Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

A New Algorithm for Computing the Frobenius Number

Abbas Taheri*[1] and Saeid Alikhani†[2]

[1]Department of Electrical Engineering, Yazd University, 89195-741, Yazd, Iran
[2]Department of Mathematical Sciences, Yazd University, 89195-741, Yazd, Iran

ABSTRACT

A number α has a representation with respect to the numbers $\alpha_1, ..., \alpha_n$, if there exist the non-negative integers $\lambda_1, ..., \lambda_n$ such that $\alpha = \lambda_1\alpha_1 + ... + \lambda_n\alpha_n$. The largest natural number that does not have a representation with respect to the numbers $\alpha_1, ..., \alpha_n$ is called the Frobenius number and is denoted by the symbol $g(\alpha_1, ..., \alpha_n)$. In this paper, we present a new algorithm to calculate the Frobenius number. Also we present the sequential form of the new algorithm.

ARTICLE INFO

Article history:
Research paper
Received 12, February 2024
Accepted 18, September 2024
Available online 20, December 2024

Keyword: Algorithm complexity, Frobenius, Number theory.

AMS subject Classification: 01B39, 11D04.

*Email:a.taheri@stu.yazd.ac.ir
†Corresponding author: S. Alikhani. Email:alikhani@yazd.ac.ir

1 Introduction

Let $\alpha_1, ..., \alpha_n$ $(n \geq 2)$ be positive integers with $\gcd(\alpha_1, ..., \alpha_n) = 1$. Finding the largest positive integer N such that the Diophantine equation $\alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_n x_n = N$ has no solution in non-negative integers is known as the Frobenius problem. Such the largest positive integer N is called the Frobenius number of $\alpha_1, ..., \alpha_n$. Various results of the Frobenius number have been studied extensively. Some of the applications of the Frobenius number are change-making problem, scheduling problems, the complexity analysis of the Shell-sort method, Petri nets, partition of a vector space, monomial curves, algebraic geometric codes, titling and generating random vectors. Let explain one the application in more details. The Frobenius number can be used to model certain scheduling problems. For instance, if you have tasks with specific durations (represented by the integers), the Frobenius number can help determine the latest possible completion time for a set of tasks given their deadlines. We refer the reader to [7] for study of of the applications of the Frobenius number.

The Frobenius problem is well known as the coin problem that asks for the largest monetary amount that cannot be obtained using only coins in the set of coin denominations which has no common divisor greater than 1. This problem is also referred to as the McNugget number problem introduced by Henri Picciotto. The origin of this problem for $n = 2$ was proposed by Sylvester (1884), and this was solved by Curran Sharp (1884), see [6, 9]. Curran Sharp [6] in 1884 proved that $g(\alpha_1, \alpha_2) = \alpha_1\alpha_2 - \alpha_1 - \alpha_2$.

Let $\alpha_1, ..., \alpha_n$ be positive integers whose greatest divisor is equal to one, in other words

$$\gcd(\alpha_1, ..., \alpha_n) = 1.$$

If $S =< \alpha_1, ..., \alpha_n >$ is the semigroup generated by $\alpha_1, ..., \alpha_n$, then finding $g(S)$ is a problem and therefore finding the bounds for $g(S)$, whenever we have a certain sequence of numbers, [4] is of interest. For example, if S is an arithmetic sequence with relative value d, then we have [5]:

$$g(a, a + d, a + 2d, ..., a + kd) = a\lfloor\frac{a-2}{k}\rfloor + d(a - 1).$$

Fibonacci sequence is a recursive sequence $F_n = F_{n-1}+F_{n-2}$, $n \geq 3$ with $F_1 = F_2 = 1$. For every integer $l \geq i+2$, we have $g(F_i, F_{i+1}, F_l) = g(F_i, F_{i+1})$. Assuming $gcd(F_i, F_j, F_l) = 1$ for the triplet $3 \leq i < j < l$, calculating $g(F_i, F_j, F_l)$ has been considered. Suppose that $i, k \geq 3$ are integers and $r = \lfloor\frac{F_i-1}{F_i}\rfloor$. In this case (see [10])

$$g(F_i, F_{i+2}, F_{i+k}) = \begin{cases} (F_i - 1)F_{i+2} - F_i(rF_{k-2} + 1); & \text{If } r = 0 \text{ or } r \geq 1 \text{ and} \\ & F_{k-2}F_i < (F_i - rF_k)F_{i+2}, \\ r(F_k - 1)F_{i+2} - F_i((r - 1)F_{k-2} + 1) & \text{otherwise} \end{cases}$$

