Optimization of the Solovay-Kitaev algorithm

Tien Trung Pham, Rodney Van Meter, and Dominic Horsman
Faculty of Environment and Information Studies, Keio University, Shonan Fujisawa Campus, 5322 Endo, Fujisawa, Kanagawa, Japan
(Received 23 September 2012; revised manuscript received 8 March 2013; published 30 May 2013)

The Solovay-Kitaev algorithm is the standard method used for approximating arbitrary single-qubit gates for fault-tolerant quantum computation. In this paper we introduce a technique called search space expansion, which modifies the initial stage of the Solovay-Kitaev algorithm, increasing the length of the possible approximating sequences but without requiring an exhaustive search over all possible sequences. This technique is combined with an efficient space search method called geometric nearest-neighbor access trees, modified for the unitary matrix lookup problem, in order to reduce significantly the algorithm run time. We show that, with low time cost, our techniques output gate sequences that are almost an order of magnitude smaller for the same level of accuracy. This therefore reduces the error correction requirements for quantum algorithms on encoded fault-tolerant hardware.

DOI: 10.1103/PhysRevA.87.052332 PACS number(s): 03.67.Ac, 03.67.Lx, 03.67.Pp

I. INTRODUCTION

The biggest challenge to building working quantum computers is the problem of fault tolerance [1]. Unless a quantum computer is specifically designed and built to withstand the effects of errors from environmental decoherence and inaccurate hardware operations, it will not be able to perform computations of any significant size [2]. One prominent way to control errors is to correct them when they arise by encoding quantum data in many physical qubits. If the underlying hardware is accurate to "threshold" value, then these error correction codes can, in principle, keep an arbitrary computation error free given enough physical resources [3–5].

One of the drawbacks to error correction codes, however, is that only a very small number of logical gates are available to the code. To implement a given algorithm, the gates in the algorithm must be decomposed into the "library" gate set available to the code. In general this cannot be performed exactly: we must look for sequences of library gates that *approximate* the gates we require [6].

When finding an approximation to an arbitrary gate, in the general case, the longer the sequence of library gates (and their inverses) is, the better the approximation to the gate in question that can be found is. However, high-accuracy exhaustive searches become untenable on current computational technology. The key problem in this issue is the exponential increase of the size of the space over which it is necessary to search in order to find a good approximation. The Solovay-Kitaev approximation algorithm was introduced to get around this difficulty [6,7]. This recursive algorithm performs an exhaustive search only over the space of sequences of length up to l_0, finding the sequence with the smallest distance (defined by the trace norm) from the gate we wish to approximate. The residual difference between the actual and approximate gates is then sent to the next level of the algorithm, where it is further approximated. At each level the length of the gate sequences grows by a factor of 5. The algorithm terminates at a "good" approximation, where the distance to the actual gate is less than a chosen constant ϵ.

While very powerful, the Solovay-Kitaev algorithm suffers from a serious weakness. While it will always find a good approximation for any value of ϵ, the search covers only a very sparse region of the entire space of possible approximation sequences. As a consequence, the output from the approximation is almost always far longer than it needs to be. This is an extreme disadvantage for fault-tolerant computation, as this greatly increases the logical depth of the computation, thus increasing the amount of error correction required, which in turn increases the physical size and run time of the algorithm. Reducing these requirements as far as possible is key to implementing realistic fault-tolerant quantum computation.

Currently, there are two well-known alternatives to the original Solovay-Kitaev algorithm. The first is an exhaustive-search algorithm that can give a very efficient library-gate decomposition. Efficient sequences, however, are only achievable in exceptional cases, when there is a short gate sequence that is a very accurate approximation to the query gate, or where there are specific mathematical properties of gate sequences of which we can take advantage [8]. However, in the general case the scaling is exponential (as in a standard exhaustive search) and therefore limited by computational ability. The second alternative to the standard algorithm is the "phase-kickback" method originally given in [7]. This can produce shorter gate sequences by using special ancilla states but requires many more qubits for the ancilla and is also restricted in the library gate set it can use.

In this paper we describe an alternative gate decomposition algorithm that modifies the original Solovay-Kitaev method and allows a much denser search of the space of sequences. The key to improving the algorithm is to concentrate on the initial approximation, and we dramatically increase the search space by combining two sequences out of the database of initial sequences. This expansion technique can be implemented recursively, creating an even better initial approximation. To make this search step computationally tractable, we have applied a geometric nearest-neighbor access tree search procedure (GNAT) [9]. Two other data structures, *R* trees [10] and *k-d* trees [11], were also tried but were found to be ineffective for this problem. The GNAT-based procedure is a fully general method for any gate and any library gate set and with high probability produces approximations that are significantly shorter than those given by the original algorithm for the same precision.

This paper is structured as follows. Section II reviews the Solovay-Kitaev theorem, along with the original implementing algorithm proposed in [6,7]. In Sec. III we introduce the search space expansion technique (SSE) that we will use to supplement the original algorithm. We show that by splitting each candidate sequence and then searching points near the subsequences, we can generate candidate sequences that with high probability are a better approximation but are shorter than those at the next level of recursion. One potential issue with this approach is an added classical search requirement over the sampled sequences, and in Sec. IV we describe a way of efficiently searching these sequences to reduce the time required compared with a standard exhaustive search. In Sec. V we compare the original Solovay-Kitaev algorithm with our modified algorithm for a set of 25 single-qubit unitaries chosen from a uniform distribution of the specifying vector. On average, the length of the candidate sequence reduces by a factor of 7, and for some sequences it can reduce by an order of magnitude. The number of levels of recursion in the algorithm is also greatly reduced, with generally only two or at most three levels being needed even for very high accuracies.

