Editors' Suggestion | Featured in Physics

Computational Speedups Using Small Quantum Devices

Vedran Dunjko,[1,2,*] Yimin Ge,[1,†] and J. Ignacio Cirac[1,‡]
[1]Max Planck Institut für Quantenoptik, Hans-Kopfermann-Straße 1, 85748 Garching, Germany
[2]LIACS, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, Netherlands

(Received 24 July 2018; revised manuscript received 14 September 2018; published 18 December 2018)

Suppose we have a small quantum computer with only M qubits. Can such a device genuinely speed up certain algorithms, even when the problem size is much larger than M? Here we answer this question to the affirmative. We present a hybrid quantum-classical algorithm to solve 3-satisfiability problems involving $n \gg M$ variables that significantly speeds up its fully classical counterpart. This question may be relevant in view of the current quest to build small quantum computers.

DOI: 10.1103/PhysRevLett.121.250501

Quantum computers use the superposition principle to speed up computations. However, it is not clear if they can be useful when they are, as expected for the foreseeable future, limited in size. The reason is that quantum (and classical) algorithms typically exploit global structures of problems, and restricting superpositions to certain block sizes will break that structure. Thus, for problems where arbitrarily sized quantum computers offer advantages, small quantum computers may be of no significant help given large inputs.

Here, we study this problem and show that this is not generally true: There are relevant algorithms that utilize the global structure, where quantum computers significantly smaller than the problem size can offer significant speedups. More precisely, we focus on the famous algorithm of Schöning for boolean satisfiability and present a modified hybrid quantum-classical algorithm that significantly outperforms its purely classical version, even given small quantum computers.

Satisfiability (SAT) problems are among the basic computational problems, and they naturally appear in many contexts involving combinatorial optimization, like scheduling or planning tasks, and in statistical physics. A prominent SAT problem is 3SAT, which involves clauses with up to three literals. 3SAT is the canonical example of the so-called NP-complete problems, believed to be exponentially difficult even for quantum computers. Nevertheless, quantum computers can still accelerate their solving [1,2], and given their ubiquity, they may become one of the most important applications of quantum computers. However, the best quantum algorithms, which "quantum-enhance" classical SAT solvers [1], require many qubits and are not directly applicable given small quantum computers. There are several possibilities for how to use a limited-size quantum device. For instance, one could speed up smaller, structure-independent subroutines, which occur, e.g., in the preparation phases of algorithms (e.g., whenever a search over a few items is performed, one could utilize Grover's algorithm [3]). However, for "genuine" speedups, e.g., those that interpolate between the runtimes of the fully classical and a fully quantum algorithm, according to the size of the available quantum device, one should attack the actual computational bottlenecks. As we show, if this is done straightforwardly, one may encounter a threshold effect: If the quantum device is too small, i.e., can handle only a small fraction of the instance, a naïve hybrid algorithm turns out to be slower than its classical version. Here, we demonstrate how this effect can sometimes be circumvented, in the context of satisfiability problems. Specifically, we provide a quantum-assisted version of a well-understood classical algorithm, which achieves genuine improvements given quantum computers of basically any size, avoiding the threshold effect [4]. Our results are applicable to broader kSATproblems and, more generally, highlight the characteristics of classical algorithms and methods, which can be exploited to provide threshold-free enhancements.

The 3SAT problems.—In SAT problems, we are given a boolean formula $F:\{0,1\}^n \to \{0,1\}$ over n binary variables $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$. The task is to find a satisfying assignment $\mathbf{x}$, i.e., fulfilling $F(\mathbf{x}) = 1$, if one exists. In 3SAT, F is defined by a set of L clauses $\{C_j\}$, where each clause specifies three literals $\{l_1^j, l_2^j, l_3^j, \}$. Each literal specifies one of the n binary variables (x) or a negated variable $(\bar{x})$; for instance, $\{l_1, l_2, l_3\}$ could be $\{x_3, \bar{x_5}, x_8\}$. An assignment of variables $\mathbf{x}$ thus sets the values of all literals. C_j is satisfied by $\mathbf{x}$ if any of the literals in C_j attains the value 1. A formula F, written as $F(\mathbf{x}) = \bigwedge_{j=1}^{L}(l_1^j \vee l_2^j \vee l_3^j)$ using standard logic operator notation, is satisfied by $\mathbf{x}$ if all its clauses are satisfied.

