SCIENTIFIC REPORTS
natureresearch

OPEN

On speeding up factoring with quantum SAT solvers

Michele Mosca[1,2,3,4], Joao Marcos Vensi Basso[5]✉ & Sebastian R. Verschoor[1,6]

There have been several efforts to apply quantum SAT solving methods to factor large integers. While these methods may provide insight into quantum SAT solving, to date they have not led to a convincing path to integer factorization that is competitive with the best known classical method, the Number Field Sieve. Many of the techniques tried involved directly encoding multiplication to SAT or an equivalent NP-hard problem and looking for satisfying assignments of the variables representing the prime factors. The main challenge in these cases is that, to compete with the Number Field Sieve, the quantum SAT solver would need to be superpolynomially faster than classical SAT solvers. In this paper the use of SAT solvers is restricted to a smaller task related to factoring: finding smooth numbers, which is an essential step of the Number Field Sieve. We present a SAT circuit that can be given to quantum SAT solvers such as annealers in order to perform this step of factoring. If quantum SAT solvers achieve any asymptotic speedup over classical brute-force search for smooth numbers, then our factoring algorithm is faster than the classical NFS.

Factoring integers by translating the problem directly into a satisfiability (SAT) instance or any equivalent NP-hard problem does not appear to be efficient, even when quantum solvers are assumed to be able to achieve a quadratic speedup[1]. More importantly, the strategy does not even appear to perform better than the best known classical method: the Number Field Sieve (NFS)[2].

A subroutine of the NFS is to search for y-smooth numbers of a particular form, where an integer is y-smooth if all of its prime factors are $\leq y$. Using Grover's algorithm[3], this search can be done faster, so that a speedup over the classical method is achieved[4]. Although the resulting algorithm runs in super-polynomial time (and is thus slower than Shor's algorithm[5]), it requires asymptotically fewer logical qubits to implement.

We investigate the strategy of replacing Grover's search in the described low-resource algorithm by translating the smooth detection process into a satisfiability instance to be evaluated by a SAT solver. While the low-resource algorithm of Bernstein, Biasse and Mosca[4] requires a fault-tolerant quantum computer, one can alternatively attempt to solve these SAT instances with any quantum SAT solving algorithm or heuristic, such as a quantum annealer. *If* the quantum SAT solving heuristic achieves a speed-up over classical circuit-SAT solving algorithms, then we show that this leads to a factoring algorithm that is asymptotically faster than the regular NFS. While there is no convincing evidence to date that non-fault-tolerant quantum SAT solvers will provide an asymptotic speed-up over classical SAT solvers, with this approach we at least avoid the situation where the quantum SAT solver must outperform classical SAT solvers by a superpolynomial factor in order to compete with the NFS.

We first demonstrate theoretically that some approaches to translating smoothness testing to a SAT instance are too expensive. In practice, one might hope that SAT solvers would be able to pick up on patterns of these specific circuits and to achieve potential speedups. Note that this does not appear to happen in the direct factoring strategy[1], but there is the possibility that it would for the more specific problem of smooth number detection. After implementing one of the circuits, however, the benchmarks suggest that this is not the case.

We found one circuit implementing the Elliptic Curve Method (ECM)[6] which, if used as a subroutine of the NFS, could result in a speedup for factoring integers since SAT solvers can use this circuit to find smooth numbers asymptotically as fast as brute-force search. Moreover, if quantum annealers or other SAT solvers achieve any speedup over such classical SAT solving, then our algorithm is faster than the classical one. In the optimistic case that quantum SAT solvers achieve a full quadratic speedup, then our algorithm has the same time complexity as the low-resource quantum algorithm of Bernstein, Biasse and Mosca[4].

[1]Institute for Quantum Computing, University of Waterloo, Waterloo, Canada. [2]Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada. [3]Perimeter Institute for Theoretical Physics, Waterloo, Canada. [4]evolutionQ Inc., Kitchener, Canada. [5]Department of Physics and Astronomy, Tufts University, Medford, USA. [6]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. ✉email: Joao.Vensi_Basso@tufts.edu

Contributions of this paper. We show in general that a few approaches for smoothness detection with SAT circuits are not enough to speed up the NFS. Moreover, we run benchmarks and find that a classical SAT solver does not appear to pick up on any patterns that allow one to claim otherwise for these approaches. Most importantly, we present a circuit that, when used as a NFS subroutine, yields an algorithm with the same asymptotic runtime as the classical NFS, and faster if quantum SAT solvers achieve any non-trivial speedup. In the optimistic case that a quantum SAT solver achieves a full quadratic speedup, the algorithm would be as fast as the low-resource quantum algorithm, while not necessarily requiring a fault-tolerant quantum computer to operate.