II. THE SOLOVAY-KITAEV APPROXIMATION

The Solovay-Kitaev theorem tells us we can always approximate a single-qubit gate G to arbitrary accuracy ϵ with a finite sequence of gates from a universal "library set" of gates $\{L_1, L_2, \ldots, L_N\}$ and their adjoints. The Solovay-Kitaev algorithm gives a method of finding what these sequences are for a given G, ϵ, and $\{L_i\}$.

The Solovay-Kitaev theorem is invoked as part of the compilation process for fault-tolerant quantum computing. The library set on a fault-tolerant computer is, in general, very limited, with only a handful of gates able to operate within the code space. For example, in the surface code we have access only to a small library: $\{\text{CNOT}, X, Z, H, S, S^\dagger, T, T^\dagger\}$ (where S is the phase gate and T is a Z rotation of $\pi/4$) [12]. In general, a library contains a two-qubit entangling gate such as controlled NOT (CNOT) or controlled phase and some single-qubit gates. Any multiqubit gate can be decomposed exactly into a combination of a two-qubit maximally entangling gate plus arbitrary single-qubit rotations [13], so this becomes the first stage of compilation for an algorithm. The next stage is then the further task of decomposing these arbitrary single qubit gates into the single qubit gates of the error correction code library. For our purposes in this paper, we ignore the X and Z gates, which are easily compensated for classically or built using H and S.

The Solovay-Kitaev theorem states that, for a given gate G, accuracy ϵ, and library gate set $\{L_i\}$, there always exists a sequence of library gates $(\Pi_{j=1}^{l} A_j \mid A_j \in \{L_i\}\{L_i^\dagger\})$ such that

$$||G - \Pi_l A_l|| \leqslant \epsilon \tag{1}$$

using the standard operator trace norm distance

$$||M - N|| = \text{Tr}\sqrt{(M-N)^\dagger (M-N)}. \tag{2}$$

All gates are represented here by unitary operators in SU(n), where n is the dimensionality of the gate. The theorem further states that the length of the sequences l varies with the required accuracy ϵ as $l = O(\log_{10}^c(1/\epsilon))$.

Exactly what the constant c is depends on the particular implementation of the decomposition. It is known that the best possible scaling is $c = 1$, but with a nonconstructive proof [14]. The standard algorithm gives a scaling of sequence size with accuracy of $c \approx 4$ [15].

The most straightforward procedure for performing such a decomposition is to search over all sequences, beginning with the shortest first, until one is found within the required ϵ of the gate G being decomposed. Unfortunately, such an exhaustive search becomes untenable very quickly. For a library of n fundamental gates, the number of sequences of length l comprising these library gates and their adjoints is $2n^l$. For example, if we have a library of five single-qubit gates (as, for example, in the surface code: $H, S, S^\dagger, T, T^\dagger$), then a modern server with 64 GB of memory could hold only up to sequences of length $l \approx 13$. Searching over this size of database is also a significant classical processing task. The longer the sequences are, the higher the chance of finding a sequence within ϵ of G is; without the ability to search longer sequences, the correct accuracy may be unobtainable.

The standard Solovay-Kitaev algorithm uses such an exhaustive search technique at its base layer but then builds on that recursively. We can describe the algorithm in iterative fashion as follows. The base-level approximation comprises a search over the space of all sequences of length up to l_0. The closest approximation to G is found:

$$\zeta(0) = \Pi_{j=1}^{l_0} A_j \mid A_j \in \{L_i\} \cup \{L_i^\dagger\}. \tag{3}$$

We can then decompose the gate as $G = U(\delta)\zeta(0)$. The operator $U(\delta)$ is the "residual" of the approximation: how far away from G the operator sequence $\zeta(0)$ still is.

If $||G - \zeta(0)|| \leqslant \epsilon$, then the algorithm terminates here and returns $\zeta(0)$ as the appropriate gate sequence. If the residual $U(\delta)$ is too great, however, the algorithm proceeds to the next level. A further exhaustive search of the space of sequences of length up to l_0 is performed, this time in order to find the best approximation to $U(\delta)$. A subtlety at this step in the algorithm is that we do not have a closed form for $U(\delta)$, so we need to find an approximation for $G\zeta^\dagger(0)$ instead. The algorithm performs this by decomposing further into $VWV^\dagger W^\dagger = G\zeta^\dagger(0)$, where V and W are the unitary gates that are then searched for. The sequence $VWV^\dagger W^\dagger$ that is closest to $U(\delta)$ is then returned by the search, so the first-level approximation becomes

$$\zeta(1) = VWV^\dagger W^\dagger \zeta(0). \tag{4}$$

Note that $V, W, \zeta(0)$ are all sequences of length up to l_0; the sequence $\zeta(1)$ is therefore of length up to $5l_0$.

