Simple encoding of a quantum circuit amplitude as a matrix permanent

Terry Rudolph

*Optics Section, Blackett Laboratory, Imperial College London, London SW7 2BW, United Kingdom
and Institute for Mathematical Sciences, Imperial College London, London SW7 2PG, United Kingdom*

(Received 18 September 2009; published 25 November 2009)

A simple construction is presented which allows computing the transition amplitude of a quantum circuit to be encoded as computing the permanent of a matrix which is of size proportional to the number of quantum gates in the circuit. This opens up some interesting classical Monte Carlo algorithms for approximating quantum circuits.

DOI: 10.1103/PhysRevA.80.054302 PACS number(s): 03.67.Lx

In a recent article [1] Loebl and Moffatt gave a method for expressing the computation of the Jones polynomial of a braid in terms of a matrix permanent. Although computing permanents is believed difficult (#P complete in the language of complexity theory), there exist probabilistic algorithms [2] which sample the permanent. This suggests some interesting new classical algorithms for estimating the output amplitudes of quantum circuits because evaluating the Jones polynomial at certain roots of unity is bounded-error quantum polynomial (BQP) complete [3]. The route to encoding a quantum circuit as the Jones polynomial of a knot, and then as a matrix permanent, is somewhat complicated—the purpose of this Brief Report is to present a simpler construction.

We restrict to quantum circuits built from Toffoli and Hadamard gates, which are universal [4,11]. We rely heavily on the construction of Dawson *et al.* [5]. There it is shown how the transition amplitude for such a quantum circuit is equivalent to counting the number of solutions of a GF(2) (i.e., XOR-AND) polynomial over some binary valued variables. More precisely, the results of [5] imply the following: given a quantum circuit U and input and output computational basis states $|\text{in}\rangle, |\text{out}\rangle$ the amplitude $\langle\text{out}|U|\text{in}\rangle$ can be expressed as the *difference* in the number of solutions to a GF(2) polynomial over (roughly) as many Boolean variables as there are Hadamard gates in the circuit. It is perhaps easiest to explain the construction using an example such as in Fig. 1. The $a_i, b_i, \ldots$ are Boolean variables, which we imagine traveling along the qubit lines. Every time the qubit goes through a Hadamard gate we create a new such variable, and whenever a variable z_i travels through the target of a Toffoli gate we replace it by $z_i \oplus x_i y_i$, where x_i, y_i are the variables at the control lines of the Toffoli gate as indicated.

Having labeled the circuit with these variables, we then create the function $f(x)$ by taking the sum (mod 2) of the product of every pair of variables on either side of a Hadamard gate. For the example of Fig. 1 we obtain

$$f(x) = a_1a_2 \oplus a_2a_3 \oplus a_3a_4 \oplus b_1b_2 \oplus b_2b_3 \oplus (b_3 \oplus d_2c_4)b_4 \oplus b_4b_5 \oplus c_1c_2 \oplus (c_2 \oplus b_2a_2)c_3 \oplus c_3c_4 \oplus c_4c_5 \oplus d_1d_2 \oplus d_2d_3.$$

If we are interested in, for example, the amplitude $\langle 0011|U|0000\rangle$ we then fix the input and output variables of f accordingly: in this case we would set $a_1=b_1=c_1=d_1=a_4=b_5=0$, $c_5=d_3=1$, and f simplifies to

$$f(x) = a_2a_3 \oplus b_2b_3 \oplus b_3b_4 \oplus d_2c_4b_4 \oplus c_2c_3 \oplus b_2a_2c_3 \oplus c_3c_4 \oplus c_4 \oplus d_2.$$

What is shown in [5] is that given a function constructed in this way, one has

$$\langle\text{out}|U|\text{in}\rangle = \frac{\#_0 - \#_1}{\sqrt{2}^h}. \qquad (1)$$

Here $\#_0, \#_1$ denote the number of solutions to the equations $f(x)=0$ and $f(x)=1$, respectively, and h denotes the number of Hadamard gates in the circuit. Note that $\#_0+\#_1=2^v$, where v is the number of variables in the function f once the input and output qubit values have been fixed. If there are q qubits in the circuit then $v=h-q$.

