Article

Not peer-reviewed version

An Efficient Simulated Annealing Algorithm for Short Addition Sequence

Hazem M. Bahig *, Mohamed A.G. Hazber, Hatem M. Bahig

Posted Date: 20 December 2023

doi: 10.20944/preprints202312.1556.v1

Keywords: addition sequence; addition chain; simulated annealing; heuristic algorithm

Article

An Efficient Simulated Annealing Algorithm for Short Addition Sequence

Hazem M. Bahig [1], Mohamed A.G. Hazber [1] and Hatem M. Bahig [2,*]

[1] Information and Computer Science Department, College of Computer Science and Engineering, University of Ha'il, Ha'il 81481, KSA; E-mail: h.bahig@uoh.edu.sa; m.hazber@uoh.edu.sa

[2] Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt; E-mail: hmbahig@sci.asu.edu.eg

* Correspondence: h.bahig@uoh.edu.sa

Abstract: An addition sequence is an important operation in many applications of computer science, such as multi-modular exponentiation and outsourcing protocols. Finding an addition sequence for a set of positive integers with the shortest length is challenging due to the high computational time required to find the solution. In this paper, a new metaheuristic algorithm is designed based on the simulated annealing strategy to generate a short addition sequence. The efficiency of the proposed algorithm was proved experimentally by comparing it with the previous exact and heuristic algorithms in terms of running time and the length of the addition sequence.

Keywords: Addition sequence; Addition chain; Simulated Annealing; Heuristic Algorithm

MSC: 11Bxx; 90C59

1. Introduction

Given a set of numbers $N = \{n_1, n_2, \dots, n_k\}$ such that $1 < n_1 < n_2 < \cdots < n_k$. An addition sequence [1,2] for the set N, denoted by ASeq(N), is an increasing sequence of numbers $as_0, as_1, as_2, \dots, as_l$ such that (1) $as_0 = 1$, (2) $as_l = n_k$, (3) $as_i = as_j + as_h, 0 \le j, h \le i-1$, and (4) $N \subseteq \{as_0, as_1, as_2, \dots, as_l\}$, i.e., each number n_i should appear in the sequence $as_0, as_1, as_2, \dots, as_l$.

The number l is called the length of ASeq(N). The minimal length of an ASeq(N) is denoted by $\ell(N)$. In the case of $k = 1$, the sequence is called addition chain [1,2].

The problem of generating a shortest *ASeq(N)* is equivalent to the simultaneous evaluation of k power monomials $g^{n_1}, g^{n_2}, \dots, g^{n_k}$ with a minimum number of multiplications.

For example, let $N = \{53, 163, 203, 363\}$. The following are two ASeqs with lengths 15 and 13, respectively.

The elements of the first ASeq are: 1, 2=1+1, 3=2+1, 6=3+3, 12=6+6, 13=12+1, 26=13+13, 39=26+13, 40=39+1, 53=40+13, 106=53+53, 159=106+53, 160=159+1, 163=160+3, 203=163+40, 363=203+160.

The elements of the second ASeq are: 1, 2=1+1, 3=2+1, 5=3+2, 10=5+5, 13=10+3, 20=10+10, 40=20+20, 53=40+13, 80=40+40, 160=80+80, 163=160+3, 203=163+40, 363=203+160

The evaluation of $g^{53}, g^{163}, g^{203}, g^{363}$, using the first sequence is

$$g, g^2, g^3, g^6, g^{12}, g^{13}, g^{26}, g^{39}, g^{40}, g^{53}, g^{106}, g^{159}, g^{160}, g^{163}, g^{203}, g^{363}$$

while the evaluation of the same powers $g^{53}, g^{163}, g^{203}, g^{363}$ using the second sequence is

$$g, g^2, g^3, g^5, g^{10}, g^{13}, g^{20}, g^{40}, g^{53}, g^{80}, g^{160}, g^{163}, g^{203}, g^{363}$$

A step i is called star if $as_i = as_{i-1} + as_h, 0 \le h \le i-1$; and non-star if $as_i = as_j + as_h, 0 \le j, h \le i-2$. In case of $j = h = i-1$, $as_i = 2as_{i-1}$, the step is called doubling. If all steps in the sequence are stars, then the sequence is called star. If $\ell^*(N)$ denotes to the minimal length of star ASeq(N), then we have

$$\ell(N) \le \ell^*(N) \quad (1)$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 December 2023 doi:10.20944/preprints202312.1556.v1

Yao [3] showed that:

$$\ell(N) \leq \log n_k + (c\ k) \log n_k / \log\log n_k, \quad (2)$$

for some constant $c = 2 + 4/\sqrt{\log n_k}$.

Bleichenbacher [4] computed the lower bound

$$\ell(N \cup \{n_{k+1}\}) \geq \ell(N) + \alpha + 1, \quad (3)$$

where $n_{k+1} > 2^{\alpha} n_k, \alpha \geq 0$

ASeqs have received a lot of consideration among mathematicians and computer scientists for the following reasons:

The first reason is that one of the fundamental operations that play an important role in the efficiency of many public key cryptosystems and protocols is group exponentiation (sometimes it is called *multi-modular exponentiation* [2,5]), i.e., computing $g^{n_1}, g^{n_2}, \ldots, g^{n_k}$ simultaneously with a minimal number of operations, where g is an element in a group. Designing a fast algorithm for generating a shortest (or short) ASeq increases the efficiency of such public key cryptosystems and protocols since evaluating $g^{n_1}, g^{n_2}, \ldots, g^{n_k}$ with a minimal number of multiplications is equivalent to finding a shortest ASeq(N).