Nomenclature. We refer to the algorithm of Buhler et al.[2] as the classical NFS, to the algorithm in Bernstein, Biasse and Mosca[4] as the low-resource quantum NFS, and to our algorithm that uses ECM for smoothness detection as circuit-NFS.

Organization. We start by reviewing previous work related to SAT solving as well as factoring, including a factoring algorithm that encodes a multiplication circuit as a SAT instance whose solution represents the prime factors. We also recall work done to speed up the Number Field Sieve using Grover's search on a quantum computer with $(\log N)^{2/3+o(1)}$ logical qubits, where N is the number being factored. Then, we present a few encodings of smoothness detection into circuits. We show in general that the SAT instances belonging to smoothness-detection circuits which have the prime exponents or the factors of the number being tested for smoothness as variables cannot be solved fast enough to speed up factoring. Most importantly, we present a circuit implementing the ECM, analyze it and make statements about the solver runtime relative to the speedup obtained by a quantum SAT solver. Lastly, we discuss the results of this paper as well as future work.

Previous work

The work by Mosca and Verschoor[1] investigates the use of SAT solvers for factoring semi-primes, that is, numbers with only two primes factors of similar size. It encodes a multiplication circuit into a SAT instance, fixing the output as the number being factored and making the multiplicands variable. Therefore, solving such SAT instance is equivalent to factoring the semi-prime. The paper finds no evidence that this approach to factoring via classical SAT solvers provides any advantage, or even matches the classical NFS. It also points out that quantum SAT solvers are not expected to do much better if factoring is encoded as a SAT instance in this direct fashion.

Further work by Benjamin et al.[7] exhibits a quantum algorithm for solving 3-SAT, an instance of SAT where all clauses contain 3 literals. The numerical simulations for small systems presented in the paper indicate that the algorithm has performance comparable to that of one of the best known classical algorithms. This is of importance to our paper since any significant speedup for SAT solving implies a speedup for factoring as well.

The general number field sieve (NFS) improves on the special number field sieve[8] by removing any restrictions on the numbers that can be factored. The NFS algorithm is conjectured to factor any integer N in $L_N[1/3,(64/9)^{1/3}+o(1)]$ time, where $L_x[a,b]=\exp\left(b(\log x)^a(\log\log x)^{1-a}\right)$ and $o(1)\to 0$ as $N\to\infty$. Here we give a brief overview of the algorithm, highlighting the details relevant to the present paper. Note that the NFS is explained and analyzed in thorough detail by Buhler et al.[2]. For a simplified overview, see[4], Section 2, whose notation we follow.

The algorithm takes in an integer N to be factored and parameters d, y, u, with

- $y \in L^{\beta+o(1)}$
- $u \in L^{\epsilon+o(1)}$
- $d \in (\delta+o(1))(\log N)^{1/3}(\log\log N)^{-1/3}$

where $N>2^{d^2}$, $L=L_N[1/3,1]$ and β,δ,ϵ are parameters to be optimized for in the analysis. Further, define

- $m := \lfloor N^{1/d}\rfloor$
- $U := \{(a,b)\in\mathbb{Z}^2 : \gcd\{a,b\}=1, |a|\le u, 0<b\le u\}$
- $f(X) := \sum_{i=0}^{d} c_i X^i$ where the c_i are obtained by writing N in base m: $N=\sum_{i=0}^{d} c_i m^i$
- α such that $f(\alpha)=0$
- $g(a,b) := (-b)^d f(-a/b) = \sum_{i=0}^{d} c_i a^i (-b)^{d-i}$
- $F(a,b) := (a+bm)g(a,b)$
- $\phi : \mathbb{Z}[\alpha]\to\mathbb{Z}/N\mathbb{Z} : \sum_i a_i\alpha^i \to \sum_i a_i m^i$, a homomorphism.

From the above, one can see that d represents the degree of the polynomial f and that u is, in a sense, a bound on the search space U. Moreover, as explained below, y is taken to be the smoothness bound on $F(a, b)$. N is assumed to be odd.

The NFS attempts to find a suitable set $S\subseteq U$ such that on the rational side

$$\prod_{(a,b)\in S}(a+bm)=X^2 \text{ is a square in } \mathbb{Z} \tag{1}$$

and on the algebraic side

$$f'(\alpha)^2\prod_{(a,b)\in S}(a+b\alpha)=\beta^2 \text{ is a square in } \mathbb{Z}[\alpha]. \tag{2}$$

The algorithm then outputs

$$\gcd\{N, \phi(\beta) - f'(m)X\}. \tag{3}$$

In order to find an appropriate S, the algorithm looks for a $T \subseteq U$ such that $T = \{(a,b) \in \mathbb{Z}^2 : \gcd\{a,b\} = 1, |a| \leq u, 0 < b \leq u, F(a,b) \text{ is } y\text{-smooth}\}$, with $\#T \in y^{1+o(1)}$. After T is found, a linear dependence relation between the exponent vectors (reduced modulo 2) of $F(a,b)$ for $(a,b) \in T$ reveals a suitable set $S \subseteq T$ such that both Eq. (1) and Eq. (2) are satisfied.