There are several other points to note in terms of the construction of f. First it will be convenient to assume that every variable goes through at most one Toffoli gate—this can be arranged by inserting double Hadamard (i.e., identity) gates where necessary [12]. This should also be done at the final outputs to the quantum circuit. Doing so ensures that the function f has the following properties: (i) it is (monotone) cubic; (ii) every variable appears in at most one cubic clause and two quadratic clauses.

Now counting solutions to a general GF(2) polynomial is a #P-complete problem [6]. That is, it has the same complexity as computing the permanent of a matrix—the prototypical #P problem—as was famously proven by Valiant in 1979 [7]. So we know that in principle we can map between these

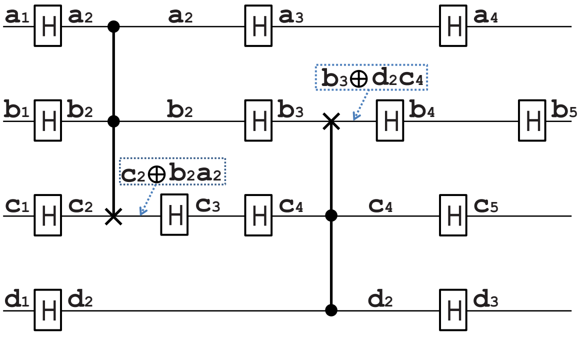

FIG. 1. (Color online) Mapping from a standard Toffoli-Hadamard circuit to counting solutions of a GF(2) polynomial.

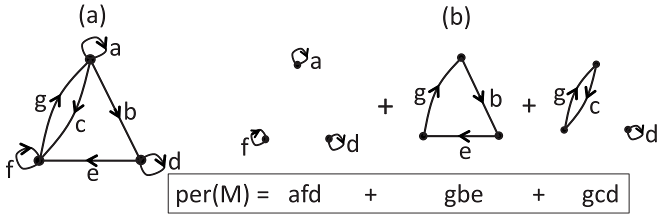

FIG. 2. The permanent of the matrix

$$M = \begin{pmatrix} a & b & c \\ 0 & d & e \\ g & 0 & f \end{pmatrix}$$

is the sum of the weighted cycle covers of the associated graph.

problems and find some matrices M_0 and M_1 such that $\mathrm{per}(M_0)=\#_0$ and $\mathrm{per}(M_1)=\#_1$, and then

$$\langle \mathrm{out}|U|\mathrm{in}\rangle = \frac{\mathrm{per}(M_0) - \mathrm{per}(M_1)}{\sqrt{2^h}}.$$

However, the actual mapping between these problems is not particularly simple or economical. In addition Valiant's construction of the matrix to count solutions of a satisfiability problem is also not particularly economical.

The purpose of this Brief Report is to present a very simple, direct, and economical construction relating quantum computing to evaluating a matrix permanent, which is also considerably more efficient than following the preceding route. Moreover, instead of expressing the solution to the problem as the difference in two matrix permanents, we will construct a single matrix or graph G such that

$$\langle \mathrm{out}|U|\mathrm{in}\rangle = \frac{\mathrm{per}(G)}{\sqrt{2^h}}. \tag{2}$$

The route to finding G uses some of the same tricks as in Valiant's proof. As this Brief Report is intended to also be accessible for physicists possibly unfamiliar with Valiant's result, we will try and make the presentation as self-contained as possible.

Any $n \times n$ matrix can be considered the weighted adjacency matrix for a weighted graph on n vertices, where the weight on the edge between vertices i and j is simply the (i,j)th element of the matrix. The permanent of a matrix, formally defined by

$$\mathrm{per}(M) = \sum_{\pi \in S_n} \prod_i M_{i,\pi(i)},$$

with S_n as the symmetric group on n symbols, is then graphically equivalent to the sum total of the weighted cycle covers of the graph: a cycle in a graph is a closed path; a cycle cover is a set of cycles for which each vertex belongs to one and only one cycle. The weight of a cycle cover is the product of the weights on the edges involved in that particular cycle cover—so the permanent is the sum of all such weights. An example is provided in Fig. 2. A brief summary of how the permanent arises in some physical considerations can be found in [8]; from the perspective of this Brief Report the close connections between evaluating permanents and certain statistical mechanical models suggest there should be connections between this work and that of [9].

