FBA SAMPLE

Abstract.

The two currently fastest general-purpose integer factorization algorithms are the Quadratic Sieve and the Number Field Sieve. Both techniques are used to find so-called smooth values of certain polynomials, i.e., values that factor completely over a set of small primes. As the names of the methods suggest, a sieving procedure is used for the task of quickly identifying smooth values among the candidates in a certain range. In the present paper, we present a novel approach based on representing such candidates as sums that are always divisible by several of the primes in the factor base. In a direct comparison with the Quadratic Sieve, the main advantage of our method is that the resulting values are smaller, which increases the likelihood of them being smooth. Using current implementations of the Self-initializing Quadratic Sieve in two Python packages for number-theoretical computations as benchmarks, a Python implementation of our approach runs faster in trials with inputs of 30 to 70 decimal digits. We discuss several promising avenues for further improvements and applications of the technique.

Introduction

Integer factorization is the task of computing the divisors of natural numbers. It is a problem with a long and fascinating history, and it is certainly among the most influential in algorithmic number theory. While there is a variety of algorithms significantly faster than the brute-force search for divisors, it is still an open problem to construct a technique that efficiently factors general numbers with hundreds to thousands of digits. The hardness of this problem is fundamental for the security of widely used cryptographical schemes, most prominently the RSA cryptosystem. Nevertheless, there is no proof for its hardness besides the fact that decades of efforts have failed to construct a more efficient technique. Quite regularly, there are set new records1 concerning the factorization of numbers of certain size, mostly due to improved implementations of the best available algorithms and advances in the hardware and computing power. In addition, the bound for the deterministic integer factorization problem has been improved multiple times in recent years ([11], [12], [10], [14]). On the other hand, there has only been little progress in the development of new techniques for practical integer factorization since the invention of the Number Field Sieve ([19]) in the 1990s. One of the earlier algorithms with sub-exponential runtime was by Dixon ([7]) in 1981.

From today’s perspective, it may be considered as a prototype for several other algorithms. To this group belong the Continued Fraction factorization method (CFRAC) described and implemented by Morrison and Brillhart ([24]) in 1975, the Linear Sieve by Schroeppel and the Quadratic Sieve by Pomerance. The latter author analyzed and compared these algorithms in 1982 in [25]. In general, there is an extensive amount of literature on practical integer factorization algorithms. The reader may find information on the mentioned methods and other factorization techniques in the survey [18] and in the monographs [27] and [30]. These sources also discuss a variety of factorization algorithms for numbers N that satisfy certain properties, or that have prime factors of certain shape. For example, Fermat’s factorization method runs particularly fast if N has co-divisors that are very close, while Pollard’s P-1 technique and the Elliptic Curve Method (ECM) work particularly well for numbers with small prime factors. In the present paper, we are interested in so-called general-purpose factorization algorithms, which means that the runtime complexity estimate of the procedure only depends on the size of the input number N. Both the Quadratic Sieve and the Number Field Sieve belong to this group. From a practical point of view, the Quadratic Sieve (and its modifications) is the best general-purpose factorization algorithm for N up to around 100 digits, while the Number Field Sieve is faster for inputs beyond that. While methods such as CFRAC and ECM stay somewhat competitive up to a small input length (see [23]), factorization tools usually use ECM to rule out the existence of relatively small prime factors, but then switch either to the Number Field Sieve or to the Self-initializing Quadratic Sieve (SIQS), which is the fastest modification of the original Quadratic Sieve algorithm.2 Our contribution is the presentation and evaluation of a novel approach that we call Smooth Subsum Search (SSS). Just like CFRAC, the Linear Sieve and the Quadratic Sieve, SSS belongs to the group of Dixon-type algorithms. In addition to a detailed explanation of our algorithm and a discussion of its advantages and disadvantages compared to other techniques, we will discuss our current implementation in Python and several experiments on the runtime complexity of SSS. Our results demonstrate that SSS outperforms SIQS on semiprime3 numbers with 30 to 70 decimal digits. While these findings need to be corroborated in experiments with different implementations of SIQS in other programming languages and in other settings, they may indicate that SSS is the fastest available factorization algorithm for general numbers in this range. The remainder of the paper is structured as follows: In Section 2, we discuss the Quadratic Sieve and its modifications in greater detail, together with an important subroutine of our own algorithm. Section 3 presents the main ideas of SSS. The results of our experiments can be found in Section 4, where SSS is compared to available implementations of the Quadratic Sieve in Python. Section 5 explores three ideas on how to improve the current implementation of SSS, in particular for numbers of larger size. Finally, we summarize and give concluding remarks in Section 6.

Related Work

Let N be the number we want to factorize. We will always assume that N is odd, composite and not a perfect power of another number. The currently fastest general-purpose factorization algorithm is the General Number Field Sieve with a heuristic asymptotic runtime complexity of