The two main bottlenecks of NFS are to (1) find T and (2) find the linear dependence relation. In the classical NFS (1) takes $L^{2\epsilon+o(1)}$ time, since that is the size of U, and (2) takes $L^{2\beta+o(1)}$ with Wiedemann's algorithm[9]. By balancing both, one obtains a total runtime of $L^{1.923}$. The low-resource algorithm does (1) using Grover's search and yields a better runtime, namely $L^{1.387}$. Note as well that, if (2) is assumed to take $L^{2.5\beta+o(1)}$, as considered by Bernstein[10], the classical NFS ends up with runtime $L^{1.976}$ and the low-resource algorithm with $L^{1.456}$. For completeness, we repeat the derivations in Corollary 5 and Corollary 6.

Circuits for smoothness detection

The circuit SAT problem asks whether there exists an input for a given Boolean circuit, encoded as a SAT instance, such that the output will be TRUE. For a satisfiable circuit SAT formula in v variables one can easily find a solution with v queries to a decision oracle for SAT. In practice, the best known algorithms for deciding SAT implicitly also provide a solution and thus the repeated applications of a SAT decision algorithm are not necessary. Using binary encoding for integers we construct circuits that encode a predicate on numbers, so that solving the corresponding SAT instance is a search for numbers satisfying the predicate. From here on we refer to this process as "solving the circuit".

Instead of using Grover's search to look for $(a,b) \in T$ as in the low-resource quantum NFS[4], we let a SAT solver find these using the encoded circuit. In particular, we encode the predicate "$F(a,b)$ is a y-smooth number" on the input pair (a,b), while we assume the conditions $|a| \leq u$ and $0 < b \leq u$ are enforced by the input encoding. Similar to the low-resource quantum NFS, we assume that the case $\gcd\{a,b\} > 1$ is handled by post-processing.

A naive algorithm for circuit SAT simply evaluates the full circuit for every possible input until a one is found at the output. For a circuit with v input variables and size g this strategy has runtime $O(2^v g)$, which is the runtime we assume for solving circuits. Given that circuit SAT is an NP-complete problem, it is widely believed that no efficient algorithm exists. However, in practice modern SAT solvers perform well on solving large SAT instances for certain problems, so that the conjectured runtime requires some confirmation in the form of benchmark results.

In this section we analyze a few natural circuits for implementing the required predicate and prove the approach does not offer any improvement over the classical NFS. We show that, in general, circuits encoding all primes $p_i \leq y$ or the prime exponents e_i can not be solved efficient enough. On the other hand, solving a circuit implementing the Elliptic Curve Method (ECM)[6] is shown to achieve runtimes comparable to that of the classical NFS. We recall a few results important for the analysis.

Lemma 1

1. $|(a+bm)| \leq 2uN^{1/d}$ *and* $|g(a,b)| \leq (d+1)N^{1/d}u^d$
2. $\log|F(a,b)| \in O((\log N)^{2/3}(\log\log N)^{1/3})$
3. $\log\log|F(a,b)| \in O(\log\log N)$

Proof

1. Follows directly from the definitions of g and m.
2. $\log|F(a,b)| \leq \log 2(d+1) + \frac{2\log N}{d} + (d+1)\log u \in O(\frac{\log N}{d} + d\log u)$. Now

$$\begin{aligned}\frac{\log N}{d} + d\log u =&(\epsilon\delta + \delta^{-1} + o(1))(\log N)^{2/3}(\log\log N)^{1/3}\\ &\subseteq O((\log N)^{2/3}(\log\log N)^{1/3}).\end{aligned} \tag{4}$$

3. Taking logs of the expression above, the dominant term is $\log\log N$.

□

Lemma 2 *If $F(a,b)$ is y-smooth, then $\omega(F(a,b)) \in O((\log N)^{2/3}(\log\log N)^{1/3})$, where $\omega(n)$ is the number of prime divisors of n without multiplicity.*

Proof All the prime factors of $F(a,b)$ are at least 2, so $\Omega(F(a,b)) \leq \log_2 F(a,b)$, where $\Omega(n)$ is the number of prime divisors of n with multiplicity. Since $\omega(n) \leq \Omega(n)$, the result follows from Lemma 1. □

Circuit with variable exponents. A natural idea is to hard-code all the primes $p_i \leq y$ into the circuit (see Fig. 1), and let a, b and e_i be the variables, where $1 \leq i \leq \pi(y)$, and $\pi(x)$ counts the number of primes $\leq x$. A satisfying assignment finds the exponent e_i for each prime p_i that forms the factorization of $F(a,b)$:

