + 2^{m-1} + R_3 + 2^{m-1} \\ &= 2^{m-3} N_5 + \underline{R_4 + 2^{m-1}} \end{aligned} \tag{14}$$

The formulas (11), (12), (13) and (14) indicate that, when $i > 2$ it always can find an $N_i = 2N_{i-1} - 1$ that makes it hold

$$n = 2^{m+2-i} N_i + R_{i-1} + 2^{m-1} = 2^{m+2-i} N_i + R_1 + 2^{m-1} i \tag{15}$$

and there must be an i that satisfies $R_1 + 2^{m-1}(i-1) < 0$ and $R_1 + 2^{m-1} i \geq 0$. Then taking $N_{ECT} = N_i$ and $R = R_1 + 2^{m-1} i$ leads to $R < 2^{m-1}$ Since $N_i = 2(N_{i-1} - 1) - 1 = 2^{i-1}(N_1 - 1) + 1$, it knows the theorem holds.

Remark 3. In practice, the initial N_1 is proposed to be the form $2^s + 2^t + 1$, where s and t are positive integers and $s > t \geq 0$.

Corollary 1. For a given large positive integer n and an initial positive integer N satisfying $N \equiv 1 (\bmod 4)$, an almost-proper N_{ECT} can always be obtained by means of adjusting (reducing or enlarging) the value of N.

Proof. Referring to the proof process, it knows that reducing the value of the N_{ECT} will make M bigger and R smaller and enlarging the value of the N_{ECT} will make M smaller and R bigger. Thereby, an N_{ECT} within a scope can always be obtained.

4. Application in Factoring Integers

Let O be a semiprime; Consider in a large interval I containing n odd integers in which an objective o that has a common divisor with O. This section presents how to apply the ROS subdivision and the DPRS search on integer factorization.

4.1. Hybrid Searching Procedure

The hybrid searching procedure means a random search (RS) is embedded with a brutal search (BS). The BS calculates the greatest common divisor (GCD) between O and each item in a subinterval, and the RS randomly selects the subinterval. Accordingly, the hybrid searching procedure is simply said to be an **RS+BS** procedure. Generally, the BS is designed to finish the search in the ECT. The procedure is designed as follows

```
==========RS+BS Procedure==========
Input: $O$, $n$;
Step 1. Determine the $N_{ECT}$ and $R$ by Theorem 2;
          thus $I$ can be subdivided into $I_P$ and $I_R$
          by the ROS.
Step 2. If $R \neq 0$, perform BS on $I_R$;
          If $o$ is found out, go to Step 4.
Step 3. Perform the DPRS procedure on $I_P$
          until o is found out.
Step 4. Output $o$ and return.
===========End of Procedure==========
```

4.2. Parallel Computing Strategy

Suppose there are n integers to be searched in large interval I and there are m processes taking part in the searching. Here proposes a two-stage subdivision (2S^2) to perform parallel computing.

Stage 1. By the Euclidean division, express *n* by

$$n = g(m-1) + r, 0 \le r < m-1$$

Then the first subdivision is to subdivide the *n* integers into *m* subintervals, $I_0, I_1, \ldots$, and I_{m-1}, among which there are $m-1$ subintervals each of which has the length *g* and the other one has the length *r*. The purpose to make such a subdivision is to assign each subinterval a process to perform the search.

Stage 2. Without loss of generality, suppose $I_{m-1}, I_{m-2}, \ldots$, and I_1 are of the length g and I_0 is the length *r*; assign the process 0 to search I_0, and each of the other processes to search one of the subintervals $I_{m-1}, I_{m-2}, \ldots$, and I_1, as illustrated by Fig. 2. Then perform ROS subdivision and DPRS on subintervals $I_{m-1}, I_{m-2}, \ldots$, and I_1; perform BS on I_0. Because there are merely *r* integers in I_0, the process 0 will soon finish its task and then it plays a role of communicating among the processes. The other processes continue their tasks until the objective integer is found.