The second reason is that ASeqs (including addition chains) are generalized to the following:

(i) B-chains [7], where every element in the B-chain has the form $a_i = a_j\ o\ a_h, 0 \leq j, h \leq i-1$, and the binary operation o belongs to a finite set of binary operations over the set of natural numbers B, i.e. $o \in B = \{+, -, *, \div\}$. Guzmán-Trampe et. al. [8] proposed a method for generating addition-subtraction, $B = \{+, -\}$, sequence for the Kachisa–Schaefer–Scott family of pairing-friendly elliptic curves.

(ii) Vectorial addition chain [9,10]: it is a sequence of k-dimensional vectors of nonnegative integers $v_i, -k+1 \leq i \leq l$, such that (1) $v_{-k+1} = [1,0,0 \ldots,0,0]$, $v_{-k+2} = [0,1,0 \ldots,0,0]$, …, $v_0 = [0,0,0 \ldots,0,1]$, (2) $v_i = v_j + v_h, 1 \leq i \leq l, -k+1 \leq j, h \leq i-1$, and (3) $v_l = [n_1, n_2, \ldots, n_k]$. Finding a shortest vectorial addition chain is equivalent to evaluating, multi-exponentiation, i.e., the product $\prod_{i=1}^{k} g_i^{n_i}$ with the minimal number of multiplications.

The third reason is that in Internet of Things, IoT, devices with limited resources have a problem when they perform some public-key primitives, such as decryption and signature, because most public-key primitives are *(i)* time-consuming compared with symmetric-key cryptosystems; and *(ii)* using private information. One of the common solutions to this problem is to use what is called "server aided secret computation protocols", denoted by SASCP [11], or sometimes it is called outsourcing protocols [12].

In SASCP, devices with limited power and resources, such as smart cards, can execute public-key primitives efficiently with the aid of an untrusted powerful server without revealing the private information. Examples of such protocols that used ASeqs are [6,11,13]. Other protocols and their security analysis are [12,14]. Another and similar solution to the problem is to define a delegation protocol. It is a protocol that satisfies two security objectives: *(i)* privacy: the private information should not be recovered by a passive attacker; and *(ii)* verifiability: the untrusted server should not be able to make the devices accept an invalid value as the result of the delegated computation. Examples of such protocols and their security analysis are [12,15–17].

The main challenge of finding a shortest ASeq, i.e., the minimal number of additions needed to compute all elements of N, that it is NP-complete [4]. Additionally, when the size of N is large and the size of exponents is large, the running time for finding ASeq is very large [18]. Therefore, designing a fast algorithm for generating a short (not necessarily shortest) ASeq is interesting using metaheuristics techniques such as simulated annulling, ant colony and evolutionary algorithms.

In this paper, a new metaheuristic algorithm based on simulated annealing strategy is proposed to find a short ASeq for the set N. The proposed simulated annulling algorithm for ASeq has three advantages over the previous ASeq algorithms. The first advantage is the designed algorithm has running time less than the exact algorithm. The second advantage is the length of ASeq generated by the proposed algorithm is shorter than the ASeq generated by previous heuristic algorithms. The

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 December 2023 doi:10.20944/preprints202312.1556.v1

third advantage is that there is no comparative study between suboptimal algorithms and the exact algorithm in terms of the length of ASeq.

The remainder of this paper is organized as follows. Section 2 includes the related works of ASeq. In Section 3, the details of the proposed algorithm is given. Section 4 includes the dataset used in the experiments, and the results and analysis of the experimental studies for the proposed simulated annulling algorithm and other algorithms. Finally, Section 5 includes the conclusion of this paper and the future works.

2. Related Works

Algorithms for generating ASeq, i.e. $k \geq 1$, can be classified into two categories. The first category is to find a shortest ASeq. In fact, there are a few papers that discussed a generation of shortest ASeqs. Bleichenbacher [4] suggested an algorithm to find a shortest ASeq(N) with length r provided that we previously computed $\ell(y)$ for all numbers $y < maximum(N)$, and $\ell(y) < r$. He used the suggested algorithm to generate a shortest addition chain up to a certain number.

The authors in [18] suggested a branch and bound depth-first search algorithm to generate a shortest ASeq for any set N. The algorithm starts by computing a lower bound, Eq(3), and looking for an addition chain for the first element n_1 in the set N. Then, it extends the chain to addition sequence for $\{n_1, n_2\}$, and so on until it generates ASeq(N). The algorithm uses different strategies to speed up the generation as follows. (*i*) Using bounding sequences to prune some branches in the search tree, which cannot lead to a shortest ASeq. (*ii*) Determining an upper bound of $\ell(n_1, n_2, \dots, n_i), 1 \leq i \leq k$. (*iii*) Using some sufficient conditions for star steps to skip the generation of non-star steps. *(vi)* If no ASeq(N) of length l is found, then the algorithm increases l by one and repeats the process until either l is equal to the length of the generated short ASeq produced by continued fraction (CF) method [19] or the algorithms finds a shortest ASeq. Recently, the authors in [20] used multicore systems to improve the generation of a shortest ASeq.

