Solving maximally-constrained 1-SAT problems with oracular access

Vojtěch Havlíček
ETH Zurich, Computational Physics Group

Antony Milne and Andrew Simmons
Imperial College London, Controlled Quantum Dynamics Theory Group

Abstract

Boolean satisfiability problems are an important class of functions in the fields of computational complexity and query complexity. Using Microsoft's LIQ$Ui\,|\rangle$ simulator we demonstrate how the family of maximally constrained 1-SAT problems displays a striking separation between classical and quantum query complexity with oracular access. We optimize existing quantum algorithms so that we can simulate significantly more qubits compared to the naïve algorithm, implement our scheme fully using LIQ$Ui\,|\rangle$, and investigate the scaling of the computational resources required.

I. INTRODUCTION

We demonstrate a classical simulation of an algorithm to solve the maximally-constrained 1-SAT problem with oracular access. The problem we address shows a linear separation in classical and quantum *query complexities*.

Comparison of quantum algorithms to their classical analogues is a nontrivial task. One possible way to address the matter is to define a measure of complexity that is independent of the physical efficiency of the algorithm; the notion of query complexity achieves exactly that. Intuitively, a *deterministic* query complexity estimates the number of queries one needs to pose to an oracle before knowing the successful algorithm outcome with certainty, in the worst or average case.

The first known example of classical to quantum query complexity separation was the Bernstein-Vazirani algorithm, both in its recursive and (stronger) non-recursive form. The non-recursive version of the algorithm has linear separation in query complexity from its classical counterpart – it takes only one query of a quantum oracle to find the solution, whereas the number of classical queries scales linearly [1, 2]. Using Microsoft's LIQ$Ui\,|\rangle$ we implement and study a less trivial algorithm that achieves the same query complexity separation.

II. QUERY COMPLEXITY FOR MAXIMALLY-CONSTRAINED 1-SAT

Satisfaction problems can be considered the backbone of computational complexity theory: unconstrained SAT was the first problem known to be NP-complete [3]; #SAT is often regarded as the standard #P-complete problem; and TQBF (true quantified Boolean formula) one of the canonical PSPACE-complete problems. Here we will consider 1-SAT, which is a subset of the family of k-SAT problems [4–6].

Definition 1 (k-SAT decision problem)**.** *A problem instance of the k-SAT decision problem is specified by a Boolean formula, given in conjunctive normal form, in which each clause contains at most (or equivalently, exactly) k literals or negated literals. Given a problem instance, to solve the* k-SAT decision problem *is to output 1 if there is an assignment of the variables that satisfies the formula, and 0 otherwise.*

For example, $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$ is an example of a 3-SAT instance. A 1-SAT instance, then, contains no $\vee$ (logical OR) symbols and is merely a conjunction of literals or negated literals. It is clear that there exists a linear-time algorithm for the solution of such a Boolean formula; the formula is satisfiable if and only if there is no literal that appears both positively and negated in the conjunction. Likewise, there is a polynomial-time algorithm for 2-SAT, although k-SAT is NP-complete for $k \geq 3$.

We focus our attention not on the decision problem form of 1-SAT but rather to a variant of this problem with restricted oracular access, particularly relevant in the context of query complexity. We ask the following question: how many queries to a specific oracle does it take for a classical or quantum algorithm to determine the satisfying assignment to the problem? We define our oracle as follows:

Definition 2 (1-SAT (mod-k) oracles)**.** *The classical 1-SAT (mod-k) oracle $O_a : \{0,1\}^n \to \mathbb{Z}_k$ takes as input a bit-string representing an assignment of variables and returns, mod-k, the number of incorrectly-assigned variables evaluated against the 1-SAT formula a. The quantum 1-SAT oracle is a unitary that takes the proposed assignment $|x\rangle$ and encodes in an ancilla the number of errors compared to the correct assignment: $O_a\,|x\rangle\,|0\rangle = |x\rangle\,|err\rangle$.*

Furthermore, we restrict our discussion to the cases in which we have a *promise* that a is a maximally-constrained 1-SAT formula [4]. In these cases, each variable appears in the Boolean formula and so there is known to be exactly one satisfying assignment of the variables. While there is a query complexity separation of linear order for all k, in this paper we will focus on the case $k = 4$. We will see that even in the case with this restrictive oracle, the quantum query complexity of this task is minimal; that is, it requires only a single query.