Integer factorization has been one of the most important foundations of modern information security [12]. The exponential speedup of integer factorization by Shor’s algorithm [13] is a great manifestation of the superiority of quantum computing. However, running Shor’s algorithm on a fault-tolerant quantum computer is quite resource intensive [14, 15]. Up to now, the largest integer factorized by Shor’s algorithm in current quantum systems is 21 [16– 18]. Alternatively, integer factorization can be transformed into an optimization problem, which can be solved by adiabatic quantum computation (AQC) [19–22] or QAOA [23]. Larger numbers have been factored using these approaches, in various physical systems [24–27]. The maximum integers factorized are 291311 (19-bit) in NMR system [26], 249919 (18- bit) in D-Wave quantum annealer [25], 1099551473989 (41- bit) in superconducting device [27]. However, it should be noted that some of the factored integers have been carefully selected with special structures [28], thus the largest integer factored by a general method in a real physical system by now is 249919 (18-bit). In this paper, we propose a universal quantum algorithm for integer factorization that requires only sublinear quantum resources. The algorithm is based on the classical Schnorr’s algorithm [29, 30], which uses lattice reduction to factor integers. We take advantage of QAOA to optimize the most time-consuming part of Schnorr’s algorithm to speed up the overall computing of the factorization progress. For an m-bit integer N, the number of qubits needed for our algorithm is O(m/logm), which is sublinear in the bit length of N. This makes it the most qubit-saving quantum algorithm for integer factorization compared with the existing algorithms, including Shor’s algorithm. Using this algorithm, we have successfully factorized the integers 1961 (11-bit), 48567227 (26-bit) and 261980999226229 (48-bit), with 3, 5 and 10 qubits in a superconducting quantum processor, respectively. The 48-bit integer, 261980999226229, also refreshes the largest integer factored by a general method in a real quantum device. We proceed by estimating the quantum resources required to factor RSA-2048. We find that a quantum circuit with 372 physical qubits and a depth of thousands is necessary to challenge RSA-2048 even in the simplest 1D-chain system. Such a scale of quantum resources is most likely to be achieved on NISQ devices in the near future.

ABSTRACT

One of the most significant challenges on cryptography today is the problem of factoring large integers since there are no algorithms that can factor in polynomial time, and factoring large numbers more than some limits (200 digits) remain difficult. The security of the current cryptosystems depends on the hardness of factoring large public keys. In this work, we want to implement two existing factoring algorithms - pollard-rho and quadratic sieve - and compare their performance. In addition, we want to analyze how close is the theoretical time complexity of both algorithms compared to their actual time complexity and how bit length of numbers can affect quadratic sieve’s performance. Finally, we verify whether the quadratic sieve would do better than pollard-rho for factoring numbers smaller than 80 bits

Abstract

Let n = p × q (p < q) and ∆ = |p − q|, where p, q are odd integers, then, it is hypothesized that factorizing this composite n will take O(1) time once the steady state value is reached for any ∆ in zone0 of some observation deck (od) with specific dial settings. We also introduce a new factorization approach by looking for ∆ in different ∆ sieve zones. Once ∆ is found and n is already given, one can easily find the factors of this composite n from any one of the following quadratic equations: p 2 + p∆− n = 0 or q 2 − q∆− n = 0. The new factorization approach does not rely on congruence of squares or any special properties of n, p or q and is only based on sieving the ∆. In addition, some other new factorization approaches are also discussed. Finally, a new trapdoor function is presented which is leveraged to encrypt and decrypt a message with different keys.

Introduction If n = p × q where p, q are some large primes, then no non-quantum algorithm exists today that can either find p or q in polynomial time when only n is given. This problem of factorizing the product of two large primes has kept great minds busy for quite some time, especially since 1977, when Rivest-Shamir-Adleman [1] leveraged this problem in building a public-key cryptosystem, famously known as RSA, that is widely used for secure data transmission. However, in 1994, Peter Shor [2] published an algorithm that can solve the integer factorization problem in polynomial time by using a universal quantum computer. When in future, a large ideal universal quantum computer with enough qubits is able to operate efficiently, then RSA scheme will no longer remain secure. Two key ideas have remained centerstage when attempting to solve the factorization problem. Variety of different methods and approaches have been developed around these two central ideas

Abstract.

We present an algorithm to compute all factorizations into linear factors of univariate polynomials over the split quaternions, provided such a factorization exists. Failure of the algorithm is equivalent to non-factorizability for which we present also geometric interpretations in terms of rulings on the quadric of non-invertible split quaternions. However, suitable real polynomial multiples of split quaternion polynomials can still be factorized and we describe how to find these real polynomials. Split quaternion polynomials describe rational motions in the hyperbolic plane. Factorization with linear factors corresponds to the decomposition of the rational motion into hyperbolic rotations. Since multiplication with a real polynomial does not change the motion, this decomposition is always possible. Some of our ideas can be transferred to the factorization theory of motion polynomials. These are polynomials over the dual quaternions with real norm polynomial and they describe rational motions in Euclidean kinematics. We transfer techniques developed for split quaternions to compute new factorizations of certain dual quaternion polynomials.

Introduction.

For a given binary matrix X ∈ {0, 1} n×m and a fixed positive integer k, the rank-k binary matrix factorisation problem (k-BMF) is concerned with finding two matrices A ∈ {0, 1} n×k , B ∈ {0, 1} k×m such that the product of A and B is a binary matrix closest to X in the squared Frobenius norm. One can define different variants of this problem depending on the underlying arithmetic used when computing the product of the matrices. In this paper we focus on solving k-BMF under Boolean arithmetic where the product of the binary matrices A and B is computed by (i) interpreting 0s as false and 1s as true, and (ii) using logical disjunction (∨) in place of addition and logical conjunction (∧) in place of multiplication. Observe that Boolean multiplication (∧) coincides with standard multiplication on binary input, hence we adopt the notation a b in place of a ∧ b in the rest of the paper. We therefore compute the Boolean matrix product of A and B as: Z = A ◦ B ⇐⇒ zij = \_ ` (ai` b`j ). Note that Boolean matrix multiplication can be equivalently written as zij = min{1, P ` ai`b`j} using standard arithmetic summation. The problem then becomes computing matrices A and B whose Boolean product Z best approximates the input matrix X. Our motivation for this study comes from data science applications where rows of the matrix X correspond to data points and columns correspond to features. In these applications low-rank matrix approximation is an essential tool for dimensionality reduction which helps understand the data better by exposing hidden features. Many practical datasets contain categorical features which can be represented by a binary data matrix using unary encoding. For example, consider a data