The second category is to find a short ASeq. Yao [3] presented an algorithm to compute g^{n_1}, $g^{n_2}, \dots, g^{n_k}$ in $O(\lg n_k + c\sum_{i=1}^{k}(\log n_i / loglog(n_i + 2))$ multiplications for some constant c. Bos and Coster [21] proposed four methods to generate a short ASeq to use it in the window method [21]. The upper bound of the length of generated an ASeq, by the four methods, could be estimated, experimentally, by

$$\ell(n_1, n_2, \dots, n_k) \leq \frac{3}{2}\log n_k + k + 1, \tag{4}$$

for $n_k \leq 1000$.

Bergeron et al. [19] proposed an efficient method based on CF. The suggested method can be considered as an extension and unifying approach of some previously known methods (such as binary and k-ary methods [1]) for generating a short addition chain, i.e., ASeq with k=1.

Recall that the CF expansion of n/d, denoted by $[c_1, \dots, c_t]$, is

$$\frac{n}{d} = c_t + \cfrac{1}{c_{t-1} + \cfrac{1}{\ddots + \cfrac{1}{c_2 + \cfrac{1}{c_1}}}} \tag{5}$$

where d is an integer in the range $[2..n-1]$.

Bergeron et al. [19] suggested different strategies for choosing the value of d. One of the efficient strategies that produces a good suboptimal ASeq is dichotomic strategy, where

$$d = \left\lfloor \frac{n}{2^{\lfloor \lfloor log_2 n \rfloor / 2 \rfloor}} \right\rfloor. \tag{6}$$

Let $N' = \{n_k, n_{k-1}, \dots, n_1\}$, and $\mathcal{L}(N')$ denotes the length of the ASeq(N') generated by CF using dichotomic strategy. Then

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 December 2023 doi:10.20944/preprints202312.1556.v1

$$\mathcal{L}(N') = \begin{cases} \mathcal{L}(N' \setminus \{n_k\}) + \ell(q), & if\ rem = 0; \\ \mathcal{L}(N' \setminus \{n_k\}) + \ell(q) + 1, & if\ rem = 1,2 \\ \mathcal{L}(N' \cup \{rem\} \setminus \{n_k\}) + \ell(q) + 1, & \text{otherwise} \end{cases}; \quad (7)$$

where $n_k = q\, n_{k-1} + rem$, and

$$\ell(n) = \begin{cases} \alpha, & if\ n = 2^{\alpha}; \\ 3, & if\ n = 3; \\ \mathcal{L}(\{n, d\}), & \text{otherwise} \end{cases} \quad (8)$$

where *d* is defined by Eq. (6).

Enge *et. al.* [22] proposed a special method to construct a short ASeq to find the first *k* nonzero terms in the sparse q-series belonging to the Dedekind eta function or the Jacobi theta constants. Nadia and Mourelle [23] used Anti Colony strategy to find a short ASeq. They tested the strategy for a small set of numbers. Abbas and Gustafsson [24] proposed a method based on integer linear programming to generate a short ASeq for a small set of numbers.

In all previous studies, there was not enough experimental study for generating a short ASeq with different sizes of the set *N*, or with different range values of each number in the set *N*. Also, there is no comparative study between suboptimal algorithms and the exact algorithm in terms of the length of ASeq.

3. The Proposed Method

In this section, we first present a brief description of the proposed algorithm that is based on simulated annealing strategy to find short ASeq, and then we present its details. The algorithm is named SAAS for simulated annealing addition sequence.

Initially, the algorithm starts by generating the initial state, AS_0, using the CF method [19], and its energy is equal to the length of AS_0, l_{AS_0}. Then, the algorithm assigns these two values to the best state and the best energy, respectively. After that, the algorithm repeats the following steps based on the number of Metropolis cycles, *m*, for a fixed temperature. In each iteration of this loop, the algorithm performs the following steps:

The first step is generating a new state, AS_{new}, and its energy, l_{new}. The second step is determining whether the algorithm accepts this new state or not. The algorithm accepts the new state and its energy, and then assigns these values to the best state and best energy if either of the following conditions is true. (1) if the energy of the new state is lower than the energy of the best state. (2) If the Boltzmann distribution is greater than a random real number in the range [0,1].

After completing the number of Metropolis cycles for a fixed temperature, the algorithm updates the temperature using the Kirkpatrick quenching method and repeats this process until it reaches the maximum number of annealing iterations.

The details of the algorithm steps are as follows.

Step 1: Generate the initial state, AS, using CF method for the set of exponents $N = \{n_1, n_2, \dots, n_k\}$, where $AS = \{as_0, as_1, as_2, \dots, as_l\}$ such that (1) $as_0 = 1$ and $as_1 = 2$, (2) $\exists\, i, l_i$ s.t. $n_i = as_{l_i}$ and $1 \le i \le k$, (3) $L = \{l_1, l_2, \dots, l_k\}$ such that $l_i < l_{i+1}$ and $l = l_k$.