Theorem 1. *The classical query complexity for a deterministic algorithm of this task is $O(n)$, where n is the length of the 1-SAT instance.*

Proof. Any deterministic algorithm must have a specific starting state that is posed to the oracle; without loss of generality, we choose the all-zero string. The oracle then reveals how many bits of this string are incorrect, *viz* the number of instances of 1 in the target string. One then queries for binary strings: $10000\ldots$, $01000\ldots$, $00100\ldots$. Let $b_i \in \{0, 1\}$ be the i-th bit of the target string. If the i-th query increases the number of errors then $b_i = 0$; if it reduces the number of errors then $b_i = 1$. It is clear that the task can be completed in n queries exactly using this scheme.

We will now demonstrate that if the oracle returns the Hamming distance, mod 4, then the number of oracle queries must be at least $\frac{1}{2}n$. Each query yields an output that can reveal at most two bits of information about the string. Since n bits are necessary to specify the string, by definition, there must be at least $\frac{1}{2}n$ queries by the standard information-theoretic argument, *viz* that the state space of possible strings is at most quartered by every query. Hence, the problem can only be completed in $O(n)$ queries to the oracle.

We also note that this can easily be extended to a proof that for any randomized or deterministic algorithm, the average query complexity is at least $\frac{1}{2}n$. To do this requires the observation that for rooted exactly-4-ary trees, the average distance of a leaf from the root is minimized by the tree in which all leaves are $\frac{1}{2}n$ away from the root. Then note that randomized algorithms can be thought of as random samples over deterministic algorithms and can therefore not have improved average-case query complexity by the linearity of expectation. □

III. QUANTUM ADVANTAGE: MINIMAL QUERY COMPLEXITY

Interestingly, a quantum algorithm is capable of solving such 1-SAT instances with just a single oracle query, even given the restricted $k = 4$ or $k = 3$ oracles. We shall also see that the Hamming weight oracle we implement in this case plays an important role in certain other quantum query complexity scenarios and could easily be adapted accordingly as a direction of further work.

In order to simulate our protocol, we must provide an implementation of the oracle. Ideally, however, as is the case with all oracular algorithms, one should only have a black-box access to an oracle encoded by another party. We will assume that this is the case and give an algorithm that requires no knowledge of the inner workings of the oracle.

Since each variable appears once in the 1-SAT formula, either positively or negated, each 1-SAT instance can be represented by a bit string a of length n. The expected output of the oracle O_a, given an input x, is then the Hamming distance between x and a. This can be obtained by the bitwise sum of the input string with the target, and a 1-SAT oracle can therefore be simulated by a Hamming weight oracle. Following definitions in the previous section, we further constrain our oracle to output the Hamming weight mod 4. This is trivial to implement if a full Hamming oracle is given by discarding the information carried in each qubit line other than the two least significant.

Consider the quantum gate B, given by

$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}. \tag{1}$$

Given a bitstring input $|a\rangle$ of length n, we have

$$B^{\otimes n} |a\rangle = \frac{1}{\sqrt{2^n}} \sum_x i^{H(x \oplus a)} |x\rangle, \tag{2}$$

where $\oplus$ denotes a mod 2 bitwise sum and $H(x \oplus a)$ the Hamming distance between bitstrings x and a. We also have

$$(B^{-1})^{\otimes n} |a\rangle = \frac{1}{\sqrt{2^n}} \sum_x (-i)^{H(x \oplus a)} |x\rangle. \tag{3}$$

Hence the gates $B^{\otimes n}$ yield a phase shift based on Hamming weight, naturally implemented mod 4 (since $i^{H(x\oplus a)} = i^{H(x\oplus a) \bmod 4}$). We implement the quantum oracle Q_a as a bit oracle [4], which acts as follows on the computational basis state $|x\rangle$ and two-qubit ancilla $|y\rangle$:

$$Q_a |x\rangle |y\rangle = |x\rangle |y + H(a \oplus x)\rangle. \tag{4}$$

For an ancilla given by

$$|y\rangle = \left(B^{-1}\right)^{\otimes 2} |00\rangle = \frac{1}{2}\left(|00\rangle - i\,|01\rangle - |10\rangle + i\,|11\rangle\right), \tag{5}$$

the action of the oracle Q_a on any given $|x\rangle$ will be phase-rotation by $-i$ for each 1 in the binary representation of $(x \oplus a)$.

In the algorithm, we first generate an even superposition over all the computational basis states by application of $H^{\otimes n}$ to an state $|00\ldots0\rangle$. We then attach the ancilla $|y\rangle$ and run Q_a, afterwards discarding the ancilla (which remains in a product state):

$$Q_a H^{\otimes n} |00\ldots0\rangle = \frac{1}{\sqrt{2^n}} \sum_x (-i)^{H(x\oplus a)} |x\rangle . \tag{6}$$

Comparing to Eq. (3) we see that this is the result of applying B^{-1} to each qubit in the x-register. Therefore, given the output of the oracle, we need only apply $B^{\otimes n}$ to the resultant state in order to obtain a, having used only a single oracle query.