If $||G - \zeta(1)|| \leqslant \epsilon$, then the algorithm terminates and returns $\zeta(1)$. If not, the previous step is repeated to find a decomposition of the residual $U(\delta_1) = G\zeta^\dagger(1)$. This is repeated until a sequence of the desired accuracy is found.

As we can see from Eq. (4), in the standard Solovay-Kitaev algorithm, the length of the approximating sequence grows by a factor of 5 at each level of recursion. As a result, the algorithm can only produce the approximating sequences of the length in the set $\zeta_l = \{l_0, 5l_0, 25l_0, 125l_0, \ldots\}$, where l_0 is the length of the approximating sequence for the basic stage of the decomposition algorithm. Therefore the vast

majority of possible approximating sequences, which are not in ζ_l, cannot be generated by the algorithm. Furthermore, the best strategy for finding a short gate sequence is to make the initial approximation length l_0 as large as possible; however, this then means that at every step in the algorithm the length of the approximating sequence grows dramatically. Because of this, the most efficient approximating sequences are likely to be missed, leading to much longer sequences than are necessary to reach the desired accuracy of approximation.

III. SEARCH SPACE EXPANSION

The technique we will use is to expand the search space at the first level of the standard algorithm so that sequences that are longer than l_0 are also covered. We assume that l_0 is the longest possible sequence space that our computational resources can exhaustively search. By improving the accuracy of this initial approximation ϵ_0, we reduce the residual to be approximated at the next level of the algorithm. It is then intuitively reasonable that this will reduce the number of recursion levels implemented to find a sequence accurate to a given ϵ. This is important, as the standard Solovay-Kitaev algorithm increases both the output sequence length and the processing time exponentially over the recursion levels. Formally, the residual error at recursion level n is given by

$$\epsilon_n = \frac{1}{c_{\text{approx}}^2}\left(\epsilon_0 c_{\text{approx}}^2\right)^{(\frac{3}{2})^n}, \tag{5}$$

where c_{approx} is a (small) constant that gives the error of the initial level of recursion, $\epsilon_0 < c_{\text{approx}}^{-2}$ [15]. We can therefore conclude that the more we reduce ϵ_0, the smaller ϵ_n will be, and therefore the sooner the algorithm will find a sequence $\epsilon_n \leqslant \epsilon$.

The initial stage of the algorithm gives us the first approximation gate sequence $\zeta(0)$, Eq. (3). The residual distance ϵ_0 from the exact gate G is

$$||G - \zeta(0)|| \leqslant \epsilon_0. \tag{6}$$

We now start our space expansion technique. For simplicity, we partition the sequence $\zeta(0)$ into two equal halves (in fact, the procedure can be performed by splitting the sequence into any number of parts, which may be unequal). Each of these subsequences $\zeta^{(1)}(0),\zeta^{(2)}(0)$ is of length $l_0/2$. For example, for a sequence $\zeta(0) = HT^\dagger S^\dagger T$, we would have $\zeta^{(1)}(0) = HT^\dagger$, $\zeta^{(2)}(0) = S^\dagger T$.

We now search once again over the space of sequences of length l_0 [16] to find approximations $\{Z_i^{(1)}(0)\}$ and $\{Z_j^{(2)}(0)\}$ within $\bar{\epsilon}_0$ of $\zeta^{(1)}(0)$ and $\zeta^{(2)}(0)$:

$$\begin{aligned} \forall\, i \;&\left|\left|\zeta^{(1)}(0) - Z_i^{(1)}(0)\right|\right| \leqslant \bar{\epsilon}_0, \\ \forall\, j \;&\left|\left|\zeta^{(2)}(0) - Z_j^{(2)}(0)\right|\right| \leqslant \bar{\epsilon}_0, \end{aligned} \tag{7}$$

where we define the search regions by

$$\bar{\epsilon}_0 = 0.5\epsilon_0. \tag{8}$$

Figure 1 shows this procedure schematically.

Algorithm 1: Space_Expansion

Input: $G \in \text{SU}(2)$: Target of approximation
Input: ζ: Universal set of gates
Input: ζ_0: Stored set of sequences of gates from ζ of length l_0
Input: ζ_1: Stored set of sequences of gates from ζ of length l_1
Input: ϵ_0: Accuracy of initial approximation
Input: $\bar{\epsilon}_0$: Accuracy of loop-internal approximation
Input: k: The desired cardinality of the set to be returned. $k = 1$ when Space_Expansion is used as a stand-alone algorithm, $k > 1$ when Space_Expansion is used as a component of Recursive_Space_Expansion.
Output: k approximating sequences for G of length $2l_1$

```
Space_Expansion (G, ζ, ζ0, ζ1, ε0, ε̄0, k)
R ← {r ∈ ζ0 | distance(r, G) ≤ ε0};
foreach r ∈ R do
    Split r into two subsequences of the same length l0/2,
      called r_pre and r_suf;
    R1 ← {r1 ∈ ζ1 | distance(r1, r_pre) ≤ ε̄0};
    R2 ← {r2 ∈ ζ1 | distance(r2, r_suf) ≤ ε̄0};
    Join R1 and R2 to have the following set:
    R3 ← {r1 r2 | r1 ∈ R1 and r2 ∈ R2};
end
R4 ← k best approximations for G ∈ R3;
return R4
```