Classical algorithms.—Many classical 3SAT solvers are significantly faster than the brute-force search. Their performance can be characterized by a constant $\gamma \in (0,1)$, meaning that they solve 3SAT in a runtime of $O^*(2^{\gamma n})$ [6]. One of the best and most famous ones is the algorithm of

Schöning [8]. It initializes a random assignment of the variables, then repeatedly finds an unsatisfied clause, randomly selects one literal in that clause, and flips the corresponding variable. This sampling algorithm terminates once a satisfying assignment is reached or once this process is iterated $O(n)$ times. Schöning proved that the probability of this algorithm finding a satisfying assignment (if one exists) is at least $(3/4)^n$, which, by iteration, leads to a Monte Carlo algorithm with expected runtime $O^*(2^{\gamma_0 n})$ with $\gamma_0 := \log_2(4/3) \approx 0.415$. A significant speedup of the classical algorithm is a reduction of its value of γ. To study the potential of small quantum computers, we investigate whether small devices suffice to achieve such a reduction.

Straightforward hybrid algorithms for small quantum computers.—In [1] a quantum algorithm inspired by Schöning's method, which exploits amplitude amplification [9], was introduced. It solves 3SAT instances with n variables in runtime $O^*(2^{\gamma_0 n/2})$ and requires $\approx \beta n$ qubits for a $\beta > 2$. Given a quantum computer with only $M \approx \beta m$ qubits ($m \ll n$), one has a few options to achieve speedups. A "bottom up" approach would be to use the quantum algorithm as an m-variable instance solver, which is then used as a subroutine in an overarching classical algorithm. For instance, to tackle the problem of n variables, one could sequentially go through all possible partial assignments of $n - m$ variables. Each partial assignment induces a SAT problem with m variables, which could then be solved on the small quantum device. The runtime of such an algorithm is $O^*(2^{(n-m)+\gamma_0 m/2})$, which highlights the *threshold effect*: The hybrid algorithm becomes *slower* than the classical algorithm of Schöning if $m/n \lesssim 0.74$. Note that, unlike thresholds for speedups induced by prefactors suppressed in an asymptotic analysis and the O^* notation, which may prevent speedups for small instances, the present threshold effect is far more fundamental: If M/n is below a constant value, no speedup is possible even for arbitrarily large problem sizes. Roughly speaking, the main problem of hybrid algorithms using a small device as a subinstance solver is that they break the global structure exploited by the classical algorithm. This results in hybrid algorithms whose runtimes interpolate between a fully quantum runtime and something slower than the classical algorithm—hence the threshold effect. Alternatively, we explore a "top-down" approach, where the most computationally expensive subroutines of the classical algorithm are identified and quantum enhanced.

Our results.—We present a hybrid algorithm that avoids the threshold effect. Specifically, given a quantum computer with $M = cn$ qubits, where $c \in (0,1)$ is an arbitrary constant, our algorithm solves 3SAT with n variables in a runtime of $O^*(2^{(\gamma_0 - f(c) + \varepsilon)n})$, where $f(c) > 0$ is a constant and ε can be made arbitrarily small. The function $f(c)$ is involved, but it is almost linear for small c (see the Supplemental Material [10] for details). Critically, irrespective of its exact form, our result constitutes a polynomial speedup over Schöning's algorithm for arbitrarily small c. Our contribution is primarily conceptual, and we assume an error-free setting.

