Topological quantum gate construction by iterative pseudogroup hashing

2011 New J. Phys. 13 025023

(http://iopscience.iop.org/1367-2630/13/2/025023)

New Journal of Physics
The open-access journal for physics

Topological quantum gate construction by iterative pseudogroup hashing

Michele Burrello[1,2,7], **Giuseppe Mussardo**[1,2,3,7] **and Xin Wan**[4,5,6]

[1] International School for Advanced Studies (SISSA), Via Bonomea 265, 34136 Trieste, Italy
[2] Istituto Nazionale di Fisica Nucleare, Sezione di Trieste, Italy
[3] International Centre for Theoretical Physics (ICTP), I-34014 Trieste, Italy
[4] Asia Pacific Center for Theoretical Physics, Pohang, Gyeongbuk 790-784, Korea
[5] Department of Physics, Pohang University of Science and Technology, Pohang, Gyeongbuk 790-784, Korea
[6] Zhejiang Institute of Modern Physics, Zhejiang University, Hangzhou 310027, People's Republic of China
E-mail: burrello@sissa.it and mussardo@sissa.it

New Journal of Physics **13** (2011) 025023 (19pp)
Received 16 November 2010
Published 23 February 2011
Online at http://www.njp.org/
doi:10.1088/1367-2630/13/2/025023

Abstract. We describe the hashing technique for obtaining a fast approximation of a target quantum gate in the unitary group SU(2) represented by a product of the elements of a universal basis. The hashing exploits the structure of the icosahedral group (or other finite subgroups of SU(2)) and its pseudogroup approximations to reduce the search within a small number of elements. One of the main advantages of the pseudogroup hashing is the possibility of iterating to obtain more accurate representations of the targets in the spirit of the renormalization group approach. We describe the iterative pseudogroup hashing algorithm using the universal basis given by the braidings of Fibonacci anyons. An analysis of the efficiency of the iterations based on the random matrix theory indicates that the runtime and braid length scale poly-logarithmically with the final error, comparing favorably to the Solovay–Kitaev algorithm.

[7] Authors to whom any correspondence should be addressed.

Contents

1. Introduction

The possibility of physically implementing quantum computation opens new scenarios for future technological development. This justifies the intensive efforts made by the scientific community to develop theoretical approaches suitable for effectively building a quantum computer and, at the same time, to reduce all the sources of errors that would spoil the achievement of quantum computation.

To implement quantum computation one needs to realize, through physical operations over the qubits, arbitrary unitary operators in the Hilbert space that describe the system. This task can be achieved by using a finite number of elementary gates that constitute a basis. A small set of gates is said to be *universal* for quantum computation if it allows us to approximate, at any given accuracy, every unitary operator in terms of a quantum circuit made of only those gates [1]. It can be shown that a basis able to reproduce every SU(2) operator and one entangling gate (as CNOT) for every pair of qubits is indeed universal [2]; therefore the problem of finding an approximation of unitary operators in SU(N) can be reduced to searching for an efficient representation, in terms of the elements of the basis, of single-qubit gates in SU(2) and of the two-qubit gate CNOT.

For SU(2), it is possible to find a universal set of single-qubit operators involving just two elementary gates, which we will label as σ_1 and σ_2 (not to be confused with Pauli matrices!), and their inverses, σ_1^{-1} and σ_2^{-2}. This means that every single-qubit gate can be efficiently approximated by a string of these four elementary elements. In the scheme of topological quantum computation, these fundamental gates are realized by elementary braid operation on the excitations of the system, the anyons. In order to be universal, the group obtained by multiplying the four σ gates must be dense in SU(2): it is therefore sufficient that σ_1 and σ_2 do

not belong to the same finite subgroup of SU(2). For what concerns controlled gates in SU(4) as CNOT, their approximation can be usually reduced to the case of operators in SU(2), as described in [5] and [24] in the context of topological quantum computation, the main subject of our following considerations.

The simplest way of obtaining an approximation of a given target gate $T \in \mathrm{SU}(2)$ using only four elementary gates $\sigma_{1,2}^{\pm 1}$ is to search for, among all the ordered products of N such operators, the one that best represents T, minimizing the distance to it (a rigorous definition of distance is given in section 3). This operation is called the *brute-force* procedure [5]. The number of all possible products of this kind grows exponentially in N as α^N (where $\alpha \approx 3$) and, because of the three-dimensional (3D) nature of SU(2), one can show that, for a suitable choice of the universal basis, the average error obtained with different targets decreases approximately as $\mathrm{e}^{-N/3\ln\alpha}$ (in section 3.2, we will describe in more detail the brute-force search for Fibonacci anyons).

This approach, consisting of a search algorithm over all possible ordered products up to a certain length, has an extremely clear representation if one encodes qubits using non-Abelian anyons. In this case, the computational basis for the quantum gate is the set of elementary braidings between every pair of anyons and their products are represented by the braids describing the world lines of these anyonic quasiparticles. Starting from this kind of universal basis, the search among all the possible products of N elements gives, of course, the optimal result, but the search time is exponential in N and therefore it is impractical to reach a sufficiently small error for arbitrary gates.

There are, however, other possible approaches that allow us to reach an arbitrary small error in a faster way, even if they do not obtain the best possible result in terms of the number N of elementary gates involved. The textbook example is the Solovay–Kitaev algorithm [1, 3, 4]. This algorithm provides a powerful tool for obtaining an approximation of arbitrary target gates in SU(2) at any given accuracy starting from an ϵ-net, i.e. a finite covering of SU(2) such that for every single-qubit operator T there is at least one gate inside the ϵ-net that has a distance from T smaller than ϵ. The Solovay–Kitaev algorithm is based on the decomposition of small rotations with elements of the ϵ-net and both the runtime and the length of the final product of elementary gates scale poly-logarithmically with the final error ε. The exponents depend on the detailed construction of the algorithm: the simplest realization of the algorithm, as provided in [5] in the context of topological quantum computation, is characterized by the following scaling:

$$N \sim (\ln(1/\varepsilon))^c \quad \text{with } c = \frac{\ln 5}{\ln(3/2)} \approx 3.97, \tag{1}$$

$$T \sim (\ln(1/\varepsilon))^d \quad \text{with } d = \frac{\ln 3}{\ln(3/2)} \approx 2.71. \tag{2}$$

As discussed in appendix 3 of [1] and in [4], a more sophisticated implementation of the Solovay–Kitaev algorithm realizes

$$N \sim (\ln(1/\varepsilon))^2 \ln(\ln(1/\varepsilon)), \tag{3}$$

$$T \sim (\ln(1/\varepsilon))^2 \ln(\ln(1/\varepsilon)). \tag{4}$$

The hashing algorithm, which was proposed in [6], has the aim of obtaining a more efficient approximation of a target operator than the Solovay–Kitaev algorithm in a practical regime,

with a better time scaling and without the necessity for building an ϵ-net covering the whole target space of unitary operators. Our main strategy will be to create a dense mesh $\mathcal{S}$ of fine rotations with a certain average distance from the identity and to reduce the search for the target approximation to the search among this finite set of operators that, in a certain way, play the role of the ϵ-net in the neighborhood of the identity. This set will be built by exploiting the composition property of a finite subgroup (the icosahedral group) of the target space SU(2) and exploiting also the almost random errors generated by a brute-force approach in order to approximate at a given precision the elements of this subgroup. Therefore, the algorithm allows us to associate a finite set of approximations with the target gate and each of them is constituted by an ordered product of the elementary gates chosen in a universal quantum computation basis. Since our braid lookup task is similar to finding items in a database given its search key, we borrow computer science terminology to name the procedure *hashing*.