We now form the set of sequences

$$\left\{Z_{ij}(0) = Z_i^{(1)}(0)Z_j^{(2)}(0)\right\}. \tag{9}$$

These sequences are of length $2l_1$ and are derived from $\zeta(0)$. We then search the sequences $Z_{ij}(0)$ to find the one, $Z(0)$, with minimal distance to the actual gate G,

$$||G - Z(0)|| = \bar{\epsilon}_{\text{min}}. \tag{10}$$

We now wish to show that, in general,

$$\bar{\epsilon}_{\text{min}} < \bar{\epsilon}(0) + \bar{\epsilon}(0) = \epsilon(0), \tag{11}$$

that is, that we have found a sequence $Z(0)$ of length $2l_1$ that is a closer approximation to G than the sequence $\zeta(0)$ of length l_0, given by the original Solovay-Kitaev algorithm. To do this, consider any four gate sequences $U_{1,2}$, $V_{1,2}$. Given that

$$||U_1 - V_1|| \leqslant \epsilon_1, \quad ||U_2 - V_2|| \leqslant \epsilon_2, \tag{12}$$

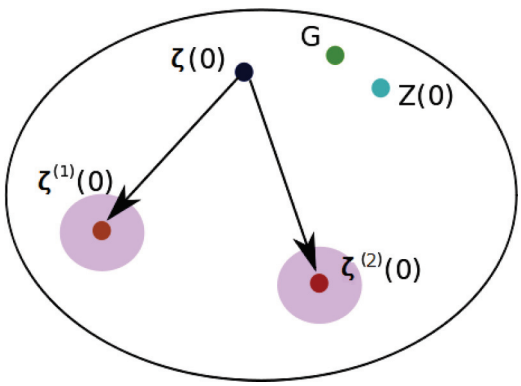

FIG. 1. (Color online) Expanding the search space. Sequence $\zeta(0)$ (a good first approximation to gate G) is split into two halves, $\zeta(0) = \zeta^{(1)}(0)\zeta^{(2)}(0)$. Points in regions of distance $\bar{\epsilon}_0$ (shaded) away from $\zeta^{(1)}(0),\zeta^{(2)}(0)$ are recombined, and the one closest to G, $Z(0)$, is chosen. With high probability, $||G - \zeta(0)|| > ||G - Z(0)||$.

we have that

$$||U_1U_2 - V_1V_2|| \leqslant ||U_1U_2 - V_1U_2|| + ||V_1U_2 - V_1V_2||; \tag{13}$$

therefore

$$\begin{aligned} &||U_1U_2 - V_1U_2|| + ||V_1U_2 - V_1V_2|| \\ &\quad = ||U_1 - V_1|| + ||U_2 - V_2|| \\ &\quad \leqslant \epsilon_1 + \epsilon_2. \end{aligned} \tag{14}$$

We can therefore conclude that

$$||U_1U_2 - V_1V_2|| \leqslant \epsilon_1 + \epsilon_2. \tag{15}$$

Since ϵ_1 and ϵ_2 are relatively small (U_1 is a close approximation for V_1 and U_2 is a close approximation for V_2), it is possible to say that a sequence U_1U_2 is a relatively close approximation for V_1V_2. If each $U \in \mathrm{SU}(2)$ is presented by the specified vector $u \in \mathbb{R}^3$ or equivalently specified point in three-dimensional space (see below for further details of this mapping), U_1U_2 is in the sphere of center V_1V_2 and radius $\epsilon_1 + \epsilon_2$. With a large number of sequences U_1 and U_2 and their combinations U_1U_2, the probability of having a very close approximation for V_1V_2 is relatively high.

We therefore conclude that

$$\bar{\epsilon}_{\min} < \epsilon(0) \tag{16}$$

for our new approximating sequence $Z(0)$; that is, we have strictly reduced the error in approximation by doubling the length of the approximating sequences but not going on to the next level of recursion of the standard algorithm, which would increase it by a factor of 5.

We call this technique as it stands *search space expansion* (SSE). At the initial stage of the Solovay-Kitaev algorithm the search space is expanded once, which significantly reduces the residual and hence the number of subsequent levels of recursion needed. Algorithmically, it can be written in the pseudocode form given in Algorithm 1.

We can, however, also apply SSE itself recursively at this initial stage to get an even better $\zeta(0)$ approximation. Rather than performing a standard search over sequences of length l_1 to find our approximations for $\zeta^{(1)}(0)$ and $\zeta^{(2)}(0)$ [Eq. (7)], we use SSE itself to find better approximations. So we use $\zeta^{(1)}(0)$ and $\zeta^{(2)}(0)$ as query gates for two SSE procedures treating the remainder of the search space as in standard SSE.