Step 2: Repeat the following *m* times:

Step 2.1: Generate a random integer number, *r*, from the range $[0, k-1]$. This number will be used as a start point of mutation based on the elements of *N*.

Step 2.2: Generate a new state, AS_{new}, by mutate AS, from the location l_r. If *r*=0, then the algorithm mutates AS from the element $as_1 = 2$. Otherwise, the algorithm mutates the state AS from $as_{l_r} = n_r$ to n_k. The process of generating the new elements from n_i to n_{i+1} is based on the following rules.

- Rule # 1: Doubling the current element, $as_{j+1} = 2\, as_j$.
- Rule # 2: Summing the last two elements, $as_{j+1} = as_j + as_{j-1}$.
- Rule # 3: Summing the last element with any other random element in the sequence, $as_{j+1} = as_j + as_h$, $0 \le h < j$.

This step can be done as follows (Steps 2.2.1-2.2.3).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 December 2023 doi:10.20944/preprints202312.1556.v1

Step 2.2.1 (Generate one element in the sequence): If the current goal is n_{i+1} and the current ASeq is $\{as_0,\ as_1, \dots, as_{l_i} = n_i, as_{l_i+1}, \dots, as_{l_i+j}\}, j \geq 0$, then the steps of generating a new element in the chain are as follows.

- $d = n_{i+1} - as_{l_i+j}$
- If $d = as_{l_i+j}$ then apply rule # 1
- Else if $d = as_{l_i+j-1}$ then apply rule # 2
- Else if $d > as_{l_i+j}$ then
- Generate a random real number $\alpha \in [0,1]$
- If $\alpha \geq 0.5$ then apply the rule # 1
- Else
- Generate a random real number $\alpha \in [0,1]$
- If $\alpha \geq 0.5$ then apply the rule # 2
- Else
- Generate a random integer number $r \in [0, l_i + j - 2]$
- Apply the rule # 3, where *h=r*.
- Else // $d < as_{l_i+j}$
- Generate a random integer number $r \in [0, l_i + j - 2]$
- Apply the rule # 3, where *h=r*.
- If the new element is less than or equal to n_{i+1} then the element is
- accepted. Otherwise, decrease the value of *r* and apply *rule #3* until
- we found a certain value of *h* such that the new element is less than or
- equal to n_{i+1}.

Step 2.2.2 (Generate all elements between n_i and n_{i+1}): Repeat Step 2.2.1 starting from $j = 0$, and $as_{l_i} = n_i$, until the algorithm finds $as_{l_i+j_i} = n_{i+1}$. In this case, the algorithm updates the value of $l_{i+1} = l_i + j_i, 1 \leq j_i$.

Step 2.3.3 (Generate the ASeq from n_r to n_k): Repeat Steps 2.2.1 and 2.2.2 until generate n_k. Therefore, $AS_{new} = \{as_0, as_1, \dots, n_r = as_{l_r}, as'_{l_r+1}, \dots, n_{i+1} = as'_{l_i+j_i}, \dots, n_k = as'_{l_{k-1}+j_{k-1}} = as'_{j_{k-1}}\}, j_i \geq 1$; and $L_{new} = \{l_1,\ l_2, \dots, l_r, l'_{r+1}, l'_{r+2}, \dots, l'_k\}$.

Step 3 : Test the acceptance of the new state by the following substeps.

- If $l'_k < l$ then $AS = AS_{new}$ and $l = l'_k$
- Else generate a random real number r.
- $d_e = l'_k - l$
- If $e^{-d_e/T} > r$ then $AS = AS_{new}$ and $l = l'_k$

Step 4: Decrease the temperature using Kirkpatrick quenching method: $T = \gamma\, T$, where $\gamma = 0.99$.

Step 5: Repeat Steps 2, 3, and 4 until reach the maximum number of annealing iterations.

The complete pseudocodes for the new proposed SAAS is given in Algorithms 1 and 2.

Algorithm 1: SAAS

Input: $N = \{n_1, n_2, \dots, n_k\},\ T_0,\ \gamma$, *succNo*, *mterropolisNo*

Output: $AS = \{as_0, as_1, \dots, as_{l_k}\}$

```
1. T = T_0
2. CFAS(N, AS, L, l)                    // L = {l_1, ..., l_r, ... l_k}, l = l_k.
3. for suc = 0 to suc < sucNo do
4.     for m = 1 to meteropolisN do
5.         Generate a random integer number r ∈ [0, k − 1]
6.         MutateAS(AS, L, r, AS_new, L_new)
7.         d_l = l_new − l
8.         if d_l < 0 then
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 December 2023 doi:10.20944/preprints202312.1556.v1