IV. IMPLEMENTATION IN LIQUi|⟩

While the oracle is given as a black box in the strict query complexity scenario, for our simulation in LIQ$Ui\,|\rangle$ we need construct the quantum oracle for Hamming weight in an explicit manner. Fig. 1 shows the quantum circuit used to implement oracle. (Any circuits can be used for a general input of n qubits; for brevity we will only give illustrations for the specific case that $n = 3$.)

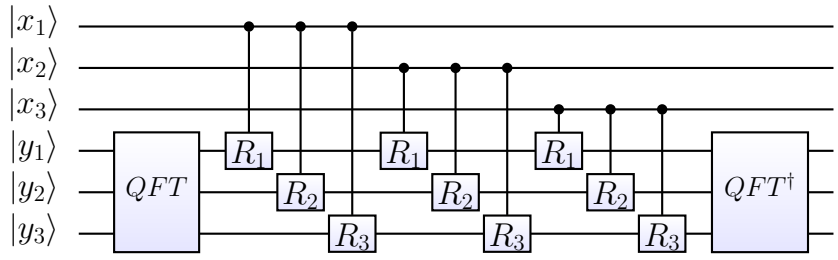

FIG. 1: The Hamming weight circuit (unfolded rotation gates for clarity).

First, we note that the Quantum Fourier Transform, a standard ingredient in quantum circuits, performs the following transformation on the ancilla $|y\rangle$:

$$QFT\,|y\rangle = \frac{1}{\sqrt{2^n}} \sum_q \omega^{yq} |q\rangle . \tag{7}$$

We then note that the controlled rotations, for each qubit line in the x-register, achieve

$$|q\rangle \to \omega^q |q\rangle . \tag{8}$$

Hence, performed on the Quantum Fourier Transform of $|y\rangle$, we have:

$$\frac{1}{\sqrt{2^n}} \sum_q \omega^{yq} |q\rangle \to \frac{1}{\sqrt{2^n}} \sum_q \omega^{yq}\omega^q |q\rangle \tag{9}$$

$$= \frac{1}{\sqrt{2^n}} \sum_q \omega^{(y+1)q} |q\rangle . \tag{10}$$

This is evidently the Quantum Fourier Transform of the state $|y+1\rangle$. Hence, each time these controlled rotations are applied, the value of the ancilla register increases by one, achieving the intended behaviour of our Hamming weight circuit.

We now note that this Hamming weight circuit can be easily transformed into a Hamming distance oracle by application of CNOT gates (if the hidden string is stored in a quantum register), or classically controlled X gates to transform a quantum state $|x\rangle$ into $|x \oplus a\rangle$. The Hamming weight circuit is then applied, followed by another series of CNOTs or classically controlled X gates. This completes the quantum implementation of the Hamming distance oracle.

So that our simulation can handle as many qubits as possible, we can consider how circuit elements may be combined to eliminate as many of the ancillary qubits as possible. Note that in equation (5), we identified one of the optimal states one could prepare to feed into the ancilla as being given by the inverse Quantum Fourier Transform of the state $|00\rangle$. Note also, that the first transformation applied to the ancilla in our Hamming distance oracle is a Quantum Fourier Transform; furthermore, since there are no more ancilla operations after the single oracle query, we can drop the Inverse Quantum Fourier Transform at the end of the oracle. The ancilla will be in a product state with the main register regardless, so the exact state of its qubits is immaterial. Making these changes, one then notes that the controlled rotations perform the identity rotation on all qubit lines except the least significant. Therefore, we can merely cut these $O(\log(n))$ qubits, a substantial saving. Fig. 2 shows the circuit developed after these savings are made.

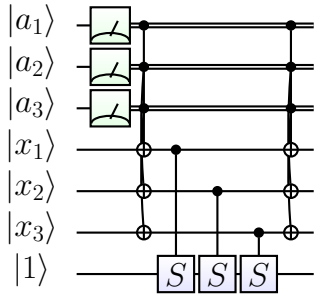

FIG. 2: Memory-optimized quantum circuit for Hamming weight (again, the controlled phase gates are shown unfolded for clarity).