The final approximation for $\zeta(0)$ is therefore a sequence $Z'(0)$ of length $4l_1$, where

$$Z'(0) = Z_a^{(1)}(0)Z_b^{(1)}(0)Z_a^{(2)}(0)Z_b^{(2)}(0). \tag{17}$$

We call this technique *recursive search space expansion* (recursive SSE), given in pseudocode form in Algorithm 2. We have now expanded the lookup space much further than was possible with the original Solovay-Kitaev algorithm, allowing for a much denser search of the space of possible approximating sequences. The structure of the algorithm remains unchanged and, as a consequence, so does the scaling of the accuracy of the approximation with the length of the sequences. The length l of a sequence for accuracy ϵ is still given by

$$l = O(\log_{10}^c(1/\epsilon)), \tag{18}$$

with $c \approx 3.97$, as in [15]. However, this technique should significantly reduce the prefactor in scaling. We also note that these techniques, SSE and recursive SSE, can be used with any modification of the original Solovay-Kitaev algorithm that requires lookup in a database of library sequences.

Algorithm 2: Recursive_Space_Expansion

Input: $G \in \mathrm{SU}(2)$: Target of approximation
Input: ζ: Universal set of instructions
Input: ζ_0: Stored set of instruction sequences of gates from ζ of length l_0
Input: ζ_1: Stored set of instruction sequences of gates from ζ of length l_1
Input: ϵ_0: Accuracy of initial approximation
Input: ϵ_1: Parameter for Space_Expansion function, indicates the accuracy of initial approximation for the called Space_Expansion function
Input: $\bar{\epsilon_1}$: Parameter for Space_Expansion function, indicates the accuracy of loop-internal approximation for the called Space_Expansion function
Input: k: Cardinality of the set of sequences that will be requested from Space_Expansion $k > 1$
Output: Approximating sequence for G of length $4l_1$

```
Recursive_Space_Expansion (G, ζ, ζ0, ζ1, ε0, ε̄0, k)
R ← {r ∈ ζ0 | distance(r, G) ≤ ε0};
foreach r ∈ R do
    Split r into two subsequences of the same length l0/2,
      called r_pre and r_suf;
    R1 ← {r1 ∈ Space_Expansion(r_pre, ζ, ζ0, ζ1, ε1, ε̄1, k)};
    R2 ← {r2 ∈ Space_Expansion(r_suf, ζ, ζ0, ζ1, ε1, ε̄1, k)};
    Join R1 and R2 to have the following set:
    R3 ← {r1 r2 | r1 ∈ R1 and r2 ∈ R2};
end
r3 ← The best approximation for G in R3;
return r3
```

IV. INCREASING LOOKUP EFFICIENCY USING GEOMETRIC SEARCH

The technique we have just described is very powerful in extending the set of searched sequences without requiring an exhaustive search over all possible approximations. However, there is an additional search cost for these methods that is not present in the Solovay-Kitaev algorithm as commonly used.

First, each time SSE is invoked (either on its own or as part of a recursive SSE step), the space of sequences of length l_0 needs to be searched to find the regions of sequences that are distance $\bar{\epsilon}_0$ away from the subsequences $\zeta^{(1,2)}(0)$. Each time SSE is used, an additional two searches are required to find the desired regions. Second, whenever sequences are combined to form a longer approximation to G, the list of combined sequences needs to be searched to find the one that is closest to G. This happens once per use of SSE or recursive SSE.

The first of these is by far the largest cost in our technique, as it can occur many times in a given use of the decomposition algorithm. The second is only incurred once per level of recursion in the standard algorithm. We can keep the second cost tolerable by not increasing the number of times SSE is used recursively on itself: for this reason, we describe recursive SSE as only splitting the sequence twice; for any more times, the search cost to find the best $Z(0)$ [Eq. (17)] would be prohibitive. Without this restriction, we could use recursive SSE many times on itself to find sequences of arbitrary length and have no need for the structure of the original Solovay-Kitaev algorithm. However, given the exponentially

increasing cost of this search, we chose instead to restrict recursive SSE to two applications of SSE and to then proceed to the next level of Solovay-Kitaev recursion if further accuracy in the decomposition is required.

In order for our techniques to be useful in feasible computational time, we need to find a way of performing the region-finding search in SSE efficiently. The most straightforward way is to search over the entire space of sequences up to length l_0 and pick out those within distance $\bar{\epsilon}_0$. Such a linear search is, however, very inefficient: in general, the search time will be exponential in l_0. However, we are searching a very structured space and should be able to make use of this structure in order to increase the efficiency of this search step. We will show now how to convert our matrix search problem into a three-dimensional (3D) geometric search problem and how we can then use the existing technique of GNATs to solve the search problem much more efficiently. We can use such a geometric technique in the original Dawson and Nielsen algorithm for the decomposition as well, and in the next section we will use it instead of a linear search when comparing our SSE techniques with the original algorithm. More sophisticated search techniques at this step are beginning to be developed, including the database searches of [17,18]. A further alternative use of geometric search techniques for the Dawson-Nielsen algorithm has been examined simultaneously and came to our attention after our work was completed [19].

We can convert any matrix SU(2) into a unique vector in a ball in a 3D real vector space of radius 2π and centered on the origin ([20], chapter 5). We can uniquely write any $U \in \mathrm{SU}(2)$ as

$$U = e^{\frac{-i}{2}\mathbf{v}\cdot\sigma} \longrightarrow u(\mathbf{v}), \tag{19}$$

where $\mathbf{v} \in \mathbb{R}^3$ and σ is formed of the Pauli matrices,

$$\sigma = \begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{pmatrix}. \tag{20}$$

We therefore can fully and uniquely specify the matrix U by specifying the vector $\mathbf{v}$. The distance function that we are using to search is now no longer the trace norm between two matrices U_1, U_2, but rather the norm between the two specifying vectors given by the Euclidean distance function.