It was shown by Curtis [2] that no closed formula exists for the Frobenius number if $n > 2$. Because of this reason, there has been a great deal of research into producing

upper bounds on $g(a_1, a_2, ..., a_n)$. These bounds share the property that in the worst-case they are of quadratic order with respect to the maximum absolute valued entry of $(a_1, ..., a_n)$. Assuming that $a_1 \leq a_2 \leq ... \leq a_n$ holds, such bounds include the classical bound by Erdős and Graham [3]

$$g(a_1, ..., a_n) \leq 2a_{n-1}\lfloor\frac{a_n}{n}\rfloor - a_n,$$

by Selmer [8]

$$g(a_1, ..., a_n) \leq 2a_n\lfloor\frac{a_1}{n}\rfloor - a_1,$$

by Vitek [11]

$$g(a_1, ..., a_n) \leq \frac{1}{2}(a_2 - 1)(a_n - 2) - 1,$$

and by Beck et al. [1]

$$g(a_1, ..., a_n) \leq \frac{1}{2}\big(\sqrt{a_1a_2a_3(a_1 + a_2 + a_3)} - a_1 - a_2 - a_3\big).$$

In Section 2, we present a new algorithm to compute the Frobenius number. Also we present the sequential form of the new algorithm in Section 3.

2 Introducing a new algorithm

In this section, we present a new algorithm to calculate the Frobenius number and also we discuss on the time complexity of this new algorithm.

2.1 New algorithm

We start this section with the following easy theorem:

Theorem 2.1. *For the numbers $\alpha_1 < \alpha_2 < ... < \alpha_n$, we have*

$$g(\alpha_1, ..., \alpha_n) \leq g(\alpha_1, ..., \alpha_{n-1}) \leq ... \leq g(\alpha_1, \alpha_2).$$

Proof. By the definition of Frobenius number, all integers strictly greater than $g(\alpha_1, \alpha_2) = \alpha_1\alpha_2 - \alpha_1 - \alpha_2$ can be expressed as $\alpha_i x_i + \alpha_j x_j$ for some $x_i, x_j \in \mathbb{Z}^+$. So it follows that all integers strictly greater than $\alpha_1\alpha_2 - \alpha_1 - \alpha_2$ can be expressed as $\sum_{k=1}^{n} \alpha_k x^k$ for non-negative integer x_k where $k \in \{1, 2, ..., n\}$. Therefore, we have the result. □

Using the upper bound of Theorem 2.1, we present a new algorithm for calculating Frobenius numbers. More precisely, since $g(\alpha_1, ..., \alpha_n) \leq g(\alpha_1, \alpha_2) = \alpha_1\alpha_2 - \alpha_1 - \alpha_2$, we compute the number $\alpha_1\alpha_2 - \alpha_1 - \alpha_2$ and using our sub-algorithm which we call it *HasRep Algorithm* examine the natural numbers less than $g(\alpha_1, \alpha_2)$ are representable respect to $\alpha_1, ..., \alpha_n$ or not. Obviously, the largest number less than $\alpha_1\alpha_2 - \alpha_1 - \alpha_2$ which dose not have a representation, is the Frobenius number of $\alpha_1, ..., \alpha_n$. After this, by the Algorithm 2.1 which

Algorithm 2.1: Frob Algorithm

input: L, the list of numbers.
output: The Frobenius number of L
$F := L[1] * L[2] - (L[1] + L[2])$;
Flag:=true;
for a **from** F **by** -1 **to** $L[1] + 1$ **while** flag **do**
$Flag := HasRep(a, L)$;
end for
return(a+1);
End.

we call it *Frob Algorithm*, we apply the Algorithm 2.2 to compute the Frobenius number of $\alpha_1, ..., \alpha_n$.
We have implemented the new algorithm in the Maple software. We briefly show the results of the implementation of this algorithm for a few examples in the Table 1. It should be note that the designed algorithm, unlike some algorithms, is responsible for any number of numbers.