```
            AS = AS_new
            L = L_new
            l = l_new
        Else
            Generate a random real number r' ∈ [0,1]
            if e^(-d_l/T) > r' then
                AS = AS_new
                l = l_new
                L = L_new
    T = γ T
```

Algorithm 2: *MutateAS*

Input: $AS = \{as_0, as_1, \dots, as_{l_k}\},\ L = \{l_1, \dots, l_r, \dots l_k\},\ 1 \le r < k.$

Output: $AS_{new} = \{as_0', as_1', \dots, as_{l_k'}'\}\ , L_{new} = \{l_1', l_2', \dots, l_{k'}'\}.$

```
for i = 1 to r do
    l_i' = l_i
for i = 0 to l_r do
    as_i' = as_i
for i = r to k − 1 do                    // from n_i we generate n_{i+1}
    j = 0
    while as'_{l_i+j} ≠ as_{l_{i+1}} (= N_{i+1}) do
        d = as_{l_{i+1}} − as'_{l_i+j}
        if d = as'_{l_i+j} then as'_{l_i+j+1} = 2 as'_{l_i+j} , j = j + 1
        else if d = as'_{l_i+j−1} then as'_{l_i+j+1} = as'_{l_i+j} − as'_{l_i+j−1} , j = j + 1
            else if d > as'_{l_i+j} then
                Generate a real random number between 0 and 1, say x.
                if x > 0.5 then as'_{l_i+j+1} = 2 as'_{l_i+j} , j = j + 1
                else
                    Generate a real random number between 0 and 1, say x
                    if x > 0.5 then as'_{l_i+j+1} = as'_{l_i+j} + as'_{l_i+j−1} , j = j + 1
                    else
                        Generate an integer random number between 0 and l_i + j − 2, say x.
                        as'_{l_i+j+1} = as'_{l_i+j} + as'_x , j = j + 1
                else //d_as < as_{l_i+j}
                    Generate an integer random number between 0 and l_i + j − 2, say x.
                    as'_{l_i+j+1} = as'_{l_i+j} + as'_x , j = j + 1
                    while as'_{l_i+j} > as_{l_{i+1}} do
                        x = x − 1
                        as'_{l_i+j} = as'_{l_i+j−1} + as'_x
    l'_{i+1} = i + j
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 December 2023 doi:10.20944/preprints202312.1556.v1

3. Results and Discussions

This section demonstrates the experimental study and its analysis for measuring the performance of the SAAS algorithm compared to the exact and heuristic solutions, ExAS and CFAS, respectively. The three algorithms were programmed using the C language and run on a machine with a processor of speed of 2.5 GHz and a memory of 16 GB. Also, the three algorithms were compared by measuring the execution time in milliseconds and the length of the short/shortest sequence. The section consists of two subsections: data generation and results.

3.1. Data Generation

The data used in the experimental study is based on two factors. The first factor is the number of elements k in the set of exponents N. The experimental values of k are 2, 4, 6, 8, and 10. The second factor is the domain of each exponent in the set N. According to the window method and its variations, the range of exponents is the integer interval $[1, 2^e - 1]$, where e is the window length (of size e-bits). Also, according to the performance of the window method, the value of each exponent should be odd. The experimental values of e are equal to 7, 8, 9, and 10. The reason for starting the values of e with 7, the running times for all compared algorithms are fast when $e < 7$.

The methodology of generating the ASeq is based on fixing the size of the window, i.e., e-bits, say e=7, and then generating different sets $N_{k,e}$ with lengths k= 2, 4, 6, 8, and 10. For each value of k, the algorithm generates 25 sets of exponents in the range $[1, 2^e - 1]$. The process of generating different sets of exponents is as follows.

1. Set e to the maximum number of bits in the exponents, i.e., the window size.
2. Set the set $N_{0,e} = \emptyset$ and i=2.
3. While $i \leq k = 10$ do the following
4. Construct a new set $N_{i,e}$, by adding two randomly odd numbers, in the range $[1..2^e - 1]$, to the set $N_{i-2,e}$, $i.e.$, $N_{i,e} = N_{i-2,e} \cup \{$ the two generated randomly odd numbers$\}$
5. Set i=i+2.
6. Make sure that $N_{i,e}$ is sorted.
7. Repeat Steps 2-4, 25 times to generate 25 sets of exponents with at most e-bits.
8. Repeat Steps 1-5 for different size of exponents e=7, 8, 9, and 10.

The following example illustrates the generation of five sets with different values of k and fixed size of exponents e=8.

- $N_{2,8} = \{177, 241\}$.
- $N_{4,8} = \{65, 125, 177, 241\}$.
- $N_{6,8} = \{65, 89, 125, 177, 189, 241\}$.
- $N_{8,8} = \{43, 65, 89, 125, 177, 189, 221, 241\}$.
- $N_{10,8} = \{43, 65, 89, 103, 125, 177, 189, 203, 221, 241\}$

3.2. Results

The results of implementing the three algorithms on the generated data in terms of the length of the output are shown in Table 1. The first two columns represent the two factors e and k, while the three last columns represent the percentage of differences in the lengths of the output for the following cases: (1) ExAS and SAAS algorithms, (2) ExAS and CFAS algorithms, and (3) SAAS and CFAS algorithms. Since the exact algorithm always produces the shortest ASeq, the methodology for analyzing the results is computing the number of cases in which the lengths of ASeqs generated by the SAAS and CFAS algorithms are greater than the shortest ASeq generated by the ExAs algorithm. The percentages of these cases represent the third and fourth columns. Also, Table 1 presents the difference between the lengths of the ASeqs generated by the SAAS algorithm and those generated by the CFAS algorithm, see the, the last column in Table 1.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 December 2023 doi:10.20944/preprints202312.1556.v1

Table 1. Comparison between three algorithms in terms of the length of ASeq.