We therefore have a 3D geometric search problem of finding vectors within the ball of radius $2\pi\bar{\epsilon}_0$ centered on the vector corresponding to the matrix $\zeta^{(1,2)}(0)$. Such geometric searches in real space have been extensively studied, and we can therefore pick from the existing techniques the one that best suits our purposes.

A. GNAT search

The standard method for increasing the efficiency of a real-space search is to use a *tree-based* approach. By dividing the search space into clusters, tree-based searches are capable of reducing the computational complexity of a search over n entries from $O(n)$ in the case of the straightforward linear search to $O(\log_{10}(n))$, at least in the best or average case. The increasing efficiency comes from the ability of the search algorithms to skip entire clusters that are evidently not going to contain a correct answer. Among the most popular approaches are those of R trees and k-d trees [10,11]. These approaches

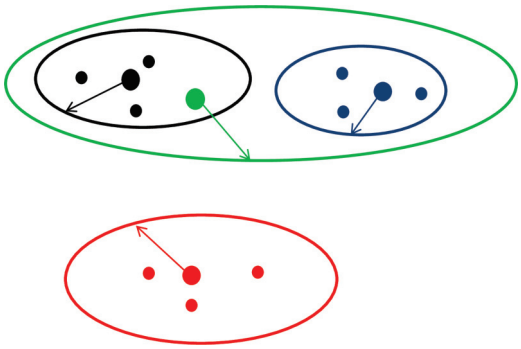

FIG. 2. (Color online) Schematic diagram of a simple GNAT with clusters (see text for explanation).

divide the search space into clusters specified by coordinates in the space. These approaches were tried and did not yield considerably improved search results. Instead, we can use the approach of geometric nearest-neighbor access trees. The region-based cluster methods of GNAT seem more applicable to the space of unitary operators than those which use a simple, global, Euclidean metric to partition the space.

In a GNAT search, the space is partitioned into clusters around a number of fixed points called *splitting points* [9]. How best to choose these splitting points is an open research question; we use the common approach of picking them at random. At the initial stage of processing, each vector in the space is then analyzed in turn to find its closest splitting point and the distance from that point. The set of vectors that are closest to a given splitting point is the *cluster* associated with that point. The cluster size can be defined by the greatest and least distances from all vectors in the cluster to the splitting point. Each cluster, if desired, can in turn be partitioned using splitting points to produce subclusters. This procedure can be applied recursively as many times as desired, until the size of each cluster is small enough that a linear search within the cluster is feasible.

Figure 2 shows in two-dimensional (2D) space a simple GNAT with clusters. Each cluster is represented by an ellipse with its inside points given by a circle. Each cluster is equivalent to a branch in GNAT. The set of all points is implicitly equivalent to the GNAT root cluster. Each cluster is derived from its specific splitting point, which is represented by a circle from which there is an arrow pointing to its border. As is evident from Fig. 2, the splitting point is inside the cluster it generates.

Calculating and storing the data associated with each point (which cluster it belongs to and how far away it is from the splitting point) and each cluster (maximum and minimum distance between splitting point and all data points in the cluster) are the most computationally expensive step in the search procedure. However, when we use this procedure in our modified Solovay-Kitaev algorithm, we only need to perform this step a single time for each set of library gates. This can be performed offline before the algorithm begins, and the structured data can be reused for any decomposition problem using that library set.

The search proceeds as follows. Suppose we are searching for all the nearest neighbors $\{p\}$ that are within distance ϵ of a given query point x, i.e., $D(p,x) \leqslant \epsilon$, where D is our distance

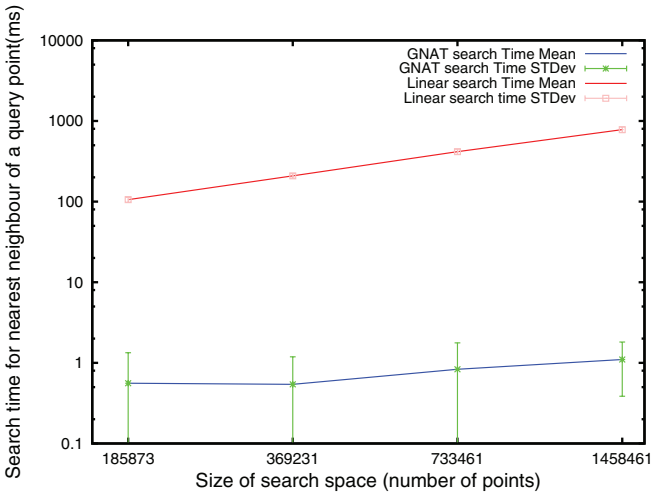

FIG. 3. (Color online) GNAT vs linear search time for stored sequence databases with $l_0 = 16,17,18,19$. The program is implemented in JAVA and run on an UBUNTU 11.10 32 byte type operating system (OS), Core i7-2670QM 2.20 GHz Intel processor, with 2.9 GB of memory. The apparent asymmetry in the error bars is due to the logarithmic vertical scale.