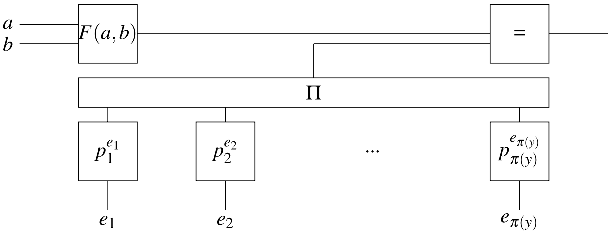

Figure 1. Circuit directly encoding Eq. (5). Variables are shown in boldface. The Π gate outputs the product of all input values.

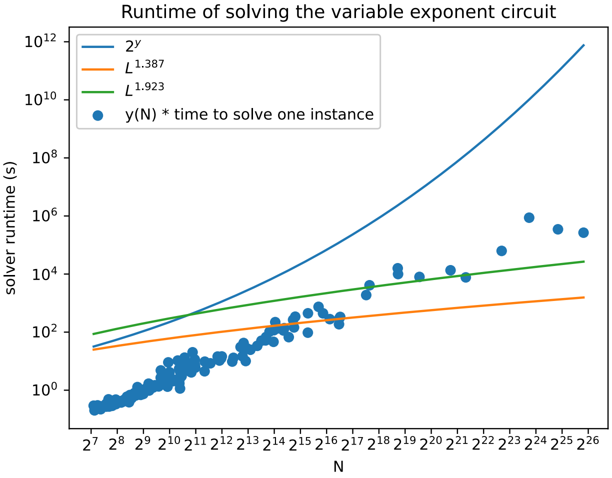

Figure 2. Scaling of solving times for the variable exponent circuit.

$$F(a,b) = \prod_{i=1}^{\pi(y)} p_i^{e_i} \tag{5}$$

The circuit provides no improvement over the classical NFS. Indeed, the number of bits necessary to represent $\vec{e} = (e_1, e_2, ..., e_{\pi(y)})$ is lower-bounded by $\pi(y) \in y^{1+o(1)}$, which implies that the time to solve the circuit is at least exponential in $L^{\beta+o(1)}$, much larger than the overall NFS complexity. This also proves the following.

Proposition 1 *Any circuit that has $\vec{e}$ as variable input to be found by an exponential-time SAT solver is not sufficient to speed up integer factorization.*

Despite the theoretical result above, one might hope that SAT solvers are able to pick up on specific patterns of this circuit and exploit them to improve the overall runtime. In order to investigate this possibility, we encoded this circuit into a satisfiability instance and ran benchmarks using MapleCOMSPS[11].

A circuit is generated for each number N, with all other parameters generated as described before and by setting $o(1) = 0$. In order to keep the circuit from growing too large, intermediate values in the computation of $\prod p_i^{e_i}$ are truncated to $\log_2 F(u, u)$ bits and multiplication is computed by schoolbook multiplication. Despite these techniques the SAT instances can grow large: on the tested range they contain up to eighty thousand variables after simplification. This is partially explained by the fact that F (both the bound $F(u, u)$ and the found values $F(a, b)$) is much larger than N for these small values of N. With the used parameters the desired $F(u, u) < N$ will only occur for 140 bit values of N and greater. All code for generating circuits (including tests for correctness), benchmarks and measurements is made available online[12].

Figure 2 shows the benchmarking results. For each $N \leq 2^{18}$ we measured the median time of solving the same instance many times, for larger N we report the solver runtime directly. Each measured runtime is multiplied by $y(N)$.

Since there are many (a, b) that satisfy the predicate, we could run the solver many times to find multiple $(a, b) \in T$. Closer inspection of our results indicate that the SAT solver does indeed find many valid pairs. If collisions are a problem, we could arbitrarily partition the search space by putting restrictions on the input and have multiple solvers work in parallel. Alternatively we could encode the negation of found solutions as a new

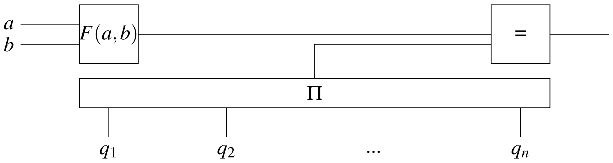

Figure 3. Circuit with variable factors. Variables are shown in boldface. The Π gate outputs the multiplication of all input values.