With this algorithm we successfully limit our search to a small number of elements instead of an exponentially growing one as in the case of the brute-force search; this significantly reduces the runtime of the hashing algorithm. Furthermore, we can easily iterate the procedure in the same spirit as the renormalization group scheme, based on the possibility of restricting the set S to a smaller neighborhood of the identity at each iteration, obtaining in this way a correction to the previous result through a denser mesh.

In section 2, we briefly review the main ideas of topological quantum computation, which is the natural playground for the implementation of the hashing procedure. We use Fibonacci anyons to encode qubits and define their braiding operators that are the main example of universal elementary gates that we use to analyze our algorithm. In section 3, we introduce the basic ideas of the icosahedral group and its braid representations, which are pseudogroups. We describe in detail the iterative hashing algorithm in section 4 and analyze the performance of its iteration scheme in section 5. We conclude in section 6. In appendix, we derive the distribution of the best approximation in a given set of braids, which can be used to estimate the performance of, e.g., the brute-force search.

2. Topological quantum computation and Fibonacci anyons

The discovery of condensed matter systems that show topological properties and cannot be described simply by local observables has opened a new perspective in the field of quantum computation. Topological quantum computation is based on the possibility of encoding the qubits necessary for quantum computation in the topological properties of physical systems, and obtaining in such a way a fault-tolerant computational scheme protected by local noise [7]–[11].

For topological quantum computation one needs anyons with non-Abelian statistics. Unlike fermions or bosons, anyons are quasiparticles whose exchange statistics is described not by the permutation group, but by the braid group, generated by the elementary exchanges of anyon pairs. In particular, in certain two-dimensional (2D) topological states of matter, a collection of non-Abelian anyonic excitations with fixed positions spans a multi-dimensional Hilbert space and, in such a space, the quantum evolution of the multi-component wavefunction of the anyons is realized by braiding them.

Therefore, it is natural to consider the unitary matrices representing the exchange of two anyons as the elementary gates for a quantum computation scheme. In this way, the universal basis for the quantum computation acquires an immediate physical meaning and its elements are implemented in a fault-tolerant way; therefore the problem of approximating a target unitary

gate is translated to finding the best 'braid' of anyons that represents the given operator up to a certain length of ordered anyon exchanges [5].

There are several physical systems characterized by a topological order that are suitable to present non-Abelian anyonic excitations. The main experimental candidates are quantum Hall systems in semiconductor devices (see [12] for an introduction to the subject), as well as similar strongly correlated states in cold atomic systems [13], $p_x + \mathrm{i}p_y$ superconductors [14] and superconductor–topological insulator heterostructures [15].

One of the most studied anyonic models is the Fibonacci anyon model (see for example [5], [16]–[19]), so named because the dimension of their Hilbert space follows the famous Fibonacci sequence, which implies that their quantum dimension is the golden ratio $\varphi = (\sqrt{5}+1)/2$. Fibonacci anyons (denoted by ϕ as opposed to the identity 1) are known to have a simple fusion algebra ($\phi \times \phi = \phi + 1$) and to support universal topological quantum computation [8]. Candidate systems supporting the Fibonacci fusion algebra and braiding matrices include fractional quantum Hall states known as the Read–Rezayi state at filling fraction $\nu = 3/5$ [20] (whose particle–hole conjugate is a candidate for the observed $\nu = 12/5$ quantum Hall plateau [21]) and the non-Abelian spin-singlet states at $\nu = 4/7$ [22].

Every anyonic model, such as the one of Fibonacci anyons, is characterized by several main components: the superselection sectors of the theory are the different kinds of anyons that are present (1 and ϕ in our case), their behavior is described by the fusion and braiding rules that are linked through the associativity rules of the related fusion algebra, expressed by the so-called F matrices (see [11, 23] for a general introduction to the anyonic theories and [5] for its application to the Fibonacci case).

If we create two pairs of ϕs out of the vacuum, both pairs must have the same fusion outcome, 1 or ϕ, forming a qubit; therefore the logical value of the qubit is associated with the result of the fusion of the two Fibonacci anyons inside the first or the second pair, while the total fusion outcome is the vacuum. The braiding of the four ϕs can be generated by two fundamental braiding matrices (related by the Yang–Baxter equation)

$$\sigma_1 = \begin{bmatrix} \mathrm{e}^{-\mathrm{i}4\pi/5} & 0 \\ 0 & -\mathrm{e}^{-\mathrm{i}2\pi/5} \end{bmatrix}, \tag{5}$$

$$\sigma_2 = F^{-1}\sigma_1 F = \begin{bmatrix} -\tau\,\mathrm{e}^{-\mathrm{i}\pi/5} & -\sqrt{\tau}\mathrm{e}^{\mathrm{i}2\pi/5} \\ -\sqrt{\tau}\,\mathrm{e}^{\mathrm{i}2\pi/5} & -\tau \end{bmatrix}, \tag{6}$$

and their inverses σ_1^{-1}, σ_2^{-1}. Here $\tau = \varphi^{-1} = (\sqrt{5}-1)/2$. We note that σ_1 is the elementary braiding involving two anyons in the same pair; thus it is diagonal in the qubit basis; instead, σ_2 describes the braiding of anyons in different pairs and can be obtained by the change of basis defined by the associativity matrix F.

This matrix representation of the braidings generates a four-strand braid group B_4 (or an equivalent three-strand braid group B_3): this is an infinite-dimensional group consisting of all possible sequences of length L of the above generators and, with increasing L, the whole set of braidings generates a dense cover of the SU(2) single-qubit rotations.

In addition, Simon *et al* [17] demonstrated that, in order to achieve universal quantum computation, it is sufficient to move a single Fibonacci anyon around the others at fixed position. Thanks to this result, one can study an infinite subgroup of the braid group B which is the group of *weaves*, braids characterized by the movement of only one quasiparticle around the others. From a practical point of view, it seems simpler to realize and control a system of this kind;

therefore we will consider only weaves in the following. There is also another advantage in doing so: the elementary gates to cover SU(2) become σ_1^2, σ_2^2 and their inverses; therefore, the weaves avoid the equivalence between two braids caused by the Yang–Baxter relations, so that it is more immediate to find the set of independent weaves up to a certain length. In fact, the only relations remaining for Fibonacci anyons that give rise to different but equivalent weaves are the relations $\sigma_1^{10} = \sigma_2^{10} = 1$, because they imply that $\sigma_i^6 = \sigma_i^{-4}$ and one can always reduce weaves with terms of the kind $\sigma_i^{\pm 6}$ to shorter but equivalent ones.

To calculate the efficiency in approximating a target gate through a brute-force search, which gives the optimal result up to a certain length, it will be useful to calculate here the number of independent weaves of Fibonacci anyons; in general a weave of length L can be written as

$$\sigma_{p_1}^{q_1}\sigma_{p_2}^{q_2}\cdots\sigma_{p_s}^{q_s}, \tag{7}$$

where $p_i \in \{1, 2\}$, $p_i \neq p_{i\pm1}$ and $\sum_i |q_i| = L$. As mentioned above, to ensure that this braid is not equivalent to a shorter braid, one must have $q_i = \pm 2$ or ± 4.

Let us assume that the number of length L weaves is $N(L)$, consisting of $N_4(L)$ weaves ending with $\sigma_p^{\pm 4}$ and $N_2(L)$ weaves ending with $\sigma_p^{\pm 2}$. To form a weave of length $L+2$, the sequence must be appended by σ_r^2 or σ_r^{-2}. For sequences ending with σ_p^2 (or σ_p^{-2}), we can append σ_p^2 (or σ_p^{-2}), σ_{3-p}^2 or σ_{3-p}^{-2}. However, for sequences ending with σ_p^4 (or σ_p^{-4}), we can only append σ_{3-p}^2 or σ_{3-p}^{-2}. Therefore, we have the recurrence relations:

$$N(L) = N_4(L) + N_2(L), \tag{8}$$

$$N(L+2) = 2N_4(L) + 3N_2(L), \tag{9}$$

$$N_4(L+2) = N_2(L). \tag{10}$$

With an ansatz $N(L) \sim \alpha^{L/2}$ (and so are $N_4(L)$ and $N_2(L)$), we find $\alpha = 1 \pm \sqrt{3}$. Therefore, the exact number of weaves of length L is

$$N(L) = \left(1 - \frac{1}{\sqrt{3}}\right)\left(1 - \sqrt{3}\right)^{L/2} + \left(1 + \frac{1}{\sqrt{3}}\right)\left(1 + \sqrt{3}\right)^{L/2}, \tag{11}$$

which grows as $(1+\sqrt{3})^{L/2}$ asymptotically.