Algorithm description.—Our hybrid algorithm is a quantum-assisted version of de-randomized variants of Schöning's algorithm [7,11], reviewed next. Given a bitstring $\mathbf{x} \in \{0,1\}^n$, let $B_r(\mathbf{x})$ denote the r-ball centered at $\mathbf{x}$, i.e., the set of all bitstrings $\mathbf{y}$ differing from $\mathbf{x}$ in at most r positions (i.e., their Hamming distance is $\leq r$). Then, relying on coding theory, the space of possible assignments is covered by a number of r-balls. Given this covering set, specified by the centers of the balls, the algorithm sequentially checks whether there exists a satisfying assignment within each of the r-balls. This "space-splitting" algorithm reduces SAT to the problem of finding a satisfying assignment within an r-ball, called Promise-Ball-SAT (PBS). A deterministic algorithm PROMISEBALL$(F, r, \mathbf{x})$ for PBS was introduced in [11]. This is a simple, recursive divide-and-conquer algorithm: On input it takes a formula, specified by a set of clauses with at most three literals, a radius, and a center $\mathbf{x}$. The algorithm first checks some conditions for (un)satisfiability [if $r \leq 0$ and $F(\mathbf{x}) = 0$ or if any clause is empty], or if $\mathbf{x}$ is a satisfying assignment. Otherwise, in the recursive step, it finds the first unsatisfied clause C and calls PROMISEBALL$(F_{|l=1}, r-1, \mathbf{x})$ for every literal $l \in C$ (the variables, literals, and clauses are enumerated in some prespecified order). Here, $F_{|l=1}$ denotes the formula obtained by setting the variable corresponding to l to the value ensuring $l = 1$; i.e., all clauses involving l ($\bar{l}$) are removed (truncated). This algorithm solves PBS in time $O^*(3^r)$. For comparison, Schöning sampling solves it in $O^*(2^r)$. The overall runtime of the space-splitting algorithm of [11] can be expressed as a function of the runtime of the PBS-solving subroutine (see Supplemental Material [10]). What is relevant is that, whenever a PBS solver with runtime $O^*(2^r)$ (e.g., randomized Schöning) is used in the space-splitting algorithm, we recover Schöning's runtime with $\gamma_0 \approx 0.415$.

Note that every recursive call in PROMISEBALL reduces r by 1. To make use of a small quantum device, one can, once r becomes small enough, use a quantum algorithm for PBS instead of a classical call. This leads to a general approach to speed up algorithms, which recursively call themselves (or other subroutines) with ever-decreasing instance sizes, using small quantum devices. We call this the *standard hybrid approach*. However, there are two obstacles to consider. First, since PROMISEBALL is significantly slower than the algorithm of Schöning, this still leads to a threshold: M would have to be a large fraction of n to gain an advantage. Second, straightforward quantum implementations of PROMISEBALL require too many qubits [$\Omega(n)$], even if r is small. While it is not difficult to reduce this to $O(r \log n)$, the resulting hybrid algorithm, although avoiding a threshold, would have a very

low, subpolynomial, advantage and not yield a true improvement of γ. We circumvent this by using more involved memory structures combined with specialized algorithms which algorithmically delete unneeded information, leading to much better memory requirements of $O(r\log(n/r)+r+\log n)$, and a true speed-up.

To summarize, we next provide the following algorithms: (1) a nonrecursive variant of PROMISEBALL, which on input only takes r ternary choices of which literals to flip; (2) a straightforward reversible implementation of (1) needed for quantization, with two parts—QBALL_1 transforms the choices to corresponding variable labels, and QBALL_2 checks whether flipping the chosen variables in $\mathbf{x}$ (ball center) satisfies the formula; (3) a method to significantly reduce the memory requirements of storing a set of labels in the second algorithm, from $O(r\log n)$ to $O(r\log(n/r)+r)$, while preserving reversibility. Algorithms (1)–(3), combined with amplitude amplification, then form the quantum algorithm QBALL for PBS, which, combined with a faster classical PBS-based solver [7], forms our final algorithm (4).

Space- and time-efficient quantum algorithm for PBS.—First, we specify a nonrecursive (classical) variant of PROMISEBALL, which does not manipulate the entire formulas explicitly. This avoids an immediate need for $\Omega(n)$ qubits for "carrying" the formula as input. Afterwards, we optimize the memory required for a reversible implementation and use amplitude amplification to achieve a faster quantum algorithm.

The structure of PROMISEBALL yields a ternary tree of depth r, induced by the up to three choices of literals in the recursive step of the algorithm. Thus, a sequence of choices $s_1,\ldots,s_r\in\{1,2,3\}$ specifies a leaf in the tree and hence the subset of literals whose values have been flipped. Thus, the algorithm PROMISEBALL induces a mapping from $s_1,\ldots,s_r$ to a set of at most r variables to be flipped, denoted $V=\{v_1,\ldots,v_{r'}\}$, where $r'\le r$. The nonrecursive algorithm computes the list of variables V indexed by the sequence $\vec{s}=s_1,\ldots,s_r$, generates the candidate assignments $\mathbf{x}_V$ realized by flipping the values of the variables specified by V, and checks if they satisfy the formula. This subroutine can be executed in polynomial time. The nonrecursive PROMISEBALL simply goes through all 3^r sequences $\vec{s}$, yielding the runtime $O^*(3^r)$.