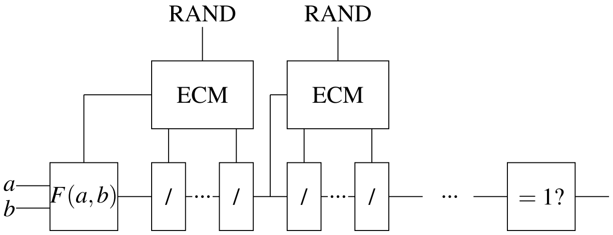

Figure 4. Circuit implementing the Elliptic Curve Method (ECM). Variables are shown in boldface. RAND stands for a source of randomness for the parameters of the ECM.

SAT clause. Determining which approach is best is left as an open question, but here we assume that finding $y(N)$ solutions takes $y(N)$ times the resources as finding one solution.

Given the asymptotic behaviour displayed in Fig. 2 it appears that the optimizations from the SAT solver do not seem large enough to provide a speed-up to the NFS. Although this is not a statement about quantum SAT solvers, it is one more argument supporting the lack of speedup attributable to the SAT solver learning specific structures of this problem.

Circuit with variable factors. Exploiting the small number of prime factors of $F(a, b)$ following from Lemma 2, one can hope to turn the factors into variables (see Fig. 3). At the end, the factors q_i must multiply to $F(a, b)$. Note that the q_i need not be prime, but only $\leq y$. This restriction could be enforced at no cost by allowing at most $\lceil \log_2 y \rceil$ bits to encode each q_i or by an efficient test on each input.

However, this strategy is too costly. That is, the number of variables in the circuit is $2 \log u + \sum_i \log q_i > \log \prod_i q_i$. In the very best case that the q_i are encoded with the exact number of necessary bits, which is $\log F(a, b)$, then by Lemma 1, results in $L_N[2/3, \cdot]$ time to solve the circuit. This also implies the following.

Proposition 2 *If $\prod_i q_i = F(a, b)$, any circuit that has the q_i as variables to be found by an exponential-time SAT solver is not sufficient to speed up integer factorization.*

ECM circuit. The Elliptic Curve Method (ECM) is a factoring algorithm devised by Lenstra[6]. One of its key features is that its runtime is conjectured to depend on the smallest prime factor of the number being factored, making it very suitable for smoothness detection. We create a circuit that executes repeated runs of the ECM to obtain prime factors $p_i \leq y$ of $F(a, b)$. For each prime obtained, repeated divisions are performed in order to eliminate that prime from the factorization. Figure 4 shows a simplified circuit. There are implicit operations such as checking if the obtained prime is $\leq y$ and only performing division when the remainder is zero. RAND represents a random choice of parameters for the ECM, more specifically a, x, y, using the notation by Lenstra[6], section (2.5). Note that, for a given SAT instance, the random generator seeds are fixed.

This circuit meets the desirable time complexity by decreasing the number of variables significantly. Indeed, the only variables are a, b, so the search space is just U. The following theorem establishes the size and probability of success of the ECM circuit.

Theorem 3 *The ECM circuit can be designed to have size upper-bounded by $L_N[1/6, \sqrt{2\beta/3} + o(1)]$ and probability of success $1 - o(1)$.*

Proof From Lenstra's work[6], (2.10), one run of the ECM, with appropriate choice of parameters, finds with probability at least $1 - e^{-1}$ a non-trivial divisor of n in time $K(p)M(n)$, where p is the least prime divisor of n, $K(p) \in L_p[1/2, \sqrt{2} + o(1)]$ and $M(n) \in (\log n)^{1+o(1)}$. It is uncertain that the found non-trivial divisor is the smallest prime dividing n, but in practical circumstances this will often be the case[6], (2.10). For our purposes the divisors are allowed to be any factor of $F(a, b)$, as long as it is $\leq y$.

By Lemma 2, one can choose a constant c so that $\omega(F(a, b)) \leq c(\log N)^{2/3}(\log \log N)^{1/3}$. However, we increase $c \to c + \Delta$, $\Delta > 0$, to allow for ECM runs to fail. If there are $B := (c + \Delta)(\log N)^{2/3}(\log \log N)^{1/3}$

ECM blocks, the probability of success is the probability of at least $\omega(F(a,b))$ events out of B succeeding. This can be seen as a binomial process with probability of success $p = 1 - \frac{1}{e}$. In the limit $N \to \infty \implies B \to \infty$, $\text{Binomial}(x; B, p) \to \text{Normal}(x; Bp, Bp(1-p))$. We seek

$$\begin{aligned}\Pr(x \geq \omega(F(a,b))) &= \frac{1}{\sqrt{2\pi Bp(1-p)}} \int_{\omega(F(a,b))}^{\infty} \exp\left[-\frac{(x-Bp)^2}{2Bp(1-p)}\right] dx \\ &= \frac{1}{2}\left[1 - \text{erf}\left(\frac{(\log N)^{2/3}(\log\log N)^{1/3}(c - pc - p\Delta)}{\sqrt{2(c+\Delta)(\log N)^{2/3}(\log\log N)^{1/3} p(1-p)}}\right)\right]\end{aligned} \tag{6}$$

Note that if we let $\Delta \in O(1)$, that is, $\frac{\partial \Delta}{\partial N} = 0$, the circuit would not work, since $\lim_{N\to\infty} \Pr(x \geq \omega(F(a,b))) = 0$. However, if we let $\Delta = \Delta(N)$ such that $\lim_{N\to\infty} \Delta(N) = \infty$, then $\lim_{N\to\infty} \Pr(x \geq \omega(F(a,b))) = 1$, as desired. Hence choosing $\Delta \in \Theta(\log\log N)$ suffices and does not alter the final complexity.