Note that the gate B, as given in equation 1, can be written as $B = SHS$. Since the application of S will not alter measurement outcomes in the computational basis, we can fold it into the measurement to achieve a further time saving. This means that we effectively implement B^{-1} rather than B. The result of this is that the circuit outputs the bitwise inverse of the target bitstring, which could then be converted into the correct bitstring via trivial classical processing. This performance tweak saves n quantum gates, however, since we chose to construct our circuit out of simple gates where possible.

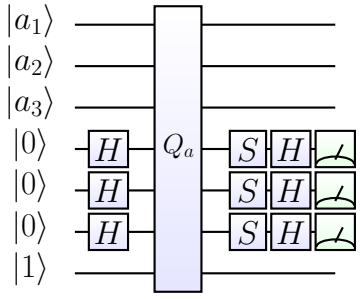

FIG. 3: The effective quantum circuit.

V. SAMPLE OUTPUT AND NUMERICAL RESULTS

Finally, we present an example of our algorithm running and investigate how the execution time scales with the size of 1-SAT problem. Our LIQ$Ui\,|\rangle$ code takes as input a bitstring a of length n, representing a maximally constrained 1-SAT instance with n literals. Whilst running, our code outputs some status updates to indicate what point of the algorithm the execution has reached. The ultimate output is then the solution string, which should be the bitwise inverse of a. For example, by setting $n = 20$ and generating a random bitstring a the output appears as follows:

```
0:0000.0/=============== Logging to: Liquid.log opened ================
0:0000.0/Solving 1-SAT problem with 20 qubits
0:0000.0/Input:
0:0000.0/  11011110101000111101
0:0000.0/----------------------------
0:0000.0/Generating superposition...
0:0000.0/Calling oracle...
0:0000.3/Performing measurements...
0:0001.0/Solution string found:
0:0001.0/  00100001010111000010
0:0001.0/=============== Logging to: Liquid.log closed ================
```

A crucial question to consider is how the simulation depends on n. Running LIQ$Ui\,|\rangle$ on a standard office laptop computer (Windows 7 64-bit, Intel Core i5 1.60GHz, 4GB RAM), LIQ$Ui\,|\rangle$ could allocate a maximum of 23 qubits. Note that the implementation outlined above uses $2n+1$ qubits. However, $|a\rangle$ is immediately measured and converted to a classical bitstring. We can therefore free up n qubits by not encoding a as qubits at all, and replacing the CNOT gates that perform $x \oplus a$ with Pauli X gates conditioned on the classical bits in a. This means that the total number of qubits required is only $n + 1$ (n qubits for $|x\rangle$ and 1 for the oracle's ancilla) and allows us to investigate bitstrings of up to length $n = 22$.