Algorithm 2.2: HasRep

input: a, L: a number and L list of numbers.
output: ture, if a has a representation with respect to L and false, otherwise.
if $\#L = 2$ **then**
flag:= false;
if $a \ mod \ L[2] = 0$ **then**
return(true) ;
elif $a = L[1] * L[2] - L[1] - L[2]$ **then**
return(false);
fi;
while $a \geq L[2]$ **and not flag do**
if $a \ mode \ L[1] \neq 0$ **then**
$a := a–L[2]$;
else
return(true);
fi;
end while;
return(false);
else
if $a \ mode \ L[-1] = 0$ or $HasRep(a, L[1..-2])$ **then**
return(true);
else
flag:= false;
while $a \geq L[-1]$ **and not flag do**
$a := a - L[-1]$;
$flag := HasRep(a, L[1..-2])$;
end while;
fi;
return (flag) ;
fi;
End.

2.2 Complexity

In this subsection, we discuss on the time complexity of the HasRep algorithm.
The time complexity of HasRep algorithm is difficult to determine precisely without more information about the list L and the potential values of a. In the worst case, the outer while loop in the base case can iterate up to $a/L[2]$ times. The outer while loop in the

Numbers	Frobenius Number
7,11,13	30
53,71,91	899
322, 654, 765	27971
123,1234,12345	71459
151, 157, 251, 711	3019
151, 157, 251, 711, 912	3019
101,109,113,119,121,131,139,149,151,161,163,167,169,187,191, 214,219,238,276,324,345,346,349,387,421,427,444,453,463,525, 530,555,579,580,625,711,719,737,752,787,814,834,856,878,899, 915,937,978,989	426

Table 1: The results of the implementation of proposed algorithm for a few examples

recursive case can iterate up to $a/L[-1]$ times. Each iteration of the outer while loop involves a recursive call to HasRep, potentially leading to further recursive calls. This suggests that the time complexity could be exponential in the worst case. To improve the efficiency, especially for the cases where the same subproblems are encountered repeatedly, memorization (caching previously computed results) could be implemented.
Now, let analyze the time complexity of the HasRep code.
First we consider the worst-case time complexity analysis. For the base case ($\#L = 2$), the while loop can iterate at most $a/L[2]$ times. Each iteration involves constant-time operations (modulo, subtraction, comparisons). Therefore, the time complexity of the base case is $O(a/L[2])$.
For the recursive case ($\#L > 2$), the recursive call to $HasRep(a, L[1..-2])$ can potentially occur in every iteration of the while loop. The while loop can iterate at most $a/L[-1]$ times. This leads to a potential exponential number of recursive calls
In general, in the HasRep algorithm, in the first part (when the condition is equal to 2), the complexity of the algorithm will be $O(n/L[2])$ and when this condition is not met, it will be $O(2^n)$. The complexity of the Frob algorithm is n times of complexity of the HasRep algorithm.

3 The sequential form of algorithm

It is interesting that the proposed algorithm can convert to the sequential form, that we do it in this section. Since the HasRep Algorithm gives YES or NO, we use a function to give us 0 and 1. We define the function f as follows:

$$f(\alpha_1, R) := \lfloor \frac{\alpha_1}{R} - \frac{1}{\lfloor \frac{R}{\alpha_1} \rfloor} \rfloor.$$

Note that if R is divisible by α_1, then the result of the function $f(\alpha_1, R)$ is zero. Otherwise, the result of the expression is between -1 and 0. Now we define the function H as follows:

$$H(R,[\alpha_1,\alpha_2]) := \lfloor \frac{\alpha_1}{R} - \frac{1}{\lfloor \frac{R}{\alpha_1} \rfloor} \rfloor \lfloor \frac{\alpha_2}{R} - \frac{2}{\lfloor \frac{R}{\alpha_2} \rfloor} \rfloor \lfloor \frac{\alpha_1}{R-\alpha_2} - \frac{1}{\lfloor \frac{R-\alpha_2}{\alpha_1} \rfloor} \rfloor \lfloor \frac{\alpha_1}{R-2\alpha_2} - \frac{1}{\lfloor \frac{R-2\alpha_2}{\alpha_1} \rfloor} \rfloor ...$$

$$\lfloor \frac{\alpha_1}{R-(\lfloor \frac{R}{\alpha_2} \rfloor -1)\alpha_2} - \frac{1}{\lfloor \frac{R-(\lfloor \frac{R}{\alpha_2} \rfloor -1)\alpha_2}{\alpha_1} \rfloor} \rfloor \lfloor \lfloor \frac{R-\lfloor \frac{R}{\alpha_2} \rfloor \alpha_2}{\alpha_1} \rfloor \alpha_1 - R + \lfloor \frac{R}{\alpha_2} \rfloor \alpha_2 \rfloor$$