Hence, let the circuit repeat the ECM step $O((\log N)^{2/3}(\log\log N)^{4/3})$ times and perform $O((\log N)^{2/3}(\log\log N)^{1/3})$ conditional divisions of an obtained prime, since this is the maximum power a prime factor can have in the factorization of $F(a,b)$, by Lemma 1. Each ECM has a different run-time since the least prime p changes and n is subsequently divided by the discovered factors. For upper-bound estimations, however, one can fix $p = y$ and $n = N$. In order to estimate the size of the ECM block, one can multiply the time and space complexity. The former is $K(y)M(N)$ and the latter is estimated to be $O(\log N)$. This yields a total circuit size of $O((\log N)^{2/3}(\log\log N)^{1/3})O((\log N)^{2/3}(\log\log N)^{4/3})K(y)M(N)O(\log N) \subseteq L_N[1/6, \sqrt{2\beta/3}+ o(1)]$
□

In order to analyze the runtime of solving the ECM circuit to find smooth $F(a,b)$, we need the following.

Definition 1 If a search space E has size $\#E$, an algorithm that is able to search through E within time $O(\#E^{1/\gamma})$ is said to achieve a γ-speedup.

For instance, Grover's search achieves a 2-speedup. The following establishes a generalization of the runtime analysis by Bernstein, Biasse and Mosca[4].

Theorem 4 *If an algorithm A achieves a γ-speedup, for $\gamma > 0$, and the linear algebra step in the NFS is assumed to take $L^{2\beta+o(1)}$, the NFS can use A to run in time $L^{\sqrt[3]{\frac{32(\gamma+1)}{9\gamma^2}}+o(1)}$.*

Proof By Lemma 1, $|F(a,b)| \leq 2(d+1)N^{2/d}u^{d+1}$. As shown in[4], section 3, a uniform random integer in $\left[1, 2(d+1)N^{2/d}u^{d+1}\right]$ has a smoothness probability of $L^{-(2/\delta+\delta\epsilon+o(1))/(3\beta)}$. We use the same heuristic and assume that this is also the smoothness probability of $F(a,b)$. Since there need to be $L^{\beta+o(1)}$ smooth $F(a,b)$ in the search space U of size $\#U \in L^{2\epsilon+o(1)}$, we must have $2\epsilon \geq \beta + (2/\delta + \delta\epsilon)/(3\beta)$. Since the constants are positive, $\epsilon\left(2 - \frac{\delta}{3\beta}\right) \geq \beta + \frac{2}{3\beta\delta}$ and $6\beta/\delta > 1$. With this relation, the smoothness probability becomes $L^{\beta-2\epsilon+o(1)}$.

Now, as in the analysis of the low-resource algorithm[4], we partition U in any systematic fashion into $L^{\beta+o(1)}$ parts of size $L^{2\epsilon-\beta+o(1)}$, each containing $L^{o(1)}$ smooth $F(a,b)$ with very high probability. Algorithm A can search each part in $L^{(2\epsilon-\beta)/\gamma+o(1)}$ time, for a total time of $L^{2\epsilon/\gamma+\beta(1-1/\gamma)+o(1)}$.

When balancing against the linear algebra step of $L^{2\beta+o(1)}$ time, we obtain $\epsilon = \beta\left(\frac{\gamma+1}{2}\right)$. Hence $\beta = \frac{(\gamma+1)\delta+\sqrt{\delta^2+96\gamma/((\gamma+1)^2\delta)}}{12\gamma}$, since $\frac{(\gamma+1)\delta-\sqrt{\delta^2+96\gamma/((\gamma+1)^2\delta)}}{12\gamma}$ is negative. By minimizing this as a function of δ, we obtain a minimum of $\beta = \sqrt[3]{\left(\frac{2}{3\gamma}\right)^2(\gamma+1)}$ given by $\delta = \sqrt[3]{\frac{12\gamma}{(\gamma+1)^2}}$. Note that $6\beta/\delta = 2(1+\frac{1}{\gamma}) > 1$. This yields a final NFS runtime of $L^{\sqrt[3]{\frac{32(\gamma+1)}{9\gamma^2}}+o(1)}$. □

The following two corollaries are restatements of the results for the low-resource algorithm[4].