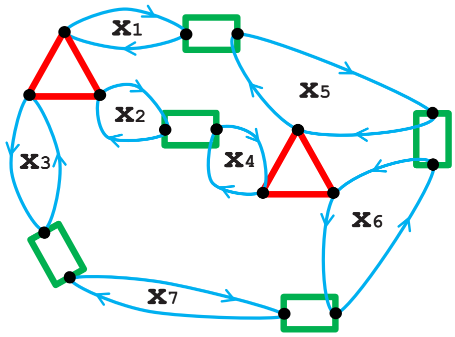

FIG. 3. (Color online) A big-picture view of the construction. The external edges form loops through the graph gadgets, and each such cycle is associated with one particular Boolean variable x_i. If the cycle *is* traversed in a particular cycle cover then that corresponds to setting that particular variable to 0. Conversely, if the particular cycle *is not* traversed then this corresponds to the associated variable having a value of 1. The graph gadgets have two or three vertices connecting to external edges according to whether they are gadgets for a quadratic or cubic clause. This graph would correspond to the polynomial $x_1x_2x_3 \oplus x_1x_5 \oplus x_4x_5x_6 \oplus x_5x_6 \oplus x_2x_4 \oplus x_6x_7 \oplus x_3x_7$.

Let us first give an overall view of the construction. We will be constructing a graph in such a way that the presence/absence of one particular cycle in any given cycle cover corresponds to whether a particular Boolean variable x_i associated with this cycle is 0 or 1. We will use the convention that if the particular cycle is present in the cycle cover then this matches the variable assignment $x_i=0$; if it is not then $x_i=1$. Not all cycles within the graph will correspond to variable assignments—the ones which do we term external cycles. In the figures the "external edges" which can make up such cycles will be thicker and colored in blue (to aid the eye only—there is no mathematical difference between these edges and other edges in the graph). The overall graph will consist of some "graph gadgets" (small subgraphs) connected by external edges. An example is given in Fig. 3. Each of the gadgets corresponds to a clause—in the figure we show only the vertices of the gadget which connect to external edges. The blue external edges form loops around two or three of the graph gadgets according to whether the variable appears in two or three clauses, and obviously they loop through a clause gadget with their corresponding partners of that clause looping through the other vertices of the gadget.

Now as we compute the sum of the weighted cycle covers of the graph (i.e., the permanent of the associated matrix) each cycle cover in the sum corresponds to a particular assignment of values to the Boolean variables—i.e., it will have a particular set of external cycles traversed setting those variables to a value 0. The graph gadgets will be designed so that if *none* of the external edges connected to that gadget are traversed—corresponding to all of the variables in that clause being equal to 1—then the weight which that gadget contributes to the particular cycle cover is −1. In all other cases the weight contributed by that gadget will be +1. Re-

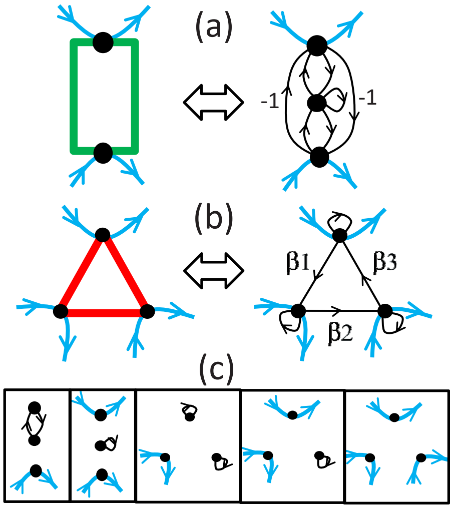

FIG. 4. (Color online) (a) The inner workings of the quadratic clause graph gadget. The weight on any edge is 1 unless otherwise indicated. (b) The inner workings of the cubic clause gadget. The weights β_i need only satisfy $\beta_1\beta_2\beta_3=-2$, which can be achieved by setting $\beta_1=-2$, $\beta_2=\beta_3=1$ if a graph with integer weights is desired. (c) The various ways in which the gadgets can be involved in a cycle cover with external edges. These are the cases when some of the associated Boolean variables in the clause are equal to 0 and, as can be seen, these cases all contribute a weight of +1 to the cycle cover. When no external edges are incident on the gadget the weight it contributes must be −1 as discussed in the text.

call that the weight of any given cycle cover is the *product* of the weights over all cycles in the cover. So for a fixed cycle cover (corresponding to a fixed assignment to the Boolean variables) the total weight will be +1 or −1 according to whether an even or an odd number of clauses are satisfied by that particular assignment. Assuming without loss of generality an even number of clauses in total, this in turn means that the weight of the particular cycle cover is +1 if $f(x)=0$ and −1 if $f(x)=1$. As we sum over all weighted cycle covers we automatically are calculating the difference in the number of solutions of $f(x)=0$ to $f(x)=1$, which is precisely what we need by Eq. (1).