		Percentage of cases when		
e	k	$\|AS_{SAAS}\| > \|AS_{ExAS}\|$	$\|AS_{CFAS}\| > \|AS_{ExAS}\|$	$\|AS_{CFAS}\| > \|AS_{SAAS}\|$
7	2	12%	28%	16%
	4	40%	64%	36%
	6	56%	76%	44%
	8	76%	84%	20%
	10	88%	92%	20%
8	2	20%	56%	44%
	4	32%	68%	52%
	6	80%	92%	40%
	8	82%	92%	36%
	10	92%	96%	32%
9	2	16%	56%	44%
	4	44%	80%	56%
	6	84%	88%	36%
	8	92%	96%	32%
	10	96%	100%	28%
10	2	16%	72%	60%
	4	52%	80%	32%
	6	88%	100%	24%
	8	92%	100%	16%
	10	100%	100%	16%

The analysis of data results shows the following observations.

First, as in Table 1, the percentage of the difference between the lengths of the ASeq generated by the exact algorithm, ExAS, and the heuristic algorithms, SAAS and CFAS, increases with the increase in the number of elements in the set N. For example, for fixed e=7 and k=2, 4, 6, 8, and 10, the percentages of cases that the exact algorithm generates ASeq with a length less than that generated by the SAAS algorithm are 12%, 40%, 56%, 76%, and 88%. Similarly, for the CFAS algorithm, the differences are 28%, 64%, 76%, 84%, and 92%.

Second, as in Table 1, the comparison between the lengths of ASeqs generated by the SAAS and CFAS algorithms, independent of the ExAS algorithm, , is presented in the last column. The data shows that the SAAS algorithm outperforms the CFAS algorithm in terms of the short ASeq for all studied cases.

Third, the length of the output generated by the SAAS algorithm is near to the shortest length compared to that generated by the CFAS algorithm. Figure 1 shows the distribution of difference between the length of the output for the SAAS algorithm (similarly the CFAS algorithm) and the output of the exact algorithm. It is clear that the SAAS algorithm generates short ASeq with lengths that are near to the shortest ASeq than that generated by the CFAS algorithm. For example, when e=8 and k=2, there are 20% of the instances where the length of ASeq generated by the SAAS algorithm is greater by one than the length of ASeq generated by the ExAS algorithm. On the other side, using the CFAS algorithm, there are 44% and 20% of instances have lengths greater than the shortest by one and two, respectively.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 December 2023 doi:10.20944/preprints202312.1556.v1

