# Embedding Functions Into Reversible Circuits: A Probabilistic Approach to the Number of Lines

Niels Gleinig
Department of Computer Science,
ETH
Zurich, Switzerland
niels.gleinig@inf.ethz.ch

Frances Ann Hubis
Department of Computer Science,
ETH
Zurich, Switzerland
hubisf@inf.ethz.ch

Torsten Hoefler
Department of Computer Science,
ETH
Zurich, Switzerland
torsten.hoefler@inf.ethz.ch

## ABSTRACT

In order to compute a non-invertible function on a reversible circuit, one needs to "embed" the function into a larger function which has some garbage bits, corresponding to additional lines. The problem of determining the minimal number of garbage bits that are needed to embed a given function has attracted extensive research, largely motivated by quantum computing, where the number of lines equals the number of qubits. However, all approaches that are known have either no theoretical quality guarantees (bounds on approximation factors) or require exponential runtime. We present an efficient probabilistic approximation algorithm with theoretical bounds.

## KEYWORDS

Reversible Computing, Quantum Computing, Reversible Circuits, Circuit Synthesis

**ACM Reference Format:**
Niels Gleinig, Frances Ann Hubis, and Torsten Hoefler. 2019. Embedding Functions Into Reversible Circuits: A Probabilistic Approach to the Number of Lines. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317814

## 1 INTRODUCTION

Reversible computing is a form of computing in which all steps of a computation can be "undone" and hence the input can be recovered from the output. Examples include unitary operations in quantum computing and adiabatic computing. The associated reversible circuits are circuits that can be built out of gates that are individually reversible.

Reversible computing has many practical applications: it could be used to perform computations that require virtually no energy to be performed, while any irreversible computation requires a minimum amount of energy as stated by Landauer's principle. Perhaps more importantly, reversibility is required to implement certain computational functions. For example, any program executed on a quantum computer must be reversible because the fundamental theory of quantum mechanics bases on unitary evolution and unitary operators only exists for reversible functions. Many quantum algorithms require so called "classical oracles", which encode a (potentially irreversible) function. In order to implement such oracles on quantum computers, these functions need to be transformed into a reversible form. Furthermore, certain arithmetic needed to implement quantum algorithms such as Shor's prime factorization [3] require reversible modulo-addition.

To perform computations reversibly, the computations need to be made "logically reversible". For this we need to elongate the original output bit string (to be explained in more detail later). For example, in a quantum computer the number of lines corresponds to the number of qubits that are needed to compute the circuit. This is the most limited resource and we are interested in making it as small as possible. Today's quantum computers only have several dozens of low-fidelity qubits. The fidelity required for ideal fault-free execution has not been reached and researchers only expect a handful of reliable qubits to be available anytime soon. Thus, minimizing the required lines is of utmost importance in the design of reversible quantum circuits. It is also important to future-proof design methods by enabling them to scale to larger number of qubits. After all, quantum computation is in a stage comparable to the first transistors in the early 1960's and once a breakthrough is achieved (e.g., the practical implementation of Majorana qubits), practical chips may quickly reach thousands or even millions of qubits.

These strong motivations gave rise to extensive research on the underlying theoretical optimization problems. However, the existing approaches are hardly able to solve those for more than 1000 lines. Using probabilistic methods we are able to give approximate solutions for millions of lines. The price that we have to pay is that erroneous results can be obtained with certain probability. However, both the error probability and the distance to optimality can be well controlled by free parameters and the error probability can be pushed to zero with exponential speed.

### 1.1 Previous work

Several previously devised heuristics and exact algorithms aim at minimizing the number of lines of a given reversible circuit.

Wille et al. [8] presents an algorithm that reduces the number of lines by applying a list of steps for a given (non-optimal) reversible circuit. Experimental results show that this method can reduce the number of lines of the given circuit on average by 17%.

Wille et al. [7] introduce an exact algorithm that solves the problem of finding the minimal number of garbage bits. Furthermore this paper presents a heuristic that does not provide bounds on the

*DAC '19, June 2–6, 2019, Las Vegas, NV, USA*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06.
https://doi.org/10.1145/3316781.3317814

approximation factors. The exact algorithm scales exponentially in time and has so far only been applied to inputs of size no larger than $n = 65$. Heuristic methods were used to calculate embeddings for up to $n = 117$.