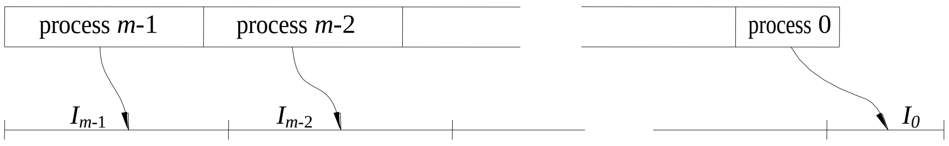

Fig. 2. Subintervals vs. computing processes.

Example. Use a computer with E5450 3.0GHz and 4GB RAM, perform MPI parallel computing with 6 processes. The applied algorithms are pure BS one, which perform BS on the whole interval, and the DPR one that is introduced above. The experiment results are list in Table 1. It can see the DPRS is generally faster than the pure BS and the global results quite match to Theorem 1. Meanwhile, if compared with the Pollard ρ's method [12], the DPRS is much faster. Fig. 3 depicts the data in the table.

Table 1. Comparison of Time Cost by Pollard ρ, Pure BS with DPRS

Index	Semiprimes	Divisor interval	Pollard ρ (ms)	BS (ms)	DPRS (ms)	Found divisor
N1	448316072600119	[13360861, 21173475]	3516	80	31	15402707
N2	35249679931198483	[131315291, 187748981]	1062	1375	340	59138501
N3	208127655734009353	[387667969, 456210100	24250	1954	750	430470917
N4	331432537700013787	[402653185, 575701778]	14641	3735	1343	114098219
N5	3070282504055021789	[1429711703, 1752222161]	380812	8578	672	1436222173
N6	1129367102454866881	[1051805077, 1062716849]	31	281	140	25869889
N7	32391050471337716069	[3770814565, 5691313598]	19281	1375	750	12152389
N8	29742315699406748437	[3462461255, 5453651593]	139891	6984	319	372173423
N9	10188337563435517819	[2372157193, 3191917537]	33063	1672	844	70901851
N10	24928816998094684879	[3221225473, 4992876625]	42018	24031	21922	347912923

5. Application in Big-Data Search

The finding integer problem can be of course an abstract model of the problems that find an objective item in a large number of data sets. For example, find out an item that exits in of m different data sets, $D_1, D_2, \ldots$ and D_m. Regarding one data set to be one subinterval subdivided from a large interval surely, the finding integer problem is surely an instance of finding out an objective item in a large number of data sets. This can surely draw out the following Proposition 1.

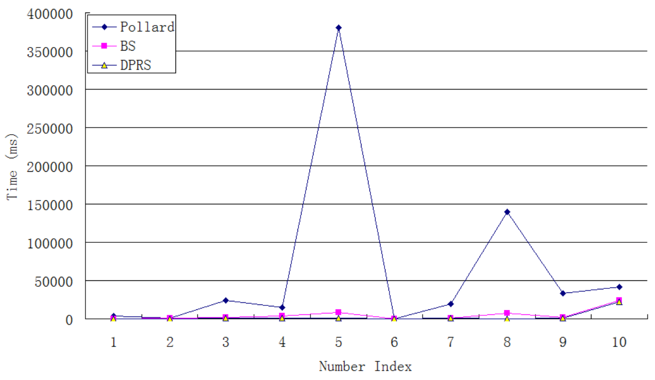

Fig. 3. Factorization time cost by Pollard ρ, BS and DPR search.

Proposition 1. Let *D* be a large number of data sets that collect *n* different data sets, $D_1, D_2, \ldots$ and D_n, where D_i is the i^{th} data set; if $D_i \cap D_{i+1} = \varnothing$ and $D_1 \cup D_2 \ldots \cup D_n = D$, then the approach and the procedures applying on integer factorization can also be available for finding an item in *D*.