3. SU(2), subgroups and pseudogroups

3.1. Single-qubit gates and distances in SU(2)

The hashing algorithm allows us to approximate every target single-qubit gate in SU(2) exploiting the composition rules of one of its subgroups, as the icosahedral one; therefore, it is useful to consider the standard homomorphism from the group of rotations in $\mathbb{R}^3$, SO(3), and the group of single-qubit gates, which permits us to write every operator $U \in \mathrm{SU}(2)$ as

$$\begin{aligned} U\left(\hat{m}, \phi\right) &= \mathrm{e}^{\mathrm{i}\hat{m}\cdot\vec{\sigma}\,(\phi/2)} \\ &= \begin{pmatrix} \cos(\phi/2) + \mathrm{i}m_z \sin(\phi/2) & m_y \sin(\phi/2) + \mathrm{i}m_x \sin(\phi/2) \\ -m_y \sin(\phi/2) + \mathrm{i}m_x \sin(\phi/2) & \cos(\phi/2) - \mathrm{i}m_z \sin(\phi/2) \end{pmatrix}. \end{aligned} \tag{12}$$

$U(\hat{m}, \phi)$ represents a rotation, in SO(3), of an angle ϕ around the axes identified by the unitary vector $\hat{m}$. Therefore, if we exclude an overall phase, which is unimportant for the purpose of single-qubit gates, SU(2) can be mapped on a sphere of radius π: a point in the sphere defined by a radius $0 \leqslant \phi \leqslant \pi$ in the direction $\hat{m}$ corresponds to the rotation $U(\hat{m}, \phi)$. In the following, we will often address the elements of SU(2) not only as single-qubit gates but also as rotations, implicitly referring to this homomorphism.

The distance d (also referred to as error) between two gates (or their matrix representations) U and V is defined as the operator norm distance

$$d(U, V) \equiv \|U - V\| = \sup_{\|\psi\|=1} \|(U - V)\psi\|. \tag{13}$$

Thus, if we consider two operators $U = U(\hat{m}, \phi)$ and $V = U(\hat{n}, \theta)$, the distance between them is

$$d(U, V) = \sqrt{2 - 2\cos\frac{\phi}{2}\cos\frac{\theta}{2} - 2\,\hat{m}\cdot\hat{n}\,\sin\frac{\phi}{2}\sin\frac{\theta}{2}}, \tag{14}$$

which is bound above by $\sqrt{2}$. We note that the distance of a rotation $U(\hat{m}, \phi)$ from the identity operator is $d = 2\sin(\phi/4)$.

3.2. Brute-force search with Fibonacci anyons

First, we estimate the efficiency of the brute-force search algorithm through the probability distribution of the error of any weave of length L from a given target gate in SU(2). We assume that the collection of nontrivial weaves of length L, whose number is $N(L)$ as given in equation (11), is uniformly distributed on the sphere of SU(2). Representing the elements of SU(2) on the surface of the 4D sphere, we find the distribution $p_{\mathrm{BF}}(d)$ of their distance from the identity operator to be

$$p_{\mathrm{BF}}(d) = \frac{4}{\pi} d^2 \sqrt{1 - (d/2)^2}, \tag{15}$$

where $d = 2\sin(\phi/4)$.

Given this distribution, we obtain the average error for the brute-force search to be

$$\bar{d}(L) = \frac{\pi^{1/3}\Gamma(1/3)}{6^{2/3}[N(L)]^{1/3}} \approx 1.021\mathrm{e}^{-L/5.970} \tag{16}$$

asymptotically (see appendix for more details).

3.3. The icosahedral group

The hashing procedure relies on the possibility of exploiting the group structure of a finite subgroup of the target space to build sets $\mathcal{S}(L)$ of fine rotations, distributed with smaller and smaller mean distances from the identity, which can be used to progressively correct a first approximation of a target gate T. Therefore, it is fundamental to search, for every target space of unitary operators, a suitable subgroup to build the sets $\mathcal{S}$. Among the different finite subgroups of SU(2) we considered the SO(3) subgroups corresponding to the symmetry group of the icosahedron and of the cube, which have order 60 and 24, respectively. However, for practical purposes, we will refer in the following mainly to the icosahedral group because its

implementation of the hashing algorithm, as we will describe below, is more efficient in terms of the final braid length.

The icosahedral rotation group $\mathcal{I}$ is the largest polyhedral subgroup of SU(2), excluding reflection. For this reason, it has often been used to replace the full SU(2) group for practical purposes, as for example in earlier Monte Carlo studies of SU(2) lattice gauge theories [25], and its structure can be exploited to build meshes that cover the whole SU(2) [26].

$\mathcal{I}$ is composed of 60 rotations around the axes of symmetry of the icosahedron (platonic solid with 20 triangular faces) or of its dual polyhedron, the dodecahedron (regular solid with 12 pentagonal faces); there are six axes of the fifth order (corresponding to rotations with fixed vertices), ten of the third (corresponding to the triangular faces) and 15 of the second (corresponding to the edges). Let us for convenience write $\mathcal{I} = \{g_0, g_1, \ldots, g_{59}\}$, where $g_0 = e$ is the identity element. In figure 2(a), the elements of the icosahedral group are represented inside the SU(2) sphere.

Because of the group structure, given any product of n elements of a subgroup $g_{i_1} g_{i_2} \cdots g_{i_n}$ there always exists a group element $g_{i_{n+1}} = g_{i_n}^{-1} \cdots g_{i_2}^{-1} g_{i_1}^{-1}$ that is its inverse. In this way, $g_{i_1} g_{i_2} \cdots g_{i_n} g_{i_{n+1}} = e$ and one can find O^n different ways of obtaining the identity element, where O is the order of the group (60 in the case of the icosahedral one). This is the key property we will use in order to create a dense mesh $\mathcal{S}$ of fine rotations distributed around the identity operator in the target space. To achieve this goal, however, we need to break the exact group structure and to exploit the errors given by a brute-force search approximation of the elements of the chosen subgroup.

3.4. Pseudogroup

The main idea in the realization of the mesh $\mathcal{S}$ is that, representing the 60 elements of the subgroup $\mathcal{I}$ with weaves of a given length L, we can control, due to the relation (16), the average distance (or error) between the exact rotations in $\mathcal{I}$ and their braid representations that constitute the set $\tilde{\mathcal{I}}(L)$, which we will refer to as a *pseudogroup*.

Thanks to the homomorphism between SU(2) and SO(3), we associate with every rotation $g \in \mathcal{I}$ a 2×2 matrix (12). Then we apply a brute-force search of length L to approximate the 60 elements in $\mathcal{I}$; in this way we obtain 60 braids that give rise to the pseudogroup $\tilde{\mathcal{I}}(L) = \{\tilde{g}_0(L), \tilde{g}_1(L), \ldots, \tilde{g}_{59}(L)\}$. These braids are characterized by an average distance $\epsilon(L)$ with their corresponding elements $g_i \in \mathcal{I}$ given by equation (16). We note from figure 2 that the large errors for $L = 8$ completely spoil the symmetry of the group (and we will exploit this characteristic in the preprocessor of the hashing algorithm); however, increasing the length L, one obtains pseudogroups with smaller and smaller errors. Choosing, for instance, a fixed braid length of $L = 24$, the error of each braid representation to its corresponding exact matrix representation varies from 0.003 to 0.094 with a mean distance of 0.018.