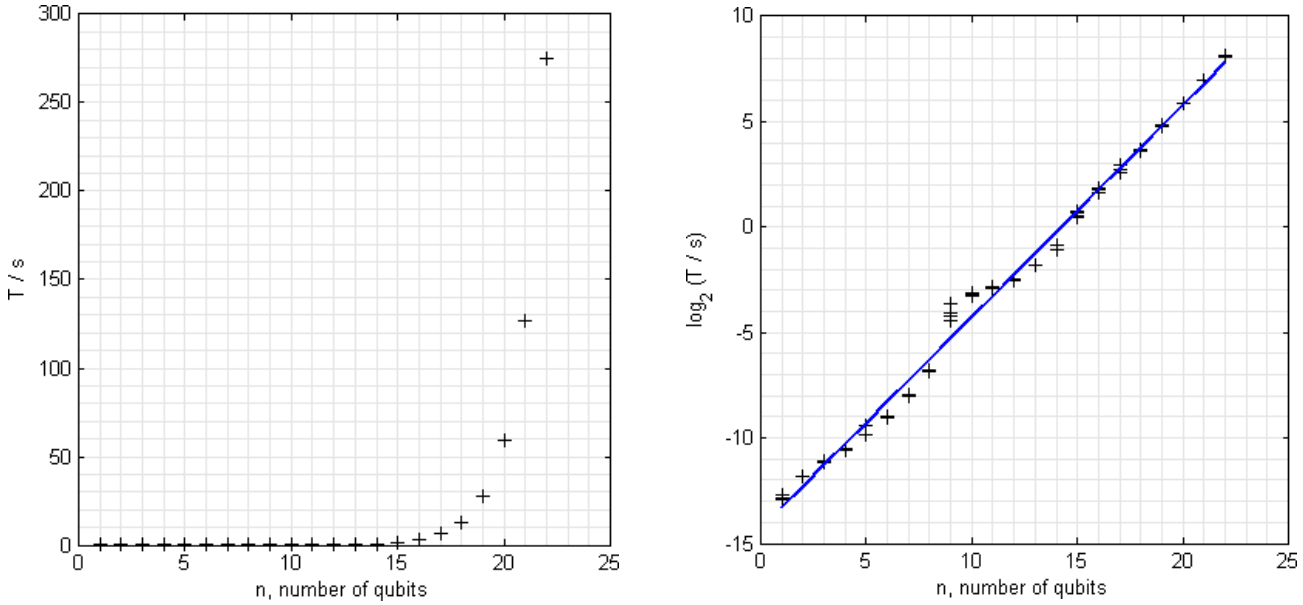

FIG. 4: Left: The execution time T of our algorithm, in seconds, for an input of length n (so that LIQ$Ui\,|\rangle$ allocates a total of $n + 1$ qubits). This suggests an exponential relationship between T and n. Right: $\log_2(T/s)$ against n, showing times for 5 separate instances of the algorithm for each n. The straight line confirms that an exponential relationship holds; specifically, we have $T \propto 2^{kn}$ with $k = 1.004$.

Using this scheme, we run the algorithm several times for each n (to ensure a fair comparison we use the all-ones input bitstring $a = 11\ldots1$ each time); the results are shown in Fig. 4. As in the above example, we see that on the laptop computer used, the execution time T for running the algorithm with $n = 20$ is about a minute; for a few qubits (say, $n \leq 10$), T is a few milliseconds. We find that the execution time scales exponentially with n – in fact, the exact relationship is remarkably close to $T \propto 2^n$. The algorithm requires a single call to the oracle, regardless of n, and the

oracle itself operates using $O(n)$ quantum gates. Hence the exponential scaling of T is clearly not derived from query or gate complexity. It must instead be a consequence of how LIQ$Ui\,|\rangle$ processes state vectors in the Hilbert space that has dimension 2^n.

In addition to this overall exponential trend, Fig. 4 exhibits some further features that should be noted. At $n = 10$ the execution time appears to jump up, and for $10 \leq n \leq 15$ the gradient is shallower than indicated by the general trend. We initially believed this to be a random error, possibly caused by peculiarities involving CPU caching. However, exactly the same behaviour is exhibited when running the timing tests on a different machine (OS X 64-bit, Intel i7 2.9 GHZ, 8GB RAM). This indicates that the results are not an artefact of the particular computer used and might instead be explained by something more systematic and intrinsic to LIQ$Ui\,|\rangle$.

VI. CONCLUSION

We implemented an oracular algorithm showing linear separation between quantum and classical query complexities in Microsoft LIQ$Ui\,|\rangle$. We introduced the problem of maximally constrained 1-SAT with oracular access and presented analysis of query complexity in the classical and quantum cases. Quantum circuits for subroutines of the algorithm have been discussed in detail. We also provided a numerical benchmark of running time of our algorithm.

The Hamming weight oracle is a general tool that might find applications in a variety of quantum algorithms. In particular, we have developed most of the necessary subroutines to implement an algorithm by Iwama et al. that solves the counterfeit coin problem [7]; the most challenging part involves developing a deterministic algorithm to generate a superposition across all evenly distributed partitions of a set. As a direction for further work it would be rewarding to investigate whether our Hamming weight oracle can be integrated into a full implementation of this larger quantum algorithm, which interestingly achieves a quartic reduction in query complexity compared to the best classical algorithm.

Acknowledgements

The authors would like to thank Ashley Montanaro, Kazuo Iwama and Harumichi Nishimura for their time spent discussing problems related to the topics of this paper, and Microsoft's Quantum Architectures and Computation Group (QuArC) for developing LIQ$Ui\,|\rangle$.

[1] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, October 1997.
[2] D. Bacon. Quantum computing. 2006.
[3] M. R. Garey and D. S. Johnson. *Computers and Intractability: a guide to the theory of NP-completness.* W. H. Freeman, 1979.
[4] A. Montanaro. *Structure, randomness and complexity in quantum computation.* PhD thesis, University of Bristol, 2007.
[5] T. Hogg. Solving highly constrained search problems with quantum computers. *Journal of Artificial Intelligence Research*, 10:39–66, February 1999.
[6] M. Hunziker and D. A. Meyer. Quantum algorithms for highly structured search problems. *Quantum Information Processing*, 1(3), June 2002.
[7] K. Iwama, H. Nishimura, R. Raymond, and J. Teruyama. Quantum counterfeit coin problems. *Theoretical Computer Science*, 456:51 – 64, 2012.