Soeken et al. [4] present an algorithm that finds for a given function a cascade of Toffoli gates that produce the function (circuit synthesis). The algorithm requires the function to be invertible already and it needs to be given in QMDD form. Experimental results are presented for up to $n = 100$ lines.

Maslov and Dueck [2] consider a more general problem than the one of this paper: For a given function $f : \{0,1\}^n \to \{0,1\}^m$ find the minimal number of lines that are needed to embed $f$ into a reversible circuit and then synthesize such a circuit. However, the part of finding the minimal number of lines is done by going through the whole truth table and counting which output occurs most often. This takes exponential time and space and hence is done in the experimental results section of that paper only for functions with $n \leq 9$ and $m \leq 10$.

The problem considered by Soeken et al. [5] is also more general than the one of this paper: Find the minimal number of lines and then find an invertible function that serves as embedding. Three different methods are presented to compute the minimal number of lines. The first method gives an approximation, but there are no theoretical bounds on the approximation factors. The other two methods give exact results, but unless coNP=P, they take more than polynomial runtime (an exact theoretical analysis of the runtime is not given, but it is shown that the problem that they solve is coNP-hard). Furthermore 2 different embedding methods are presented. In the experimental results section, exact results are obtained for $n, m \leq 65$. Heuristic results are obtained for $n \leq 143, m \leq 139$.

Zulehner and Wille [9] introduce another method for the embedding problem. In the experimental results section, results with up to $n \leq 65, m \leq 139$ are presented.

Overall, there exist many interesting approaches to solve the problem that we consider in this paper. However, to the best of our knowledge, none of the existing exact solutions works for general functions with, say, $n, m \geq 200$ (and due to the coNP-hardness they will most likely never exist). For the non-exact methods, there are no non-trivial theoretical bounds on approximation factors, that tell us how far away from optimality they are. Also many of them still do not work for $n, m \geq 1000$. So for the number of qubits in existing quantum computers, the existing methods may suffice, but for future quantum computers they will not suffice.

## 2 BACKGROUND AND PROBLEM FORMULATION

Suppose we are given a function $f : \{0,1\}^n \to \{0,1\}^m$. If $f$ is non-invertible (which is for example certainly the case if $m < n$) it can not directly be computed by a reversible circuit. A certain trick is needed: We search a $k \in \mathbb{N}$ for which there is an invertible function $\tilde{f} : \{0,1\}^{m+k} \to \{0,1\}^{m+k}$ such that for all $\vec{x} \in \{0,1\}^n$ and all $1 \leq i \leq m$, the $i$-th component of the output satisfies $\tilde{f}_i(\vec{x}, \vec{0}) = f_i(\vec{x})$, where $\vec{0}$ denotes the zero-vector with $m + k - n$ components. Now the functions $f$ and $\tilde{f}$ coincide on the first $m$ components and we will say that $f$ is embedded into $\tilde{f}$.

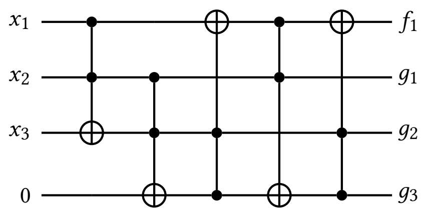


**Figure 1: Reversible circuit for $f : \{0,1\}^3 \to \{0,1\}$ with *Sum − Of − Product* (SOP) representation $(x_1, x_2, x_3) \mapsto f_1 = x_1 \vee x_2x_3$ and a truth table given in Table 1. From the truth table one can see that this function has $\mu = 5$ and hence $\lceil \log_2(5) \rceil = 3$ garbage bits are needed. Consequently this circuit is an embedding with the minimal number of lines.**

**Table 1: Truth Tables for Figure 1**

**(a) Original**

| $x_1$ | $x_2$ | $x_3$ | $f_1$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

**(b) Possible Embedding**

| $x_1$ | $x_2$ | $x_3$ | 0 | $f_1$ | $g_1$ | $g_2$ | $g_3$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

The new function $\tilde{f}$ is invertible and therefore it can be computed by a reversible circuit with $m + k$ lines.

It is easy to see that the smallest possible $k$ satisfies $n - m \leq k \leq n$, which gives good bounds when $m$ is small, but may be far off for large $m$.