6. Conclusion

It is a popular problem to search an objective item in an interval. The calculus provides many methods to perform the search. Unfortunately, those methods provided in the calculus are suitable for the continuous sets but not for the discrete sets. It is known that the divide-and-conquer philosophy is available for both the continuous sets and the discrete sets. The subdivision ideal adopted in this paper is surely a way of divide-and-conquer. To meet the needs of the Monte Carlo algorithm, it is necessary to make a special subdivision on a large interval. Since the Monte Carlo method can merely select a computed unit, performing deterministic algorithm on the selected unit is a natural consequence if the unit itself is an interval too. This paper merely realizes the stated thoughts. Actually, comparing to the Pollard ρ method, it can see that, the DPRS is an extensive Pollard ρ method because it extends the original Pollard ρ method that select one integer each time to the selection of a subinterval. This is one advantage of the DPRS; another advantage of the DPRS is that, it is available for both sequential computing and parallel computing because the approach can be applied on every computing process; and the third one, as the Proposition 1 stated, the approach can be applied on other aspects such as the big data searches. That is why this paper is so entitled.

On the other hand, since a randomized algorithm greatly depends on the pseudorandom number generator and there are other pseudorandom number generator except for the one mentioned by Lemma 2, the DPRS is improvable when utilizing some other pseudorandom number generator. This leaves a future task. Hope to see splendid work in the future.

Acknowledgment

The research work is supported by the State Key Laboratory of Mathematical Engineering and Advanced

Computing under Open Project Program No.2017A01, Department of Guangdong Science and Technology under project 2015A010104011, Foshan Bureau of Science and Technology under projects 2016AG100311, Project gg040981 from Foshan University. The authors sincerely present thanks to them all.

References

[1] Goodman, E. S., & Hedetniemi, S. T. (1977). *Introduction to the Design and Analysis of Algorithms*. McGraw-Hill, Inc.

[2] Steven, S. S. (2008). *The Algorithm Design Manual*. Springer New York.

[3] Anany, L. (2012). *Introduction to the Design & Analysis of Algorithms*. Pearson.

[4] Richardson, H. R., & Corwin, T. L. (1980). An overview of computer assisted search information processing. *Search Theory and Applications*, Springer US.

[5] Fu, D. (2017). A parallel algorithm for factorization of big odd numbers. *IOSR Journal of Computer Engineering*, *19(2)*, 51-54.

[6] Wang, X. (2017). Strategy for algorithm design in factoring RSA numbers. *IOSR Journal of Computer Engineering*, *19(3,ver.2)*, 1-7.

[7] Li, J. (2017). Algorithm design and implementation for a mathematical model of factoring integers. *IOSR Journal of Mathematics*, *13(I Ver. VI)*, 37-41.

[8] Li, J. (2018). A parallel probabilistic approach to factorize a semiprime. *American Journal of Computational Mathematics*, *8(2)*, 175-183.

[9] Wang, X. (2019). Two number-guessing problems plus applications in cryptography. *International Journal of Network Security*, *21(3)*, 498-504

[10] Hromkovic, J. (2005). Design and analysis of randomized algorithms. *Introduction to Design Paradigms*. Springer-Verlag GmbH .

[11] Hull, T. E., & Dobell, A. R. (1962). Random number generator. *SIAM Review*, *l.4(3)*, 230-254.

[12] Pollard, J. M. (1975). A monte carlo method for factorization. *BIT Numerical Mathematics*, *15*, 331-334.

Xingbo Wang was born in Hubei, China. He got his master and doctor's degree at National University of Defense Technology of China and had been a staff in charge of researching and developing CAD/CAM/NC technologies in the university. Since 2010, he has been a professor in Foshan University with research interests in computer application and information security. He is now the chief of Guangdong engineering center of information security for intelligent manufacturing system. Prof. Wang was in charge of more than 40 projects including projects from the National Science Foundation Committee, published 8 books and over 90 papers related with mathematics, computer science and mechatronic engineering, and invented 30 more patents in the related fields.

Jianxiang Guo was born in Hubei. He received a bachelor's degree at Hubei Polytechnic University and became a graduate student of Foshan University in 2018. He is now a member of Guangdong engineering center of information security for intelligent manufacturing system