Therefore, the braid representations of the icosahedral group with different lengths are obtained by a brute-force search once and for all. The so obtained braids are then stored for future utilization and this is the only step in which we apply, preliminarily, a brute-force search. Due to limiting computing resources, we construct the pseudogroup representations for the 60 rotations up to the length $L = 68$, which, as we will describe below, is sufficient to implement three iterations of the hashing algorithm. In principle, one could calculate the brute-force approximation of the group elements to larger lengths once and for all, in order to use them

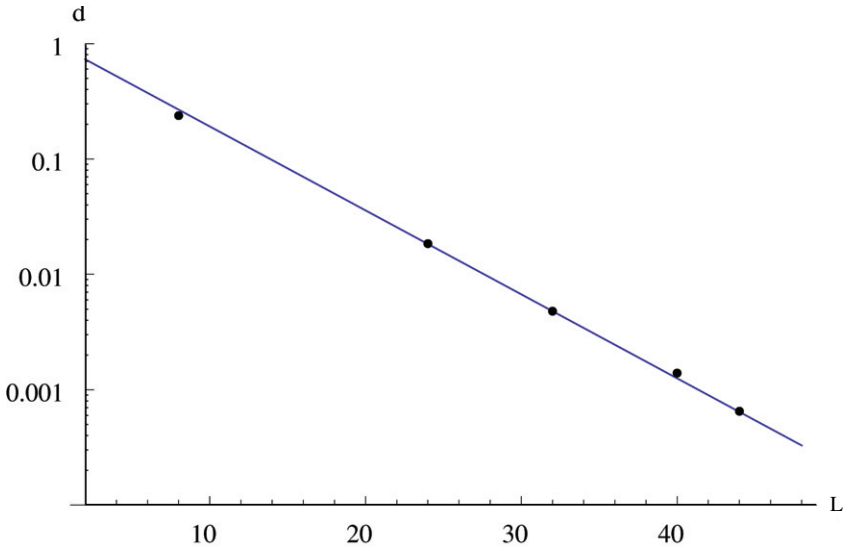

Figure 1. The average errors (dots) of the approximations to the 60 rotations of the icosahedral group with the brute-force search as a function of the length of the weaves used. These errors characterize the pseudogroups used in the hashing algorithm for the lengths $L = 8$, 24, 32, 40 and 44. The results are in very good agreement with the prediction in equation (16) (solid line).

for a greater number of iterations. The average errors characterizing the main pseudogroups we use in the iterations are shown in figure 1; they agree well with equation (16).

Let us stress that the 60 elements of $\tilde{\mathcal{I}}(L)$ (for any finite L) do not close the composition laws of the icosahedral group; in fact a pseudogroup $\tilde{\mathcal{G}}(L)$ becomes isomorphic to its corresponding group $\mathcal{G}$ only in the limit $L \to \infty$. If the composition law $g_i g_j = g_k$ holds in $\mathcal{I}$, the product of the corresponding elements $\tilde{g}_i(L)$ and $\tilde{g}_j(L)$ is not $\tilde{g}_k(L)$, although it can be very close to it for large enough L. Interestingly, the distance between the product $\tilde{g}_i(L)\,\tilde{g}_j(L)$ and the corresponding element g_k of $\mathcal{I}$ can be linked to the Wigner–Dyson distribution (see section 4.2).

Using the pseudogroup structure of $\tilde{\mathcal{I}}(L)$, it is easy to generate a mesh $\mathcal{S}(L)$ made of a large number of braids only in the vicinity of the identity matrix: this is a simple consequence of the original group algebra, in which the composition laws allow us to obtain the identity group element in various ways. The set $\mathcal{S}$ is instrumental in achieving an important goal, i.e. to search for, among the elements of $\mathcal{S}$, the best correction to apply to a previous approximation of the target single-qubit gate T we want to hash.

It is important to note that, changing the length L of the pseudogroup representation, we can control the average distance of the fine rotations in $\mathcal{S}$ from the identity. To correct an approximation of T with an error ε, we need a mesh $\mathcal{S}$ characterized by roughly the same average error in order to reach an optimal density of possible corrections and thus increase the efficiency of the algorithm. Therefore, knowing the average error of the distribution of the approximations we want to improve, we can choose a suitable L to generate a mesh. As represented in figure 3, this allows us to define a series of denser and denser meshes to iterate the hashing algorithm in order to correct at each step the expected mean error, which we have determined by analyzing the distributions of errors of 10 000 random targets after the corresponding iteration.

To create the mesh of fine rotations, labeled by $\mathcal{S}(L,n)$, we consider all the possible ordered products $\tilde{g}_{i_1}(L)\tilde{g}_{i_2}(L)\ldots\tilde{g}_{i_n}(L)$ of fixed $n\geqslant 2$ elements of $\tilde{\mathcal{I}}(L)$ of length L and multiply them by the matrix $\tilde{g}_{i_{n+1}}(L)\in\tilde{\mathcal{I}}(L)$ such that $g_{i_{n+1}}=g_{i_n}^{-1}\ldots g_{i_2}^{-1}g_{i_1}^{-1}$. In this way we generate all the possible combinations of $n+1$ elements of $\mathcal{I}$ that produce the identity but, thanks to the errors that characterize the braid representation $\tilde{\mathcal{I}}$, we obtain 60^n small rotations in SU(2), corresponding to braids of length $(n+1)L$. In section 4.2, we will describe their distribution around the identity with the help of random matrices.

4. Iterative pseudogroup hashing in SU(2)

4.1. The iterative pseudogroup hashing algorithm

The hashing algorithm [27] is based on the possibility of finding progressive corrections to minimize the error between the target gate $T\in\mathrm{SU}(2)$ and the braids that represent it. These corrections are chosen among the meshes $\mathcal{S}(L_i,3)$ whose distribution around the identity operator is shown in figure 3.

The algorithm consists of a first building block, called *preprocessor*, whose aim is to give an initial approximation $\tilde{T}_0$ of T, and a *main processor* composed of a series of iterations of the hashing procedure that, at each step, extend the previous representation by a braid in $\mathcal{S}(L_i,3)$. The final braid has the form

$$\tilde{T}_3=\underbrace{\tilde{g}_{j_1}(L_0)\cdots\tilde{g}_{j_3}(L_0)}_{\text{Preprocessor}}\underbrace{\tilde{g}_{p_1}(L_1)\cdots\tilde{g}_{p_4}(L_1)}_{\text{1st iteration}}\underbrace{\tilde{g}_{q_1}(L_2)\cdots\tilde{g}_{q_4}(L_2)}_{\text{2nd iteration}}\underbrace{\tilde{g}_{r_1}(L_3)\cdots\tilde{g}_{r_4}(L_3)}_{\text{3rd iteration}}. \tag{17}$$

Each $\tilde{g}_j(L_i)$ is an element of the pseudogroup $\tilde{\mathcal{I}}(L_i)$ and, as explained in the previous section, the braid segment in each main iteration is constrained by $g_{k_4}=g_{k_3}^{-1}g_{k_2}^{-1}g_{k_1}^{-1}$, $k=p,q$ or r.

Each iteration starts from an input approximation $\tilde{T}_{i-1}$ with a distance ε_{i-1} from the target T. We exploit the elements of the mesh $\mathcal{S}(L_i,n)$ to generate a new braid $\tilde{T}_i$ with a smaller distance ε_i. The lengths L_i in equation (17), which characterize the pseudogroups used in the main processor, control the density of the corresponding meshes and are chosen among the sets of stored pseudogroups in order to correct the residual error in an efficient way (see section 5).