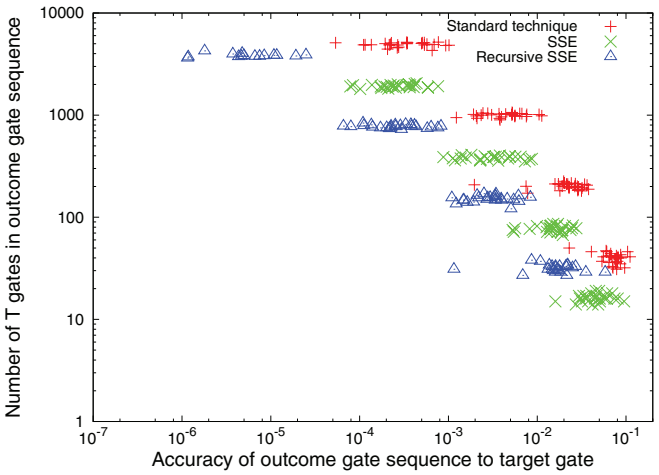

FIG. 4. (Color online) Approximation accuracy vs the length of the best approximating gate sequence found (number of T gates).

function. We start by looking at the top-level clusters, with splitting points s_i. The distance between the splitting point and the query point is $D(s_i,x) = d_i$. By then applying the triangle inequality, we can see that the points $\{p\}$ can only belong to clusters where the distance between the point p and its splitting point satisfies

$$d_i - \epsilon \leqslant D(p,s_i) \leqslant d_i + \epsilon. \tag{21}$$

All other clusters can be rejected, and the search at the next level can be concentrated on those that remain.

We can see how GNAT significantly improves the search time over a linear search. Figure 3 shows the average time in milliseconds to find the ϵ region around a query point for data sets with different volumes. These data are the average over 120 runs for each search space size, choosing different query points each time. Note in particular that the time axis is on a log scale: GNAT clearly outperforms linear search by two orders of magnitude. Figure 3 may look curious as the time cost of GNAT search appears to vary nonmonotonically. This comes from the fact that the efficiency of a GNAT search varies given how good the splitting points turn out to be for a given data space. Relatively distant splitting points result in a more balanced search tree and better performance. A random distribution of splitting points was used here; an alternative method of empirically choosing the points also exists [9].

V. COMPARISON WITH THE ORIGINAL ALGORITHM

In order to test the modifications to the Solovay-Kitaev algorithm, we implemented the original, original + SSE, and original + recursive SSE algorithms to find the length of gate sequences generated for a given gate and level of accuracy. A set of 25 different randomly generated matrices in SU(2) was generated, and then approximations were found for each matrix using the three different algorithms. The library gate set used was the minimal set $\{H,T,T^\dagger\}$, and the stored sequences were of length up to $l_0 = 18$. For each gate the best approximation at each of the first four levels of Solovay-Kitaev recursion was recorded. In all cases, GNAT search was used to find ϵ regions in the space of sequences of length up to l_0.

The largest quantum cost in fault-tolerantly implementing a sequence approximating the desired gate is, in general, in implementing the T gates (as these require magic state injection). We therefore take as the appropriate measure of length for an approximating sequence the number of T gates in the sequence. Figure 4 shows the length (number of T gates) vs the accuracy of the approximation for the best approximating sequence found by each algorithm for each of the 25 randomly generated unitary matrices. The steplike behavior in all three cases is caused by the recursion levels of the original algorithm. Both SSE and recursive SSE are significant improvements over the original algorithm, with recursive SSE clearly the better of the two. In both cases the length of sequences for a given approximation accuracy is reduced; for example, when the accuracy required is around 10^{-4}, then SSE alone reduces the length of the best sequence by a factor of 3, and recursive SSE reduces it by a factor of 7. The number of levels of Solovay-Kitaev recursion also reduces significantly, from $n = 4$ using the original algorithm to $n = 3$ with SSE and to $n = 2$ with recursive SSE.

Recursive SSE is therefore clearly better than the standard Solovay-Kitaev algorithm at producing gate sequences that cost less to implement in terms of quantum resources on a quantum computer. However, this is not the only consideration: we must also take into account the *classical* processing time needed to find the best approximating sequence in each of the cases. Figure 5 shows this classical preprocessing time vs the accuracy of the resulting approximating sequence. As the two plots are of the same data set, they may be directly compared through their x axes. As we would expect, the more computationally intensive recursive SSE algorithm takes a much longer time to run. However, note the log scale on the time axis; the curve for recursive SSE is, in fact, subexponential in its scaling.