We define for any $x \in \{0,1\}^n$ the set of input values that map to the same output as $x$

$$M_x := \{x' \in \{0,1\}^n | f(x) = f(x')\} \tag{1}$$

and the cardinality of the largest set on which $f$ is constant

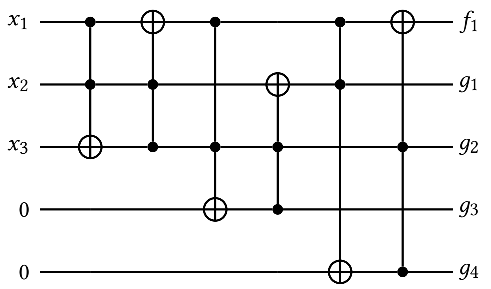
$$\mu := \max_{x \in \{0,1\}^n} |M_x|. \tag{2}$$

Then the smallest possible $k$ for which an embedding $\tilde{f}$ exists, is given by [2]

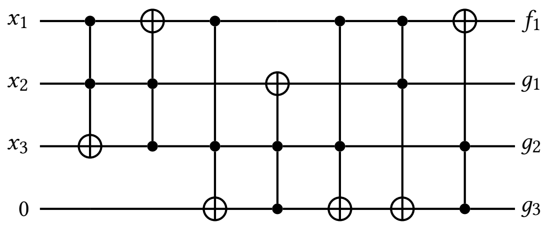
$$k = \lceil log_2(\mu) \rceil. \tag{3}$$

This can be deduced from the simple observation that with each bit that we add to the output, we can half the number of inputs (or half + $\frac{1}{2}$ if the cardinality is odd) that map to one particular output. So by adding $i$ bits to the output the cardinality of the largest set on which our function is constant, can be reduced to $\lceil \mu \cdot 2^{-i} \rceil$ and with $i = k = \lceil log_2(\mu) \rceil$ this quantity becomes 1 and all inputs map to different outputs.

Summarizing this, the problem of determining the minimal number of garbage bits that are needed is computationally equivalent



(a)



(b)

**Figure 2: Reversible circuits for $f : \{0,1\}^3 \to \{0,1\}$ with** *SumOfProducts* **(SOP) representation $(x_1, x_2, x_3) \mapsto f_1 = x_1\bar{x_2}\bar{x_3} \vee x_1x_2x_3$, and a truth table given in Table 2. This function has $\mu = 5$ and hence $\lceil \log_2(5) \rceil = 3$ garbage bits are needed (a) Non-optimal embedding with 4 garbage bits, (b) Optimal embedding with 3 garbage bits**

to solving the following problem in which we introduced the normalizing factor $\frac{1}{2^n}$.

**The maximal preimage problem (MPP):** Let $f : \{0,1\}^n \to \{0,1\}^m$. Determine

$$c_f := \frac{1}{2^n} \max_{x \in \{0,1\}^n} |M_x|. \tag{4}$$

Unfortunately this problem is coNP-hard [5].

The reason why we have introduced the normalizing factor to the problem formulation is to facilitate the probabilistic interpretation of this problem: The probability that a uniformly at random chosen $x$ satisfies $|M_x| = \mu$ is at least $c_f$.

## 3 ALGORITHM AND RESULTS

In this paper we introduce the Maximal Preimage Sampling Algorithm (MPS algorithm), which is a probabilistic additive approximation scheme for the MPP. The properties of the MPS algorithm are summarized in the following theorem.

Theorem 3.1. *The MPS algorithm takes as input a function $f : \{0,1\}^n \to \{0,1\}^m$ and $\epsilon_d, \epsilon_p > 0$ and outputs a number $\hat{c}$. If $c_f$ is the actual answer to the MPP problem for $f$, we have $|\hat{c} - c_f| < \epsilon_d$ with probability at least $1 - \epsilon_p$. The runtime of the algorithm is*

$$\Theta\left((n + m + ET(f)) \frac{\log_2(\epsilon_p)}{\epsilon_d^3} \log_e\left(1 - (1 - \frac{\epsilon_p}{2})^{\frac{\log_2(1-\epsilon_d)}{\log_2(\epsilon_p/2)}}\right)\right), \tag{5}$$

*where $ET(f)$ denotes the average time that it takes to evaluate $f$.*

**Table 2: Truth Tables for Figure 2**

**(a) Original**

| $x_1$ | $x_2$ | $x_3$ | $f_1$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

**(b) Possible Embedding**

| $x_1$ | $x_2$ | $x_3$ | 0 | $f_1$ | $g_1$ | $g_2$ | $g_3$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