The function H has two variables (two input) which investigate the linear representation of R with respect to a list. It is obvious that if R has a linear representation with respect to L, then the value of the function H is 0, otherwise is a number in $(-1, 0)$ or in $(0, 1)$. Now we define another function which we denote it by N (it inverts the answer of the H function) as follows:

$$N(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } \{x| -1 \leq x \leq 1; x \neq 0\} \end{cases}$$

Note that

$$N(x) := \lfloor -|x| \rfloor + 1.$$

Based the Frob Algorithm, we first find the upper bound U and then enter the numbers from 1 to U into the function $N(H(U,[\alpha_1,\alpha_2]))$. The numbers that remain have no representation with respect to the list. But the Frobenius number is the largest among them, so we can calculate it. If $\delta_i := N(H(i,[\alpha_1,\alpha_2]))$, then the following theorem gives the Frobenius number $g(\alpha_1, ..., \alpha_n)$:

Theorem 3.1. *The Frobenius number $g(\alpha_1, ..., \alpha_n)$ based on the values of δ_i and function $N(\delta_i)$ is equal to*

$$g(\alpha_1, ..., \alpha_n) := (U)(\delta_U) + (U-1) \times (\delta_{U-1})N(\delta_U) + \cdots + (1)(\delta_1)N(\delta_U)...N(\delta_2)$$

We close the paper by the following remark:

Remark 3.2. *The function H can be generalized for n numbers, i.e., if $\delta_i := N(H(i,[\alpha_1, ..., \alpha_n]))$,, then*

$$H(R,[\alpha_1,\alpha_2]) \times H(R,[\alpha_1,\alpha_2,\alpha_3]) \times ... \times H(R,[\alpha_1,...,\alpha_{n-1}]) \times \lfloor \frac{\alpha_n}{R} - \frac{1}{\lfloor \frac{R}{\alpha_n} \rfloor} \rfloor \times$$

$$\prod_{i=1}^{\lfloor \frac{R}{\alpha_n} \rfloor} H(R - i\alpha_n, [\alpha_1, ..., \alpha_{n-1}])$$

4 Conclusions

Exact determination of the Frobenius number is a difficult problem in general. There are only a few cases where the Frobenius number has been exactly determined for any n variables. In the absence of exact results, research on the Frobenius problem has often been focused on sharpening bounds on the Frobenius number and on algorithmic aspects. In this paper, using the upper bound in Theorem 2.1, we presented a new algorithm for calculating Frobenius numbers. Also we present the sequential form of the new algorithm. The idea of this algorithm proposed by the first author in 2017 when he was a ninth grade high school student and presented in the Khwarazmi Youth Festival Competitions.

References

[1] M. Beck, R. Diaz, S. Robins, The Frobenius problem, rational polytopes, and fourier dedekind sums, Journal of number theory, 96(1) 1–21, 2002.

[2] F. Curtis, On formulas for the Frobenius number of a numerical semigroup, Mathematica Scandinavica, 67(2) 190–192, 1990.

[3] P. Erdős, R. Graham, On a linear diophantine problem of frobenius, Acta Arithmetica, 21(1) 399–408, 1972.

[4] L. Fukshansky , A. Schürmann, Bounds on generalized Frobenius numbers, Eur. J. Combin, 32 (3) 361–368, 2011.

[5] J.B. Roberts, Note on linear forms, Proc. Amer. Math. Soc. 7, 465-469, 1956.

[6] W.J. Curran Sharp, Solution to problem 7382 (Mathematics). Educational Time, 41, 1884.

[7] J.L. J.L. Ramırez Alfonsın, The Diophantine Frobenius Problem, Oxford university press, 2005.

[8] E.S. Selmer, On the linear diophantine problem of frobenius. Journal für die reine und angewandte Mathematik, (293-294) 1–17, 1977.

[9] J.J. Sylvester, Problem 7382, Mathematical Questions from the Educational Times, 41, 21, 1884.

[10] A. Tripathi, Formulae for the Frobenius number in three variables, J. Number Theory 170, 368-389, 2017.

[11] Y. Vitek, Bounds for a linear diophantine problem of frobenius, Journal of the London Mathematical Society, 2(1), 79-85, 1975.