This can be turned into a quantum algorithm QBALL_{12}, realizing the mapping:

$$|s_1,\ldots,s_r\rangle|0\rangle|0\rangle \xrightarrow{\mathrm{QBALL}_{12}} \underbrace{|s_1,\ldots,s_r\rangle\overbrace{|V\rangle}^{\mathrm{QBALL}_2}}_{\mathrm{QBALL}_1}|F(\mathbf{x}v)\rangle, \quad (1)$$

where the first part (QBALL_1) generates the indices of the variables in V, and the second part (QBALL_2) verifies whether F is satisfied by $\mathbf{x}_V$. The full quantum algorithm for PBS, which we call QBALL, then uses amplitude amplification to find one sequence $\vec{s}$ which yields a satisfying assignment, using $O(3^{r/2})$ calls to $\mathrm{QBALL}_{1,2}$, each with polynomial runtime. We next give the basic ideas for how to implement the algorithm space efficiently (for details see the Supplemental Material [10]). For ease of presentation, we first show how QBALL_2 can be realized straightforwardly, using many ancillas, before reducing their number. A straightforward implementation of QBALL_2 would utilize n additional qubits and assign them the value $\mathbf{x}$. Then, the circuit would iterate through the registers specifying V [naïvely requiring $O(r\log n)$ qubits] and introduce (controlled) negations to those ancillary qubits selected by the values in V. This would finalize the variable presetting stage and set the input to the formula to $\mathbf{x}_V$. Next, we would sequentially evaluate each clause, by associating a gate controlled by the variable qubits corresponding to the variables occurring in the clause. This controlled gate applies the appropriate negations to realize the literals, increasing a counter if a clause is satisfied. After doing this for each clause, the output qubit is flipped only if the counter equals L, meaning all clauses are satisfied. Such a circuit uses $O(\log(L)+n)$ ancillas. Since $L=O(\mathrm{poly}(n))$, the key problem is the contribution of n. This is simplified by noting that, although the circuits our algorithms generate depend on F, $\mathbf{x}$ and r, we can w.l.o.g. assume that $\mathbf{x}=(0,\ldots,0)$ and subsume the negations directly into F [12]. Next, since the clauses are evaluated sequentially, we only require three variable-specific ancillas, specifying the bit values appearing in the current clause: For each clause, each of the three ancillas corresponds to the three variables occurring in that clause. The variable presetting stage is now done individually for each clause C_j: Before clause evaluation, the circuit iterates through the V-specifying register and flips the kth ancilla if the specification matches the kth variable within the clause C_j. The three ancillas are uncomputed after evaluating C_j and can be reused. This requires only $O(\log n)$ additional qubits.

The algorithm QBALL_1 is more involved. QBALL_1 comprises a main loop which sequentially adds one variable specification v_i to the already specified set as follows: The ith circuit block takes the specifications of the first $i-1$ variables $v_1,\ldots,v_{i-1}$ as inputs, iterates through all clauses, and evaluates each clause C_j (in a manner similar to QBALL_2), using the values $v_1,\ldots,v_{i-1}$ to correctly preset the clause-specific input. If the clause is not satisfied, it uses the value of s_i to select the specification of one variable occurring in C_j, taking into account that variables which have already been flipped cannot be selected again, and storing this specification as v_i. For reversibility, additional counters are used, but these can be uncomputed and recycled. The final compression relies on efficient encodings of V, which, as an ordered list, would use $O(r\log n)$ qubits. Since the ordering does not matter, instead of storing the positions, we can store the relative shifts v_1, $v_2-v_1,\ldots,v_{r'}-v_{r'-1}>0$ of a sorted version of