Let us analyze now the details of each step in the hashing algorithm. The preprocessor is a fast procedure to generate a rough approximation of the target gate $T\in\mathrm{SU}(2)$ and, in general, it associates with every T a braid that is an element of $[\tilde{\mathcal{I}}(L_0)]^m$ (of length mL_0). Therefore, the preprocessor approximates T with the ordered product of elements in the icosahedral pseudogroup $\tilde{\mathcal{I}}(L_0)$ that best represents it, minimizing their distance. Thus we obtain a starting braid

$$\tilde{T}_0^{L_0,m}=\tilde{g}_{j_1}(L_0)\,\tilde{g}_{j_2}(L_0)\ldots\tilde{g}_{j_m}(L_0) \tag{18}$$

with an initial error to reduce. The preprocessor procedure relies on the fact that, choosing a small L_0, we obtain a substantial discrepancy between the elements g_i of the icosahedral group and their representatives $\tilde{g}_i$, as shown in figure 2. Because of these seemingly random errors, the set $[\tilde{\mathcal{I}}(L_0)]^m$ of all the products $\tilde{g}_{j_1}\tilde{g}_{j_2}\ldots\tilde{g}_{j_m}$ is well spread all over SU(2) and can be thought of as a random discretization of the group. In particular, we find that the pseudogroup $\tilde{\mathcal{I}}(8)$ has an average error of about 0.24 and it is sufficient to take $m=3$ (as we did in equation (17)) to cover the whole SU(2) in an efficient way with 60^3 operators. The average error for an arbitrary single-qubit gate with its nearest element $\tilde{T}_0^{8,3}\in[\tilde{\mathcal{I}}(8)]^3$ is about 0.027.

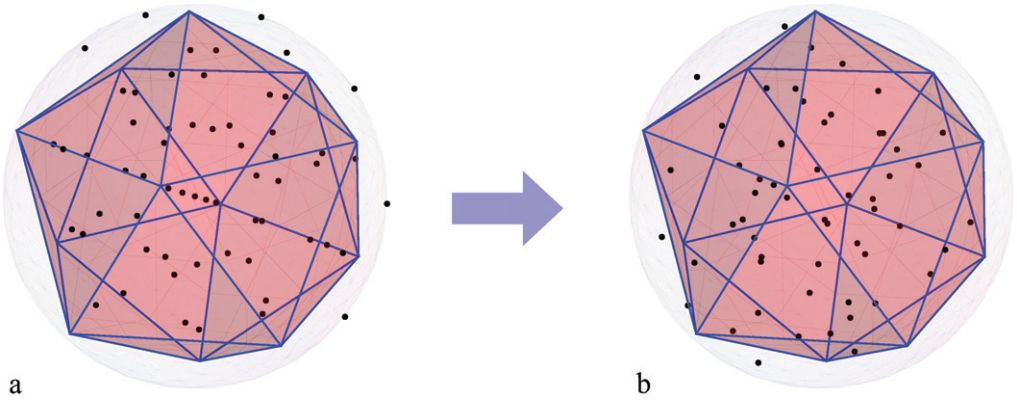

Figure 2. (a) The icosahedral group representation inside the SU(2) sphere. (b) the pseudogroup representation $\tilde{\mathcal{I}}(8)$ used in the preprocessor. Due to the large errors (~ 0.24) the elements of $\tilde{\mathcal{I}}(8)$ seem to span the SU(2) sphere randomly.

One can then apply the main processor to the first approximation $\tilde{T}_0$ of the target gate. Each subsequent iteration improves the previous braid representation of T by adding a finer rotation to correct the discrepancy with the target and generate a new braid. In the first iteration, we use the mesh $\mathcal{S}(L_1, n)$ to efficiently reduce the error in $\tilde{T}_0^{l,m}$. Multiplying $\tilde{T}_0^{l,m}$ by all the elements of $\mathcal{S}(L_1, n)$, we generate 60^n (O^n if we use a subgroup of order O) possible braid representations of T:

$$\tilde{T}_0^{l,m}\tilde{g}_{i_1}\tilde{g}_{i_2}\ldots\tilde{g}_{i_n}\tilde{g}_{i_{n+1}}. \tag{19}$$

Among these braids of length $(n+1)L_1 + mL_0$, we search for the one with the shortest distance to the target gate T. This braid, $\tilde{T}_1\,(L_0, m, L_1, n)$, is the result of the first iteration in the main processor.

The choice of $L = 24$ for the first step is dictated by the analysis of the mean error of the preprocessor (~ 0.03) that requires, as we will see in the following section, a pseudogroup with compatible error for an efficient correction. An example of the first iteration is illustrated in [6].

With $\tilde{T}_1$ we can then apply the second and third iterations of the main processor to obtain an output braid of the form in equation (17). These iterations further reduce the residual discrepancy in decreasing error scales. Each step in the main processor requires the same runtime, during which a search within 60^n braids selects the one with the shortest distance to T. One must choose appropriate pseudogroups with longer braid lengths L_2 and L_3 to generate finer meshes. As for L_1, we choose L_2 and L_3 to match the error of the corresponding pseudogroup to the respective mean residual error. In practice, we choose $L_2 = 44$ and $L_3 = 68$. The final output assumes the form in equation (17) and the average distance to the target braid (in 10 000 random tests) is 2.29×10^{-5} after the second iteration and 8.24×10^{-7} after the third. Without reduction the final length is 568; however, due to shortenings at the junctions where different braid segments meet, the practical final length of the weave is usually smaller.

4.2. Connection with random matrix theory and reduction factor for the main processor

To analyze the efficiency of the main processor we must study the random nature of the meshes $\mathcal{S}(L, n)$. The distribution of the distance between the identity and their elements has

an intriguing connection to the Gaussian unitary ensemble of random matrices, which helps us to understand how close we can approach the identity in this way and therefore what the optimal choice of the lengths of the pseudogroups is for each iteration of the main processor.

Let us analyze the group property deviation for the pseudogroup $\tilde{\mathcal{I}}(L)$ for braids of length L. One can write $\tilde{g}_i = g_i e^{i\Delta_i}$, where Δ_i is a Hermitian matrix, indicating the small deviation of the finite braid representation from the corresponding SU(2) representation for an individual element. For a product of $\tilde{g}_i$ that approximate $g_i g_j \cdots g_{n+1} = e$, one has

$$\tilde{g}_i \tilde{g}_j \cdots \tilde{g}_{n+1} = g_i e^{i\Delta_i} g_j e^{j\Delta_j} \cdots g_{n+1} e^{i\Delta_{n+1}} = e^{iH_n}, \tag{20}$$

where H_n, related to the accumulated deviation, is

$$H_n = g_i \Delta_i g_i^{-1} + g_i g_j \Delta_j g_j^{-1} g_i^{-1} + \cdots + g_i g_j \cdots g_n \Delta_n g_n^{-1} \cdots g_j^{-1} g_i^{-1} + \Delta_{n+1} + O(\Delta^2). \tag{21}$$

The natural conjecture is that, for a long enough sequence of matrix products, the Hermitian matrix H_n tends to a random matrix corresponding to the Gaussian unitary ensemble. This is plausible as H_n is a Hermitian matrix that is the sum of random initial deviation matrices with random unitary transformations. A direct consequence is that the distribution of the eigenvalue spacing s obeys the Wigner–Dyson form [28]

$$P(s) = \frac{32}{\pi^2 s_0} \left(\frac{s}{s_0}\right)^2 e^{-(4/\pi)(s/s_0)^2}, \tag{22}$$

where s_0 is the mean level spacing. For small enough deviations, the distance of H_n to the identity, $d(1, e^{iH_n}) = \|H_n\| + O(\|H_n\|^3)$, is proportional to the eigenvalue spacing of H and, therefore, should obey the same Wigner–Dyson distribution. The conjecture above is indeed well supported by our numerical analysis, even for n as small as 3 or 4: the distances characterizing the meshes with $L = 24$ and $L = 48$ obtained from the corresponding pseudogroups (figure 3) follow the unitary Wigner–Dyson distribution.