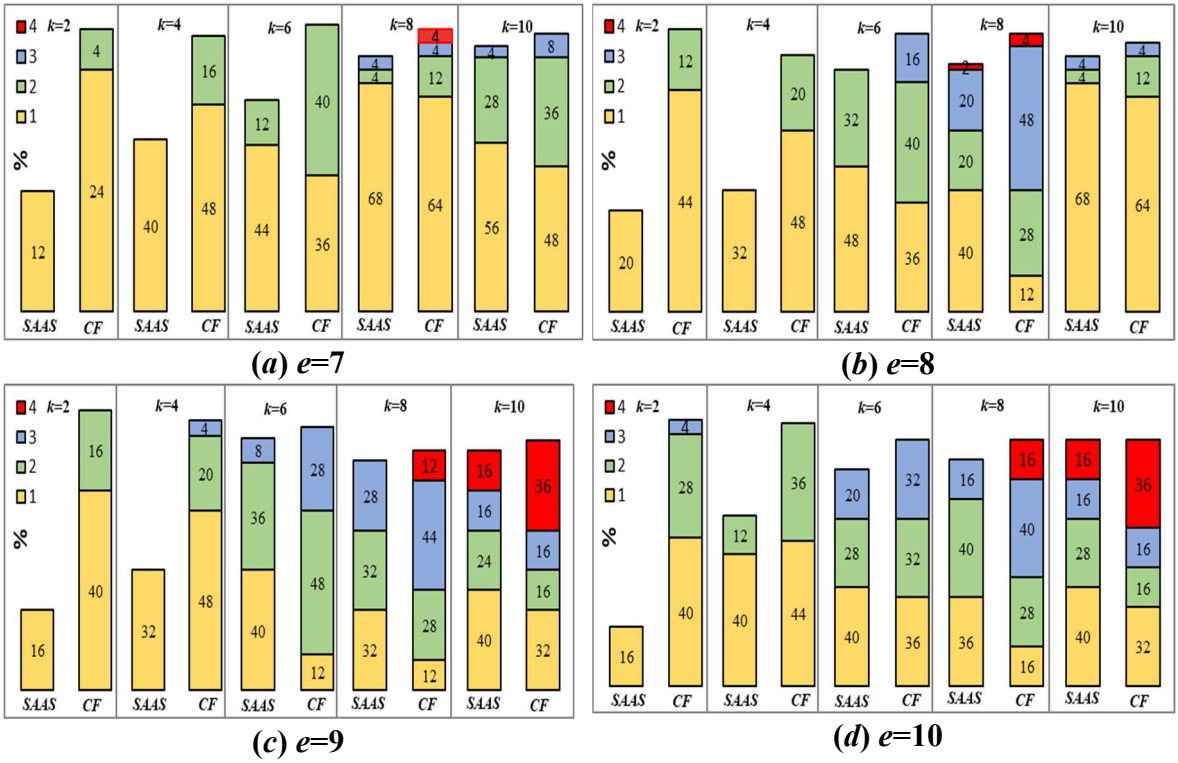

Figure 1. Percentage of differences in terms of the length of ASeq for the cases: (i) ExASt & SAAS, and (ii) ExaAS & CFAS. The bar in the figure contains four colors at maximum. The gold, green, blue and red colors represent the percentage of cases that have difference equal to 1, 2, 3, and 4, respectively. The figure includes four subfigures in case of (a) e=7, (b) e=8, (c) e=9, and (d) e=10. Each subfigure contains five pairs of bars, one bar for SAAS algorithm and the other for CFAS algorithm. The five pairs of bars represent the five cases k=2,4, 6, 8, and 10.

The comparison between the three algorithms, ExAS, SAAS, and CFAS, in terms of execution time is shown in Table 2. The analysis of data in the table demonstrates the following observations. (1) The fastest running time for all compared algorithms is CFAS algorithm. (2) The CFAS algorithm is not affected by the values of *e* and *k* in general. On the other side, the SAAS algorithm is slightly affected by increasing *e* and *k*, whereas the ExAS algorithm is significantly affected by increasing *e* and *k*. (3) The running time for the SAAS algorithm is affected by the two parameters, *succNo* and *metropolis*. The increase in the values of two parameters leads to a slight increase in the running time. (4) The running time for SAAS algorithm is faster than the exact algorithm, and the difference between the two algorithms in running time increases with increase in *e* and *k*. (5) The last column of Table 2 shows the percentage improvement for the SAAS algorithm compared to the exact algorithm.

Table 2. Comparison between different algorithms in terms of running time in milliseconds.

e	k	Exact Alg.	SAAS Alg.	CF Alg.	% of improvement SAAS & ExAS
7	2	10	65	1	---
	4	12	76	1	---
	6	14	89	1	---
	8	16	95	2	---
	10	17	99.4	2	---
8	2	12.44	81.84	1	---
	4	107	101.1	2	5.51%
	6	175	112.2	2	35.89%
	8	245	114.52	3	53.26%

	10	307	116.3	4	62.12%
9	2	13	107.24	2	---
	4	423	131.23	2	68.98%
	6	4375.64	144.52	3	96.70%
	8	14782.28	158.24	4	98.93%
	10	46592.12	162.2	4	99.65%
10	2	14.92	147.36	4	---
	4	57827.28	166.6	4	99.71%
	6	805166.32	177.56	16	99.98%
	8	15878846.4	185.7	16	100%
	10	58645310	197.2	18	100%

5. Conclusion and Future Works

In this paper, finding a short addition sequence for a set of positive integers was studied. A new metaheuristic algorithm was proposed to find an addition sequence with short length. The proposed algorithm starts with generating sequence using continued fraction and then apply the simulated annealing strategy to improve the length of the sequence. The proposed algorithm is fast compared to the exact algorithm and able to generate addition sequence with length less than the previous heuristic algorithm.

The efficiency of the proposed algorithm was conducted with considering different parameters such as the number of elements in the set and the size of positive integer.

There are many research directions related to addition sequence such as (1) extend the concept of B-chains and verctorial chain to ASeq, (2) use high-performance system to accelerate the computation of ASeq, and (3) accelerating the multi-modular exponentiation used ASeq.

Author Contributions: Conceptualization, Hazem B. and Hatem M.; methodology, Hazem B. and Hatem M.; software, Hazem B. and Hatem M.; validation, Hazem B., Hatem B.; formal analysis, Hazem B.; data curation, Hatem B.; writing—original draft preparation, Hazem B., M. H., and Hatem M.; writing—review and editing, Hazem B., M.H., and Hatem B.; visualization, Hazem B., and M.H.; supervision, Hazem B.; project administration, Hazem B.; funding acquisition, Hazem B.