the list, using no more digits than necessary and a separation symbol to denote the next number. Since these values add up to at most n, one can see that $O(r\log(n/r)+r)$ qubits suffice for this encoding. This structure indeed encodes a set, erasing the ordering. However, straightforward algorithms that use encodings of sets instead of lists encounter reversibility problems. To illustrate this, note that in the process of adding a new variable to V, one must realize the two steps $|\{v_1,\ldots,v_{i-1}\}\rangle|0\rangle \mapsto |\{v_1,\ldots,v_{i-1}\}\rangle|v_i\rangle$, i.e., finding the new element, and $|\{v_1,\ldots,v_{i-1}\}\rangle|v_i\rangle \mapsto |\{v_1,\ldots,v_i\}\rangle|0\rangle$, i.e., placing it into the set, and, critically, freeing the ancillary qubits for the next step. However, this is irreversible since the information about which element was added last is lost. The full ordering information requires $O(r\log r)$ additional qubits, nullifying all advantages. Of course, one could instead realize the reversible operation $|\{v_1,\ldots,v_{i-1}\}\rangle|v_i\rangle|0\rangle \mapsto |\{v_1,\ldots,v_{i-1}\}\rangle|v_i\rangle|\{v_1,\ldots,v_i\}\rangle$, followed by applying the inverse of the entire circuit up to this point to uncompute $|\{v_1,\ldots,v_{i-1}\}\rangle|v_i\rangle$, but this would result in an exponential instead of polynomial runtime of QBALL_1. We circumvent this issue by splitting V into $O(\log r)$ sets of sizes $1,2,4,\ldots$, and the loading of each larger block is followed by an algorithmic deletion of all smaller blocks. This ensures the overall number of qubits needed for this encoding is still $O(r\log(n/r)+r)$, at the cost of $2^{O(\log(r))}$ additional steps, which is just polynomial. These structures and primitives lead to an overall space- and time-efficient implementation of QBALL_1 (see Supplemental Material [10] for details). Combining these subroutines with a quantum search over $\vec{s}$, we obtain the algorithm QBALL, solving PBS in time $O^*(3^{r/2})$ and using $O(r\log(n/r)+r+\log n)$ qubits. In particular, QBALL outperforms the randomized algorithm of Schöning for PBS (2^r vs. $2^{\log_2(3)r/2}\approx 2^{0.79r}$).

Hybrid algorithm for 3SAT.—We could now use the standard hybrid approach for PROMISEBALL, i.e., call QBALL instead of PROMISEBALL when r is sufficiently small. However, this still leads to a threshold. This is resolved by using an improved deterministic algorithm for PBS [7], where coding theory is applied to cover the space of choice vectors $\vec{s}$. This yields an algorithm with runtime $O^*((2+\epsilon)^r)$, where ϵ can be chosen arbitrarily small—thus, the runtime of this algorithm for PBS essentially matches the runtime of Schöning. While we do not need the details of this algorithm, the critical point is that, like PROMISEBALL, it recursively calls itself to solve PBS with ever smaller values of r (sequentially reduced by a quantity depending on ϵ). Now, we can apply the standard hybrid approach. Since QBALL beats the runtime of this improved classical algorithm for PBS, the hybrid algorithm is faster than Schöning's, and, unlike using PROMISEBALL, there is no threshold induced by slow classical algorithms.

To estimate the runtime of our algorithm, note that since QBALL only requires $O(r\log(n/r)+r+\log n)$ qubits, a device with $M=cn$ qubits can solve PBS for $r=\beta(c)n$ for some $\beta(c)>0$. Since the hybrid algorithm replaces a classical subroutine of runtime $O^*(2^r)$ with a subroutine of runtime $O^*(3^{r/2})$ in a recursion tree below depth $r=\beta(c)n$, the runtime of the hybrid algorithm beats that of the classical one by a factor of $O^*((\sqrt{3}/2)^{\beta(c)n})$. Thus, the combined runtime of the hybrid algorithm is $O^*(2^{(\gamma_0-f(c)+\varepsilon)n})$ for $f(c)=\log_2(2/\sqrt{3})\beta(c)\approx 0.21\beta(c)$. The forms of β and f are involved, but in the relevant regime of small c, $f(c)$ achieves almost linear scaling, specifically $f(c)=\Theta(c/\log(c^{-1}))$ (for details see Supplemental Material [10]).