The elements of the meshes $\mathcal{S}(L, n)$ are of the form in equation (20) and this implies that, once we choose a pseudogroup $\tilde{\mathcal{I}}(L)$ whose average error $\bar{d}(L)$ is given by equation (16), the mean distance s_0 in (22) of the corresponding mesh from the identity is $s_0(L) \approx \sqrt{n+1}\,\bar{d}(L)$ as resulting from the sum of $n+1$ Gaussian terms.

At each iteration of the main processor, we increase the braid by $(n+1) = 4$ braid segments with length L_i. By doing that, we create 60^n braids and the main processor searches, among them, the best approximation of the target. Therefore, the runtime is linear in the dimension of the meshes used and in the number of iterations. The unitary random matrix distribution implies that the mean deviation of the four-segment braids from the identity (or any other in its vicinity) is a factor of $\sqrt{n+1}$ times larger than that of an individual segment. Considering the 3D nature of the unitary matrix space, we find that at each iteration the error (of the final braid to the target gate) is reduced, on average, by a factor of $f \sim 60^{n/3}/\sqrt{n+1} = 30$, where 60 is the number of elements in the icosahedral group. This has been confirmed in the numerical implementation.

4.3. Hashing with the cubic group

For comparison, we also implemented the hashing procedure with the smaller cubic group. In this case the rotations in the group of the cube are 24; thus we choose $n = 4$ to generate a comparable number of elements in each mesh $\mathcal{S}(L, n)$. Our implementation of the hashing with the cubic group uses a preprocessor with $L_0 = 8$ and $m = 4$ and a main processor with $L_1 = 24$ and $n = 4$. Approximating over 10 000 random targets, we obtained an average error

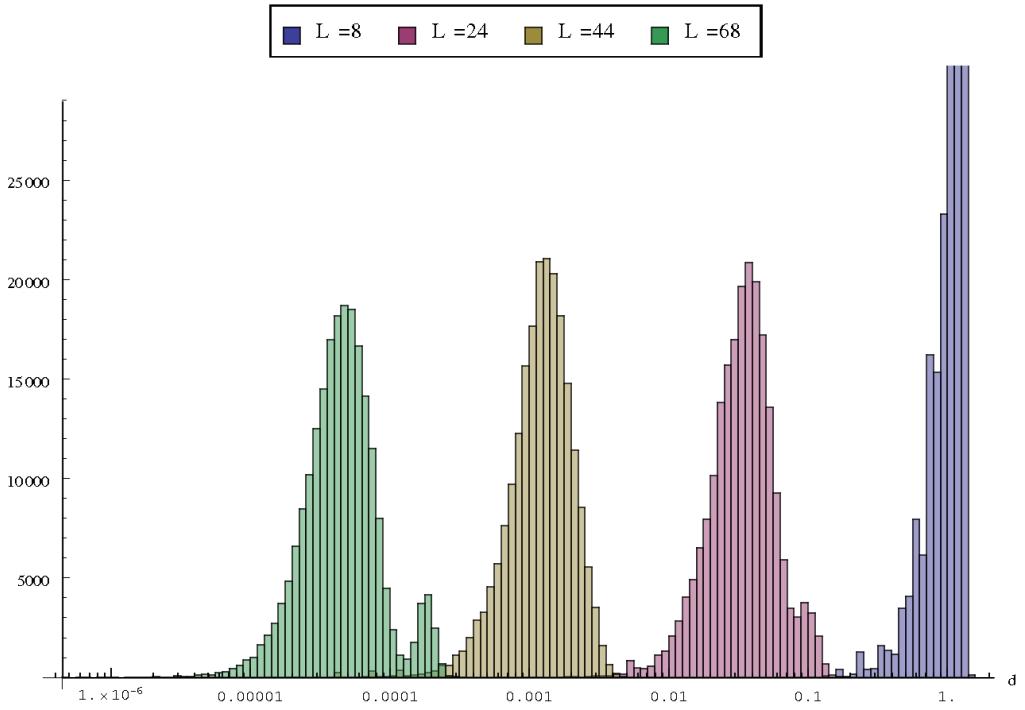

Figure 3. Distribution of the meshes $\mathcal{S}(L, 3)$ used by the renormalization scheme of the hashing algorithm. The distance d from the identity is represented in logarithmic scale. The mesh with $L = 8$ comprehends every possible product of three elements in $\tilde{\mathcal{I}}(8)$ and, therefore, it spans the whole SU(2) space up to the distance $\sqrt{2}$. The meshes $\mathcal{S}(24, 3)$, $\mathcal{S}(44, 3)$ and $\mathcal{S}(68, 3)$, instead, are the product of four braids whose corresponding rotations are combined to approximate the identity. $\mathcal{S}(24, 3)$ and $\mathcal{S}(44, 3)$ follow the Wigner–Dyson distribution, while $\mathcal{S}(68, 3)$ exhibits a second local maximum due to the incomplete brute-force search we used to obtain $\tilde{\mathcal{I}}(68)$.

7.09×10^{-4} after the main processor, comparable with 7.24×10^{-4} after the first iteration in the previous icosahedral group implementation. This result is consistent with the new reduction factor $f_{\text{cube}} = 24^{4/3}/\sqrt{5} \approx 30.96$. However, we note that the cubic hashing is less efficient both in terms of the braid length (because it requires $n = 4$ instead of $n = 3$) and in terms of the runtime (because the time required for the searching algorithm is linear in the elements of $\mathcal{S}$ and $24^4 > 60^3$).

4.4. Tail correction

The choice of the proper L_i is important. We determine them by the average error before each iteration. If a certain L_i is too large, it generates a mesh around the identity that may be too small to correct the error relatively far from the identity, where the mesh is very fine. On the other hand, if L_i is too small, the mesh may be too sparse to correct efficiently. The former situation occurs exactly when we treat the braids with errors significantly larger than the average; they correspond to the rare events in the tail of the distributions as shown in figure 4. Such an already

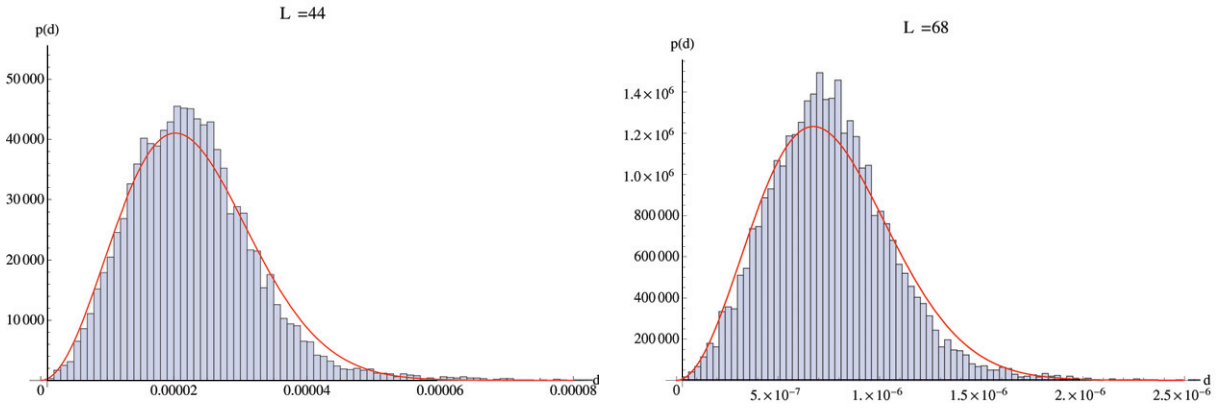

Figure 4. Probability densities of the distance d from the target of 10 000 random tests after the second ($L = 44$) and third ($L = 68$) main processors. The trend agrees with the unitary Wigner–Dyson distribution with average errors 2.28×10^{-5} and 7.60×10^{-7}, respectively.