The inner workings of graph gadgets which act in the desired manner are shown in Fig. 4. If in some cycle cover no external edges are incident on the gadget then its contribution will be −1, which can be readily verified by computing the permanents of their adjacency matrices:

$$\begin{pmatrix} 0 & -1 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & \beta_1 & 0 \\ 0 & 1 & \beta_2 \\ \beta_3 & 0 & 1 \end{pmatrix}.$$

If one, two, or three external edges are incident on the gadgets then the contribution to the cycle cover has weight +1; this is depicted in Fig. 4(c).

There is one potential problem which has not been addressed. What is to stop a particular cycle cover involving only *part* of an external cycle corresponding to some given variable. Why, for example, do we not get screwed up by cycle covers which, say, enter at one vertex of the graph gadget but leave at a different one? A figurative picture of such an undesirable type of cycle is given in Fig. 5(a).

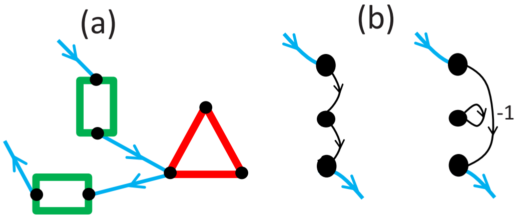

FIG. 5. (Color online) (a) The sort of cycle covers we need to avoid: cycles which only partially traverse an external cycle, as such setting the associated variable to 0 in one clause and 1 in another. (b) The inner workings of the quadratic clause gadget which ensure that any external edge must exit by the same vertex it entered. The two depicted contributions to the cycle cover have opposite signs and cause the necessary cancellation.

The possibility of such problematic cycles is ruled out by the internal workings of the quadratic clause graph gadget. This is shown in Fig. 5(b). Any cycle cover which enters the gadget along one external edge and tries to leave out via the external edge on the other side of the gadget has two possible paths for doing so. These paths pick up opposite signs, and so when summed over contribute 0 to the total. The process is somewhat reminiscent of Mach-Zender interferometry. Note that we did not need to design the cubic graph gadget to have the same property. This is because in the formulation we have chosen any variable appears in only one cubic clause, and it must then also appear in two quadratic clauses. The quadratic clause gadgets suffice to “force” an external edge which is incident into the cubic clause gadget to leave via the same vertex it entered.

In terms of the basic construction the final thing to mention is that it is simple to force the values at the boundaries (the input or output to the circuit) to be 1 or 0. This is done either by simply not connecting any external edges into the associated gadget (setting the variable to 1) or by forcing an external edge through the gadget by having that edge also loop through a vertex which has no “self-loop” (setting the

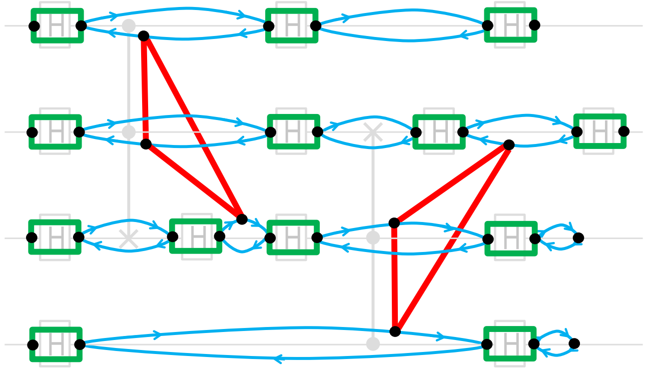

FIG. 6. (Color online) Putting everything together—how to draw the final graph G over the top of the associated circuit. Note it is the variable on the target line of a Toffoli gate which is created by the Hadamard that acts *after* the Toffoli that is involved in the associated cubic clause. This graph would be computing the transition amplitude with $|\text{in}\rangle=|1111\rangle$ and $|\text{out}\rangle=|1100\rangle$.

variable to 0). An example of this can be seen in Fig. 6 where the input qubits are all fixed to have value 1, and the top two qubits have value 1 at the output while the bottom two qubits are set to the value of 0 at the output [13].