Corollary 5 [4] *The classical NFS runs in $L^{\sqrt[3]{64/9}+o(1)}$ time, where $\sqrt[3]{64/9} \approx 1.923$.*

Proof Set $\gamma = 1$ in Theorem 4. □

Corollary 6 [4] *The low-resource quantum algorithm runs in $L^{\sqrt[3]{8/3}+o(1)}$ time, where $\sqrt[3]{8/3} \approx 1.387$.*

Proof Set $\gamma = 2$ in Theorem 4. □

The final runtime of circuit-NFS depends on the runtime of the SAT solver used. Figure 5 shows the exponent α in the final runtime $L^{\alpha+o(1)}$ of circuit-NFS achieved if the SAT solver used achieves a γ-speedup, that is, solves a circuit with ν variables in $2^{\nu/\gamma+o(1)}$ time.

The following results portray the two extreme scenarios highlighted in Fig. 5: a classical solver with $2^{\nu+o(1)}$ runtime versus an ideal quantum SAT solver that achieves a $2^{\nu/2+o(1)}$ runtime. The naive circuit SAT algorithm applied to the ECM circuit achieves runtime $O(2^{2\log_2 u} L_N[1/6, \cdot]) = L^{\sqrt[3]{64/9}+o(1)}$, corresponding to $\gamma = 1$. Note that we do not expect $\gamma > 2$ since $\gamma = 2$ has been proved optimal for a quantum computer[13] in a black-box context.

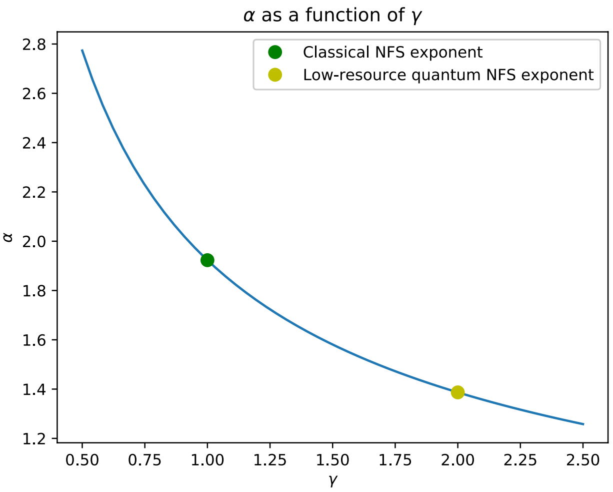

Figure 5. Exponent α of the final NFS runtime $L^{\alpha+o(1)}$ with the use of a SAT solver with γ-speedup. The relation between α and γ is given in Theorem 4.

Theorem 7 *With a classical circuit SAT solver, one can factor an integer N in* $L^{\sqrt[3]{64/9}+o(1)}$ *time, where* $\sqrt[3]{64/9} \approx 1.923$.

Theorem 8 *If a quantum SAT solver is assumed to achieve a full 2-speedup, it can be used to factor an integer N in* $L^{\sqrt[3]{8/3}+o(1)}$ *time, where* $\sqrt[3]{8/3} \approx 1.387$.

Theorem 7 is not an improvement on the classical NFS, but it shows that the circuit-NFS approach is asymptotically at least as good. Under the assumption that quantum annealears can achieve the aforementioned 2-speedup in solving SAT circuits, one can obtain the same asymptotic runtime as the low-resource quantum algorithm. However, this does not require a fault-tolerant quantum computer capable of running Grover's algorithm.

We emphasize that the speedup is computed over the naive *circuit SAT* algorithm. A standard translation of the circuit to CNF-SAT results in a SAT instance of size $L_N[1/6, \cdot]$ and a superpolynomial speedup over exponential-time CNF-SAT solvers would be required for speeding up factoring. Given the highly structured nature of the resulting SAT instance this might be feasible. Alternative solutions to avoid the superpolynomial overhead, such as direct translations from the ECM method to quantum annealer instances, are left as an open question for future work.

It is harder to make a statement about the qubit requirement of circuit-NFS. Instead of SAT, one can reduce to other NP-hard problems like QUBO for more direct application of DWave's quantum annealer. If the smoothness detection circuit could be simplified and written as an instance of QUBO in terms of the variables a, b only, that would total $2 \log u \in (\log N)^{1/3+o(1)}$ qubits. However, simplification is not trivial and does not seem to come without overhead, given our preliminary tests. It is more likely that intermediate wires of the circuit would also have to be QUBO variables, increasing the qubit requirement up to the full circuit size $L_N[1/6, \sqrt{2\beta/3} + o(1)]$. Therefore it remains an open question how many annealing qubits circuit-NFS requires. On the other hand, annealing qubits are currently produced in much higher quantity than other types of qubits, suggesting the possibility that circuit-NFS could be implemented sooner than the low-resource quantum NFS.