Conclusions.—We have shown that a small quantum computer can speed up relevant classical algorithms even for significantly larger inputs. While obvious for structureless scenarios (e.g., unstructured search), when considering algorithms that use the problem's structure, like Schöning's algorithm, speedups are nontrivial: The way the problem is partitioned must maintain the algorithm's structure to avoid thresholds. Our algorithm achieves a significant speedup, namely, a reduction of the relevant parameter γ, characterizing runtimes of the form $O^*(2^{\gamma n})$. The speedup holds relative to a variant of Schöning's algorithm. Our results, however, generalize to other algorithms based on Schöning's approach (e.g., Refs. [13,14], since those rely on better initial assignments) and to the variants handling kSAT ($k>3$). Historically, the best classical SAT solvers with provable bounds are based either on the ideas of Schöning or on the approach of [15], which includes the current record holder [16]. It would be interesting to see whether this second class of algorithms is also amenable to the types of enhancements achieved here. The broader question of this work is becoming increasingly more relevant given the current progress in prototypes of small quantum computers [17–19]. As our contribution is conceptual, we assume an error-free scenario. Still, our results may also have pragmatic relevance. Indeed, while the number of physical qubits of implementations is rapidly growing, the number of protected logical qubits we may expect in the near term is likely to be very limited. In practice, both the overheads and the noise may be bottlenecks to exploit small devices in general [20]. Thus, it would be particularly interesting to optimize the overheads of our algorithm. Specifically, any methods to decrease the number of gates and ancillas would increase the tolerance of our scheme. Finally, an in-depth analysis of optimal device-specific implementations could further help make our algorithms suitable for near-term realizations.

V. D. acknowledges support from the Alexander von Humboldt Foundation. J. I. C. and Y. G. acknowledge the ERC Advanced Grant QENOCOBA under the EU Horizon 2020 program (Grant Agreement No. 742102).

*v.dunjko@liacs.leidenuniv.nl
†yimin.ge@mpq.mpg.de
‡ignacio.cirac@mpq.mpg.de

[1] A. Ambainis, ACM SIGACT News **35**, 22 (2004).
[2] A. Ambainis, K. Balodis, J. Iraids, M. Kokainis, K. Prūsis, and J. Vihrovs, arXiv:1807.05209.
[3] L. K. Grover, in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, STOC '96* (ACM, New York, NY, USA, 1996), pp. 212–219.
[4] Our scenario is related to, but distinct from, Ref. [5] which considers how a classical computer can help simulate a given quantum algorithm which requires only slightly more qubits than are available. In contrast, we change the algorithm, i.e., propose hybrid algorithms which use a significantly smaller quantum device, to speed up a fixed classical algorithm. The two approaches are also complementary in their applicability: The main results of [5] are most powerful for shallow computations and could only be directly applied to SAT solving if one has an exponentially sized quantum computer to begin with, as the computation may be exponentially deep.
[5] S. Bravyi, G. Smith, and J. A. Smolin, Phys. Rev. X **6**, 021043 (2016).
[6] The O^* notation suppresses the terms that contribute only polynomially; see, e.g., Ref. [7].
[7] R. A. Moser and D. Scheder, in *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing, STOC '11* (ACM, New York, NY, USA, 2011), pp. 245–252.
[8] U. Schöning, in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society, Washington, DC, 1999), pp. 410–414.
[9] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, arXiv: quant-ph/0005055.
[10] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevLett.121.250501, for the details of the algorithms and of the runtime analysis.
[11] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning, Theor. Comput. Sci. **289**, 69 (2002).
[12] Let $F_{\mathbf{x}}$ be the formula obtained by negating any literal in F which corresponds to a variable to which $\mathbf{x}$ assigns the value 1, so that $F_{\mathbf{x}}(0,\ldots,0) = F(\mathbf{x})$. By subsuming $\mathbf{x}$ into F, we mean that we use $F_{\mathbf{x}}$ instead of F.
[13] T. Hofmeister, U. Schöning, R. Schuler, and O. Watanabe, in *STACS 2002*, edited by H. Alt and A. Ferreira (Springer, Berlin, Heidelberg, 2002), pp. 192–202.
[14] K. Iwama, K. Seto, T. Takai, and S. Tamaki, in *Algorithms and Computation*, edited by O. Cheong, K.-Y. Chwa, and K. Park (Springer, Berlin, Heidelberg, 2010), pp. 73–84.
[15] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane, J. ACM **52**, 337 (2005).
[16] T. Hertli, SIAM J. Comput. **43**, 718 (2014).
[17] IEEE spectrum, IBM edges closer to quantum supremacy with 50-qubit processor, https://spectrum.ieee.org/tech-talk/computing/hardware/ibm-edges-closer-to-quantum-supremacy-with-50qubit-processor.
[18] American Physical Society meeting, Engineering superconducting qubit arrays for quantum supremacy, http://meetings.aps.org/Meeting/MAR18/Session/A33.1.
[19] Intel newsroom. 2018 ces: Intel advances quantum and neuromorphic computing research, https://newsroom.intel.com/news/intel-advances-quantum-neuromorphic-computing-research.
[20] J. Preskill, Quantum **2**, 79 (2018).