The overall construction can be naturally laid out by drawing the graph directly on top of the circuit diagram. This is illustrated in Fig. 6 for the same circuit of Fig. 1.

Note that the number of vertices in the graph G we associate to a given circuit is basically three times the number of gates in the circuit. Let us denote this number of vertices as m. We have that

$$\langle \mathrm{out}|U|\mathrm{in}\rangle = \frac{\mathrm{per}(G)}{\sqrt{2}^{h}} = \mathrm{per}\left(\frac{G}{\sqrt{2}^{h/m}}\right).$$

If it were the case that $\|G/\sqrt{2}^{h/m}\| < 1$ then the results of [10] imply there would exist an efficient classical algorithm to simulate this quantum circuit.

We acknowledge the support of the EPSRC-GB and helpful comments by S. Severini.

[1] M. Loebl and I. Moffatt, e-print arXiv:0705.4548.

[2] C. D. Godsil and I. Gutman, *Algebraic Methods in Graph Theory* (North-Holland, Amsterdam, 1981), Vol. I, Vol. II; N. Karmarkar, R. Karp, R. Lipton, L. Lovasz, and M. Luby, SIAM J. Comput. **22**, 284 (1993); A. Barvinok, Random Struct. Algorithms **14**, 29 (1999); M. Jerrum, A. Sinclair, and E. Vigoda, J. ACM **51**, 671 (2004).

[3] M. Freedman, A. Kitaev, M. Larsen, and Z. Wang, Bull. Am. Math. Soc. **40**, 31 (2002); M. Bordewich, M. Freedman, L. Lovasz, and D. Welsh, Combinatorics, Probab. Comput. **14**, 737 (2005); D. Aharonov, V. Jones, and Z. Landau, *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing (STOC, 2006)* (ACM Press, New York, 1988), p. 427.

[4] Y. Shi, Quantum Inf. Comput. **3**, 84 (2003); D. Aharonov, e-print arXiv:quant-ph/0301040.

[5] C. M. Dawson, H. L. Haselgrove, A. P. Hines, D. Mortimer, M. A. Nielsen, and T. J. Osborne, Quantum Inf. Comput. **5**, 102 (2005).

[6] A. Ehrenfeucht and M. Karpinski, The Computational Complexity of (XOR, AND)-Counting Problems, ICSI Technical Report No. TR-90-033, 1990 (unpublished).

[7] L. G. Valiant, Theor. Comput. Sci. **8**, 189 (1979).

[8] T.-C. Wei and S. Severini, e-print arXiv:0905.0012.

[9] D. A. Lidar, New J. Phys. **6**, 167 (2004); J. Geraci and D. A. Lidar, Commun. Math. Phys. **279**, 735 (2008).

[10] L. Gurvits, *On the Complexity of Mixed Discriminants and Related Problems: Mathematical Foundations of Computer Science 2005* (Springer, Berlin, 2005), Vol. 3618.

[11] These gates are universal in the sense of allowing simulation of arbitrary quantum computations and not the generation of arbitrary unitary evolution. To generalize the results of this Brief Report to the latter, stronger, form of universality is not trivial—at least for this author. The place to start would be the obvious generalizations of the results of [5] to other finite fields.

[12] As pointed out by the referee, at first this appears incompatible with Eq. (1) remaining invariant, as the number of variables increases by 2, meaning the number of cases increases by 4, and yet the difference in number of solutions had better only double as $h \to h+2$ implies only an extra factor of $\sqrt{2}^2$ in the denominator. The trick is that such a splitting in terms of the terms of f takes the form $x_1x_2 \to x_1x_1' \oplus x_1'x_2' \oplus x_2'x_2$, and comparing all cases of the original and expanded terms, one finds that only one out of the four flip their parity. That is, in the new formula $f'(x)$ there are now $3\#_0+\#_1$ solutions to $f'(x)=0$ and $3\#_1+\#_0$ solutions to $f'(x)=1$, and so the difference is, to both my and the referee's relief, $2(\#_0-\#_1)$.

[13] In practice we can make things slightly more economical by removing some of these redundant vertices and using the fact that a suitable clause gadget for a clause consisting of a single variable is simply a single vertex with a self-loop of weight −1.