We can therefore conclude that there is a straightforward trade-off between reducing the quantum cost of implementing

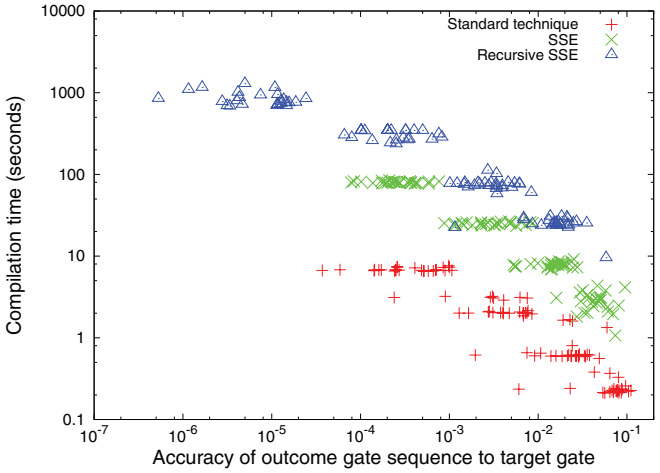

FIG. 5. (Color online) Approximation accuracy vs classical compilation time to find the best approximating gate sequence. The program is implemented in JAVA and run on an UBUNTU 11.10 32 byte type OS, Core i7-2670QM 2.20 GHz Intel processor, with 2.9 GB of memory.

a given single-qubit unitary and the classical preprocessing time to find the sequence. In general, given the relative state of the two technologies, we would prioritize decreasing the quantum cost at the expense of classical processing. It is also important to bear in mind that sequence finding can be run offline, before the algorithm starts, whereas implementing the approximating sequence in terms of quantum gates is by definition online. We therefore have an algorithm that gives us a shorter approximating sequence, using classical processing that scales less strongly than in [8]. This is therefore a middle ground between such a flat, linear search and the structured, sparse search of the standard recursive Solovay-Kitaev algorithm.

VI. CONCLUSION

We have given a modified version of the Solovay-Kitaev algorithm that greatly reduces the length of the sequences used to approximate a unitary single-qubit gate. Our technique also reduces the number of levels of recursion required by the algorithm to reach a given level of accuracy. By reducing the depth of the quantum circuit used to approximate a given gate, we are then able to reduce the amount of error correction needed for fault-tolerant implementations of quantum algorithms. The cost for this is an increase in the classical processing needed to find these shorter and less costly quantum sequences, but the use of structured GNAT searches enables this to be performed in time that scales subexponentially (and can also be performed offline before the quantum algorithm starts). By increasing the space that is searched at the initial level of recursion in the original algorithm, we are able to use the powerful method of the recursive steps without leaving so much of the search space unexplored between levels of recursion.

Note added. Recently, we became aware of [21,22] that show new methods for effective compilation using the $\langle H,T\rangle$ basis set. The techniques presented in this paper, however, remain valuable extensions to the core search algorithm and will be especially valuable for compilation using other basis sets.

ACKNOWLEDGMENTS

D.H. and R.V. acknowledge useful discussions with Krysta Svore and Alex Bocharov. D.H. appreciates valuable discussions with Dominic Berry and Barry Sanders. This work was supported by the Japan Society for the Promotion of Science (JSPS) through its "Funding Program for World-Leading Innovative R&D on Science and Technology (FIRST Program)."

[1] J. Preskill, in *Introduction to Quantum Computation and Information*, edited by H.-K. Lo, T. P. Spiller, and S. Popescu (World Scientific, Singapore, 1998), pp. 213–270.
[2] J. Preskill, Proc. R. Soc. London, Ser. A **454**, 385 (1998).
[3] P. W. Shor, Phys. Rev. A **52**, R2493 (1995).
[4] D. P. DiVincenzo and P. W. Shor, Phys. Rev. Lett. **77**, 3260 (1996).
[5] S. J. Devitt, K. Nemoto, and W. J. Munro, arXiv:0905.2794.
[6] A. Yu. Kitaev, Russ. Math. Surv. 6 **52**, 1191 (1997).
[7] A. Yu. Kitaev, A. Shen, and M. N Vyalyi, *Classical and Quantum Computation* (American Mathematical Society, Providence, RI, 2002).
[8] A. G. Fowler, Quantum Inf. Comput. **11**, 867 (2011).
[9] S. Brin, in *Proceedings of the 21st International Conference on Very Large Data Bases* (Morgan Kaufmann, Burlington, MA, 1995), pp. 574–584.
[10] T. K. Sellis, N. Roussopoulos, and C. Faloutsos, in *Proceedings of the 13th International Conference on Very Large Data Bases* (Morgan Kaufmann, Burlington, MA, 1987), pp. 507–518.
[11] J. L. Bentley, Commun. ACM **18**, 509 (1975).
[12] R. Raussendorf and J. Harrington, Phys. Rev. Lett. **98**, 190504 (2007).
[13] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, Phys. Rev. A **52**, 3457 (1995).
[14] A. W. Harrow, B. Recht, and I. L. Chuang, J. Math. Phys. **43**, 4445 (2002).
[15] C. M. Dawson and M. A. Nielsen, Quantum Inf. Comput. **6**, 81 (2006).
[16] In practice we will search over this maximum sequence space, although the general technique works for any subset of sequences $l_1 \leqslant l_0$.
[17] A. Bocharov and K. M. Svore, Phys. Rev. Lett. **109**, 190501 (2012).
[18] M. Mosca, V. Kliuchnikov, and D. Maslov, Quantum Inf. Comput. **13**, 607 (2013).
[19] D.-S. Wang, M. C. de Olivereira, D. W. Berry, and B. C. Sanders (private communication).
[20] Y. Kosmann-Schwarzbach, *Groups and Symmetries: From Finite Groups to Lie Groups* (Springer, Berlin, 2010).
[21] P. Selinger, arXiv:1212.6253.
[22] V. Kliuchnikov, D. Maslov, and M. Mosca, arXiv:1212.6964.