Conclusion

A potential speedup to integer factorization comes from replacing the search for smooth numbers in the NFS by finding those numbers using a SAT solver. This requires solving a circuit that detects if $F(a, b)$ is smooth upon input a and b. Two natural circuits for that task are the circuit with variable exponents of Fig. 1 which explicitly lists all primes that can be factors and the circuit with variable factors of Fig. 3 which relaxes the requirement that these factors are prime. Both have too many input wires for any exponential-time SAT solver to provide any asymptotic speedup over brute-force search.

Despite the exponential upper bound on the runtime of SAT solvers, practical solvers are known to perform well on certain problems by picking up on patterns in the problem instances. One could hope that a speedup over the theoretical upper bound is therefore achieved in practice on these particular circuits, although this speedup would have to be superpolynomial in order to result in more efficient integer factorization. Benchmarks on the variable exponents circuit suggest that no such speedup is realized in practice.

The circuit-NFS algorithm is specialized to the smoothness detection problem in the sense that the ECM performs well for finding small factors. Our algorithm has at least the same asymptotic runtime as the classical NFS. Measurements of solving smoothness detection circuits however indicate that there is a massive overhead to this approach. Any speedup in SAT solving (be it quantum or classical) needs to make up for this overhead before resulting in a speedup for factoring. Still, if the overhead is only constant then any γ-speedup will eventually be sufficient. Given a quantum annealer that solves SAT instances with any γ-speedup ($\gamma > 1$) over classical search, circuit-NFS performs asymptotically better than the classical NFS. If a full quadratic speedup is attained, circuit-NFS achieves the asymptotic time complexity of the low-resource quantum NFS, while perhaps not requiring a fault-tolerant quantum computer (depending on the quantum SAT solving device).

Open problems remain, such as benchmarking circuit-NFS on the ECM circuit and estimating its quantum resource requirements.

Received: 31 March 2020; Accepted: 19 August 2020
Published online: 14 September 2020

References

1. Mosca, M. & Verschoor, S. R. Factoring semi-primes with (quantum) SAT-solvers. Preprint at https://arxiv.org/abs/1902.01448 (2019).
2. Buhler, J. P., Lenstra, H. W. & Pomerance, C. Factoring integers with the number field sieve. In *The Development of the Number Field Sieve* 50–94 (Springer, Berlin, 1993).
3. Grover, L. K. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing* 212–219 (1996).
4. Bernstein, D. J., Biasse, J. & Mosca, M. A low-resource quantum factoring algorithm. In *Post-Quantum Cryptography* 330–346 (Springer, Cham, 2017).
5. Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **41**, 303–332 (1999).
6. Lenstra, H. W. Factoring integers with elliptic curves. *Ann. Math.* **126**, 649–673 (1987).
7. Benjamin, S. C., Zhao, L. & Fitzsimons, J. F. Measurement-driven quantum computing: Performance of a 3-SAT solver. Preprint at https://arxiv.org/abs/1711.02687 (2017).
8. Lenstra, A. K., Lenstra Jr., H. W., Manasse, M. S. & Pollard, J. M. The number field sieve. In *Proceedings of the 22 Annual ACM Symposium on Theory of Computing* 564–572 (1990).
9. Wiedemann, D. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory* **32**, 54–62 (1986).
10. Bernstein, D. J. Circuits for integer factorization: a proposal. https://cr.yp.to/papers/nfscircuit.pdf (2001). Accessed 27 August 2020.
11. Liang, J. H., Ganesh, V., Poupart, P. & Czarnecki, K. Learning rate based branching heuristic for SAT solvers. In *Theory and Applications of Satisfiability Testing—SAT 2016: 19th International Conference*, 123–140 (2016).
12. Basso, J. M. V. & Verschoor, S. R. NFS-SAT. *Github:* https://github.com/sebastianv89/NFS-SAT (2019).
13. Bennett, C. H., Bernstein, E., Brassard, G. & Vazirani, U. Strengths and weaknesses of quantum computing. *SIAM J. Comput.* **26**, 1510–1523 (1997).

Acknowledgements

This work was supported in part by the Natural Sciences and Engineering Research Council and in part by the CIFAR. IQC and the Perimeter Institute are supported in part by the Government of Canada and P.I. is also supported in part by the Province of Ontario.

Author contributions

M.M. proposed and supervised the project and possible approaches. J.M.V.B. led the development of the theorems and their proofs. S.R.V. contributed to the theoretical analysis and developed the code, simulations and plots. All authors contributed to the text and reviewed the manuscript. Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to J.M.V.B.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.