Table 1. Average error and standard deviation for the hashing algorithm after the preprocessor and the three iterations of the main processor. The output in the absence or presence of a tail correction for the second and third iterations is shown (the asterisks indicate that the preprocessor and the first iteration are not affected by the tail correction). This correction is based on the pseudogroups with length 40 and 64 instead of the standard ones, 44 and 68. Only about 0.6% of the targets used the tail correction in the second iteration, but note that the results, both in terms of average error and in terms of standard deviation σ, are extremely affected by these rare events.

	Without tail correction		With tail correction	
10 000 trials	Average error	σ	Average error	σ
Preprocessor*	0.027	0.010	0.027	0.010
Main, first iteration*	7.24×10^{-4}	3.36×10^{-4}	7.24×10^{-4}	3.36×10^{-4}
Main, second iteration	2.29×10^{-5}	1.3×10^{-5}	2.28×10^{-5}	9.79×10^{-6}
Main, third iteration	8.24×10^{-7}	5.6×10^{-6}	7.60×10^{-7}	3.27×10^{-7}

large error is then amplified with the fixed selection of $L_i = 24$, 44 and 68. To avoid this, one can correct the 'rare' braids $\tilde{T}_{i-1}$, whose error is higher than a certain threshold, with a broader mesh (e.g. $\mathcal{S}(L_i - 4, n)$).

The tail correction is very efficient. If we correct for the 0.6% of the targets with the largest errors in the second iteration with $\mathcal{S}(40, 3)$ instead of $\mathcal{S}(44, 3)$, we reduce the average error by about 8% after the third iteration (see table 1). The drastic improvement is due to the fact that once a braid is not properly corrected in the second iteration, the third one becomes ineffective. We can illustrate this situation with the example of the operator iY (where Y is the Pauli matrix): without tail correction it is approximated in the first iteration with an error of 0.0039, which is more than 5 times the average error expected. After the second iteration, we obtain an error of 4.3×10^{-4} (almost 20 times higher than the average value) and after the third the error becomes

2.0×10^{-4} (more than 200 times the mean value). If we use $\mathcal{S}(40, 3)$ instead of $\mathcal{S}(44, 3)$ in the second iteration we obtain an error 4.46×10^{-5}, with a shorter braid than before, and a final error of 1.31×10^{-6}, which is less than twice the average error.

5. General efficiency of the algorithm

To evaluate the efficiency of the hashing algorithm it is useful to calculate the behavior of the maximum length of the braids and of the runtime with respect to the average error obtained. We compare the results with those of the brute-force search (which gives the optimal braid length) and of the Solovay–Kitaev algorithm.

As described in section 4.2, we reduce the average error at the ith iteration to

$$\varepsilon_i \sim \varepsilon_{i-1}/f \quad \text{with } f \approx 30. \tag{23}$$

The total number of iterations (or *depth*) to achieve a final error of ε is then

$$M \sim \frac{\ln(1/\varepsilon)}{\ln f}, \tag{24}$$

so the expected error after each iteration is

$$\ln(1/\varepsilon_i) \sim \mathrm{i} \ln f. \tag{25}$$

For efficient optimization, we choose the length L_i of the braid segments at the ith iteration to approximate the corresponding icosahedral group elements with an average error of ε_{i-1} (see discussions in section 4.4). So we have, from equation (16),

$$L_i \sim \mathcal{L} \ln(1/\varepsilon_{i-1}) \tag{26}$$

with $\mathcal{L} \approx 6$ (see equation (16)), and the length of the braid we construct increases by $(n+1)L_i = 4L_i$ at the ith iteration, i.e.

$$L_i - L_{i-1} = 4\mathcal{L} \ln(1/\varepsilon_{i-1}) \sim 4\mathcal{L}(i-1) \ln f. \tag{27}$$

Thus the total length of the braid with an error of ε is

$$L_M \sim \sum_{i=1}^{M} 4\mathcal{L}(i-1) \ln f \sim M^2. \tag{28}$$

The final results for the hashing algorithm are, therefore,

$$L_{\mathrm{qh}} \sim (\ln(1/\varepsilon))^2, \tag{29}$$

$$T_{\mathrm{qh}} \sim M \sim \ln(1/\varepsilon). \tag{30}$$

We have explicitly confirmed that we can realize the perfect length–error scaling as shown in figure 5 up to three iterations in the main processor.

We can conclude that while no method beats the brute-force search in length, we achieve a respectable gain in time. Comparing these results with the length of the braids obtained by the Solovay–Kitaev algorithm in equation (3) and with its runtime in equation (4), the hashing algorithm gives results that are better than the Solovay–Kitaev results in terms of length and significantly better in terms of time.

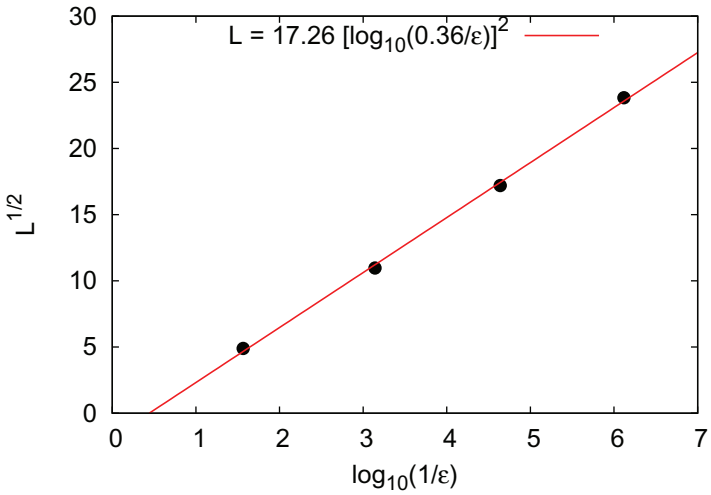

Figure 5. The scaling performance of the hashing algorithm in terms of the square root of the maximum length versus the logarithm of the inverse error. The error is the average error in approximating 10 000 random targets after the preprocessor and the three iterations in the main processor. The results agree with the expected behavior (equations (29) and (30)).

6. Conclusions

We have demonstrated, for a generic universal topological quantum computer, that the iterative pseudogroup hashing algorithm allows an efficient search for a braid sequence that approximates an arbitrary given single-qubit gate. This can be generalized to the search for two-qubit gates as well. The algorithm applies to the quantum gate construction in a conventional quantum computer given a small universal gate set or any other problems that involve realizing an arbitrary unitary rotation approximately by a sequence of 'elementary' rotations.

The algorithm uses a set of pseudogroups based on the icosahedral group or other finite subgroups of SU(2), whose multiplication tables help generate, in a controllable fashion, smaller and smaller unitary rotations, which gradually (exponentially) reduce the distance between the result and the target. The iteration is in the same spirit as a generic renormalization group approach, which guarantees that the runtime of the algorithm is proportional to the logarithm of the inverse average final error $1/\varepsilon$; the total length of the braid is instead quadratic in $\ln(1/\varepsilon)$, and both the results are better than the Solovay–Kitaev algorithm introduced in textbooks.

We have explicitly demonstrated that the result from the performance analysis is in excellent agreement with that from our computer simulation. We also showed that the residual error distributes according to the Wigner–Dyson unitary ensemble of random matrices. The connection of the error distribution with random matrix theory ensures that we can efficiently carry out the algorithm and improve the rare cases in the distribution tail.

The overhead of the algorithm is that we need to prepare several sets of braid representations of the finite subgroup elements. Obtaining the longer representation can be time-consuming, but fortunately, we need to compute them only once and use the same sets of representations for all future searches.