Funding: This research has been funded by Scientific Research Deanship at University of Ha'il - Saudi Arabia through project number IFP-22 025.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to acknowledge the support provided by Scientific Research Deanship at University of Ha'il - Saudi Arabia through project number IFP-22 025.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Knuth, D.E . The art of computer programming: seminumerical algorithms, vol 2, 3rd edn. Addison-Wesley, Reading, pp 461–485, (1997)
2. Menezes, A.; van Oorschot, P.; Vanstone, S. Handbook of Applied Cryptography, CRC Press, Boca Raton, 1996 (Chapter 14)
3. Yao, A.. On the Evaluation of Powers. SIAM Journal on Computing, Vol.5, No.1, pp.100-103,, 1976. Doi:10.1137/0205008.
4. Bleichenbacher, D. Efficiency and security of cryptosystems based on number theory, chap 4. A Docotor Thesis, Swiss Federal Institue of Technology Zurich, Zurich, 1996. https://www.research-collection.ethz.ch/handle/20.500.11850/142613
5. Fathy, K.; Bahig, H.; Farag, M.. Speeding up multi-exponentiation algorithm on a multicore system. Journal of the Egyptian Mathematical Society, 26(2), 235-244, 2018. DOI: 10.21608/joems.2018.2540.1008.
6. Laih, C.; Yen, S.; Harn. L. Two efficient server-aided secret computation protocols based on the addition sequence. In: Advances in cryptology-ASIACRYPT'91, pp 450–459, 1991

7. Bahig, H. M.; Nassr, D. I. Generating a Shortest B-Chain using Multi-GPUs. Information Sciences Letters 11 (3), 745-750 , 2022.
8. Juan E. Guzmán-Trampe, Nareli Cruz-Cortés, Luis J. Dominguez Perez, Daniel Ortiz-Arroyo, Francisco Rodríguez-Henríquez. Low-cost addition–subtraction sequences for the final exponentiation in pairings. Finite Fields and Their Applications, Vol. 29, 2014, pp. 1-17, https://doi.org/10.1016/j.ffa.2014.02.009.
9. Thurber, E,; Clift, N. Addition chains, vector chains, and efficient computation. Discret. Math., 344(2):112200, 2021.
10. Downey, P.; Leong, B.; Sethi, R. Computing sequences with addition chains. SIAM J Comput 10(3):638–646, (1981)
11. Laih, C.; Yen, S. Secure addition sequence and its applications on the server-aided secret computation protocols. In: Advances in cryptology-AUSCRYPT'92, lecture notes in computer science, vol 718, pp 219–229, 1992
12. Bouillaguet, C.; Martinez, F.; Vergnaud, D. Cryptanalysis of Modular Exponentiation Outsourcing Protocols, The Computer Journal, Volume 65, Issue 9, September 2022, Pages 2299–2314, https://doi.org/10.1093/comjnl/bxab066
13. Matsumoto, T.; Kato, K.; Imai, H. Speeding Up Secret Computations with Insecure Auxiliary Devices. In: Goldwasser, S. (eds) Advances in Cryptology — CRYPTO' 88. CRYPTO 1988. Lecture Notes in Computer Science, vol 403. Springer, New York, NY. (1990)https://doi.org/10.1007/0-387-34799-2_35
14. Nguyen, P.; Shparlinski, I.E.. On the Insecurity of a Server-Aided RSA Protocol. In: Boyd, C. (eds) Advances in Cryp-tology — ASIACRYPT 2001. ASIACRYPT 2001. Lecture Notes in Computer Science, vol 2248. Springer, Berlin, Heidelberg. (2001)https://doi.org/10.1007/3-540-45682-1_2
15. Chen, X.; Li, J.; Ma, J.; Tang, Q.; Lou, W. New Algorithms for Secure Outsourcing of Modular Exponentiations. in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 9, pp. 2386-2396, Sept. 2014, doi: 10.1109/TPDS.2013.180.
16. Chevalier, C.; Laguillaumie, F.; Vergnaud, D. Privately outsourcing exponentiation to a single server: cryptanalysis and optimal constructions. Computer Security – ESORICS 2016, 261-278. https://doi.org/10.1007/978-3-319-45744-4_13
17. G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain. Delegating a Product of Group Exponentiations with Application to Signature Schemes. J. Math. Cryptol. 2020; 14:438–459
18. Bahig, H.M.; Bahig, H.M. A new strategy for generating shortest addition sequences. Computing 2011, 91, 285–306.
19. Bergeron, F.; Berstel, J.; Brlek, S. Efficient computation of addition chains. J Theor Nombres Bord 6:21–38, 1994
20. Bahig, H.M.; Kotb, Y. An Efficient Multicore Algorithm for Minimal Length Addition Chains. *Computers* 2019, *8*, 23. https://doi.org/10.3390/computers8010023.
21. Bos, J., Coster, M. (1990). Addition Chain Heuristics. In: Brassard, G. (eds) Advances in Cryptology — CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol 435. Springer, New York, NY. https://doi.org/10.1007/0-387-34805-0_37
22. Enge, A.; Hart, W.; Johansson, F. Short Addition Sequences for Theta Functions. Journal of Integer Sequences, Vol. 21, 3 (2018), Article 18.2.4.
23. Nedjah, N., de Macedo Mourelle, L. (2005). Efficient Pre-processing for Large Window-Based Modular Exponentiation Using Ant Colony. In: Khosla, R., Howlett, R.J., Jain, L.C. (eds) Knowledge-Based Intelligent Information and Engineering Systems. KES 2005. Lecture Notes in Computer Science(), vol 3684. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11554028_89
24. Abbas, M.; Gustafsson, O. Integer Linear Programming Modeling of Addition Sequences with Additional Constraints for Evaluation of Power Terms. 2023. https://doi.org/10.48550/arXiv.2306.15002