Acknowledgments

This work was supported by INSTANS (from ESF), 2007JHLPEZ (from MIUR) and the 973 Program under project no. 2009CB929100. XW acknowledges the Max Planck Society and the Korea Ministry of Education, Science and Technology for the support of the Independent Junior Research Group at APCTP. MB and GM are grateful to APCTP for the hospitality in Pohang where part of this study was carried out.

Appendix. Distribution of the best approximation in a given set of braids

In this appendix, we derive the distribution of the approximation to a gate in a given set of N braids in the vicinity of the identity. While it is sufficient for the discussions in the main text, this derivation can be generalized to the more generic cases. Let us assume that the targeted gate is $g_0 = U(\hat{l}, \phi_0)$ as defined in equation (12). The distance between g_0 and the identity is $s_0 = 2\sin(\phi_0/4) \approx \phi_0/2$ for small ϕ_0. We then search for, in a given set of braids either with a distribution as given in equation (15) or from a generated random matrix distribution as discussed in section 4.2, the one with the shortest distance to the target.

We consider an arbitrary braid with representation $g = U(\hat{m}, \phi)$ in a collection with a distribution $p(s)$ of the distance to the identity $s = 2\sin(\phi/4)$. We define

$$P(x) = \int_0^x p(s)\,\mathrm{d}s, \tag{A.1}$$

which is the probability of having a distance less than $x \leqslant \sqrt{2}$. Obviously $P(x)$ is a monotonically increasing function bound by $P(0) = 0$ and $P(\sqrt{2}) = 1$. The distance $d(g_0, g)$ between g and g_0 is the same as that between $g_0^{-1}g$ and the identity. To the first order in ϕ and ϕ_0 (as we assume that all braids/gates are in the vicinity of the identity braid), we have, from equation (14),

$$d(g_0, g) \approx \sqrt{s^2 + s_0^2 - 2(\hat{l}\cdot\hat{m})s_0\, s} = \| s\hat{m} - s_0\hat{l} \|, \tag{A.2}$$

which is bound by $s + s_0$ and $|s - s_0|$. We can see that the chance of finding a braid that is close to g_0 is large when $p(s)$ peaks around s_0. If we denote the probability of having no braids within a distance of t by $Q(t)$, the probability of having the braid with the shortest distance between t and $t + \mathrm{d}t$ is

$$Q(t) - Q(t + \mathrm{d}t) = Q(t)\left\langle N\mathrm{d}t \int_0^{\sqrt{2}} p(s)\,\mathrm{d}s\,\delta\left(t - d(g_0, g)\right)\right\rangle, \tag{A.3}$$

where the angular brackets imply the angular average of the average number of braids with a distance between t and $t + \mathrm{d}t$. Therefore,

$$-\frac{\mathrm{d}\ln Q(t)}{\mathrm{d}t} = N\left\langle \int_0^{\sqrt{2}} p(s)\,\mathrm{d}s\,\delta\left(t - d(g_0, g)\right)\right\rangle. \tag{A.4}$$

As an example, we consider $s_0 = 0$ (i.e. with the full SU(2) rotation symmetry of the distribution), in which case

$$-\ln Q(t) = N\int_0^t p(s)\,\mathrm{d}s = NP(t), \tag{A.5}$$

or $Q(t) = \exp[-NP(t)]$. $NP(t)$ is the expected number of braids with a distance to the identity less than t. The differential probability of having the braid with the nearest distance between t and $t + \mathrm{d}t$ is, therefore,

$$q(t) \equiv -\frac{\mathrm{d}Q(t)}{\mathrm{d}t} = N\frac{\mathrm{d}P(t)}{\mathrm{d}t}\mathrm{e}^{-NP(t)} = Np(t)\mathrm{e}^{-NP(t)}. \tag{A.6}$$

Combining the results with equation (15), we estimate for the brute-force search

$$q_{\mathrm{BF}}(t; L) = N(L)p_{\mathrm{BF}}(t)\mathrm{e}^{-N(L)P_{\mathrm{BF}}(t)} \tag{A.7}$$

with an average distance

$$\bar{d}(L) = \frac{\pi^{1/3}\left[\Gamma\left(\frac{1}{3}\right) - \Gamma\left(\frac{1}{3}, \frac{8\sqrt{2}N(L)}{3\pi}\right)\right]}{6^{2/3}[N(L)]^{1/3}}, \tag{A.8}$$

where $\Gamma(a, x)$ is the incomplete gamma function

$$\Gamma(a, x) = \int_x^\infty \mathrm{d}t t^{a-1}\,\mathrm{e}^{-t}. \tag{A.9}$$

In the large L limit, $\bar{d}(L) \approx 1.021\mathrm{e}^{-L/5.970}$. This is the result we quoted in equation (16).

References

[1] Nielsen M A and Chuang I L 2000 *Quantum Computation and Quantum Information* (Cambridge: Cambridge University Press) chapter 4
[2] DiVincenzo D P 1995 *Phys. Rev.* A **51** 1015
[3] Kitaev Y A, Shen A H and Vyalyi M N 2002 *Classical and Quantum Computation* (Providence, RI: American Mathematical Society) section 8
[4] Dawson C M and Nielsen M A 2006 *Quantum Inf. Comput.* **6** 81
[5] Hormozi L *et al* 2007 *Phys. Rev.* B **75** 165310
[6] Burrello M *et al* 2010 *Phys. Rev. Lett.* **104** 160502
[7] Kitaev Y A 2003 *Ann. Phys.* **303** 2
[8] Freedman M, Larsen M and Wang Z 2002 *Commun. Math. Phys.* **227** 605
Freedman M, Larsen M and Wang Z 2002 *Commun. Math. Phys.* **228** 177
Freedman M, Kitaev A and Wang Z 2002 *Commun. Math. Phys.* **227** 587
[9] Nayak C *et al* 2008 *Rev. Mod. Phys.* **80** 1083
[10] Brennen G K and Pachos J K 2008 *Proc. R. Soc.* A **464** 1
[11] Preskill J *Lecture Notes on Topological Quantum Computation* Available online at www.theory.caltech.edu/~preskill/ph219/topological.pdf
[12] Stern A 2008 *Ann. Phys.* **323** 1
[13] Cooper N R, Wilkin N K and Gunn J M F 2001 *Phys. Rev. Lett.* **87** 120405
[14] Read N and Green D 2000 *Phys. Rev.* B **61** 10267
[15] Bonderson P *et al* 2010 arXiv:1003.2856
[16] Bonesteel N E *et al* 2005 *Phys. Rev. Lett.* **95** 140503
[17] Simon S H *et al* 2006 *Phys. Rev. Lett.* **96** 070503
[18] Xu H and Wan X 2008 *Phys. Rev.* A **78** 042325
[19] Baraban M, Bonesteel N E and Simon S H 2010 *Phys. Rev.* A **81** 062317
[20] Read N and Rezayi E H 1999 *Phys. Rev.* B **59** 8084
[21] Xia J S *et al* 2004 *Phys. Rev. Lett.* **93** 176809

[22] Ardonne E and Schoutens K 1999 *Phys. Rev. Lett.* **82** 5096
Ardonne E *et al* 2001 *Nucl. Phys.* B **607** 549
Ardonne E and Schoutens K 2007 *Ann. Phys.* **322** 201
[23] Kitaev Y A 2006 *Ann. Phys.* **321** 2
[24] Xu H and Wan X 2009 *Phys. Rev.* A **80** 012306
[25] Rebbi C 1980 *Phys. Rev.* D **21** 3350
[26] Mosseri R 2008 *J. Phys. A: Math. Theor.* **41** 175302
[27] Object-oriented source codes available for download at http://sites.google.com/site/braidanyons/
[28] Mehta M 1991 *Random Matrices* 2nd edn (San Diego, CA: Academic)