The algorithm implements a basic sampling procedure, where the sample sizes correspond to the number of iterations of an inner and an outer loop. For a probabilistic guarantee given by Theorem 3.1 and sample sizes

$$\lceil S \rceil = \lceil \log_2(\epsilon_p/2)/\log_2(1 - \epsilon_d) \rceil \tag{6}$$

$$\lceil T \rceil = \lceil -\frac{1}{2\epsilon_d^2} \log_e\left(\frac{1}{2}\left(1 - (1 - \frac{\epsilon_p}{2})^{\frac{1}{\lceil S \rceil}}\right)\right) \rceil \tag{7}$$

the algorithm outputs a $(\epsilon_d, \epsilon_p)$-secure MPP estimate. On a high level, the outer loop samples an input bit string uniformly at random, evaluates the output bit string of the given function and compares this image to the image of all inner-loop input bit strings, which are also sampled uniformly at random. If the number of identical images is larger than for any other previous sample of the outer loop, this preimage size is normalized and replaces the current MPP estimate.

The double loop seems inefficient: Why not just run one loop and count which output occurs most often? In that case one would be looking at a random variable that is not binomial and consequently Hoeffding's inequality and all of our probabilistic analysis would not apply anymore. The general problem of a random variable that can take many different values, is that when we sample it many times, some values will occur with a much larger frequency than their actual probability.

Interestingly the algorithm does not require to have $f$ represented in any particular data structure (like a QMDD, ESOP, BDD, truth table etc) and if $f$ is a function that can be evaluated in polynomial time then the MPS runs in polynomial time.

Also, using some calculus one can see that for fixed $n, m, f, \epsilon_d$, the runtime grows like $\log(\epsilon_p)^2$ as $\epsilon_p \to 0$. So the error probability can be pushed to 0 with speed $\exp(\sqrt{\text{runtime}})$.

### 3.1 Proofs

To prove Theorem 3.1 we first need a lemma. We also define for this whole section the quantities

$$c_x := \frac{1}{2^n}|\{x' \in \{0,1\}^n \,|f(x') = f(x)\}| = \frac{1}{2^n}|M_x|. \tag{8}$$

**ALGORITHM 1:** Input: A function $f : \{0,1\}^n \to \{0,1\}^m$ and $\epsilon_d, \epsilon_p > 0$ . Output: MPP estimate $\hat{c} \in \mathbb{R}$". Multivariate random variables : x (drawn in outer loop of size $\lceil S \rceil$), x' (drawn in inner loop of size $\lceil T \rceil$).

```
Set $\hat{c} = 0$ and $S = \log_2(\epsilon_p/2)/\log_2(1-\epsilon_d)$ and
$T = -\frac{1}{2\epsilon_d^2}\log_e\left(\frac{1}{2}\left(1-(1-\frac{\epsilon_p}{2})^{\frac{1}{\lceil S \rceil}}\right)\right)$ ;
for  $i$ from 1 to $\lceil S \rceil$ do
    Sample $x$ uniformly at random from $\{0,1\}^n$ and set $\hat{c}_x = 0$;
    for $j$ from 1 to $\lceil T \rceil$ do
        Sample $x'$ uniformly at random from $\{0,1\}^n$;
        if $f(x') = f(x)$ then
            $\hat{c}_x = \hat{c}_x + 1$;
        end if
    end for
    $\hat{c}_x = \frac{1}{\lceil T \rceil}\hat{c}_x$
    $\hat{c} = \max(\hat{c}, \hat{c}_x)$;
end for
return  $\hat{c}$;
```

Lemma 3.2. *Define the event*

$$A := \text{"for all } x \text{ that we sample in the outer for-loop of the MPS algorithm, we have } |c_x - \hat{c}_x| \le \epsilon_d\text{"}. \tag{9}$$

*Then*

$$\mathbb{P}(A) \ge 1 - \epsilon_p/2. \tag{10}$$

Proof of Lemma 3.2. Notice that for each $x'$ that we sample in the inner for-loop, we will have $f(x) = f(x')$ with probability $c_x$ (because $c_x$ is precisely the proportion of input values that satisfy $f(x) = f(x')$). So $\lceil T \rceil \cdot \hat{c}_x$ is distributed like a $B(\lceil T \rceil, c_x)$ random variable (see appendix A). By Hoeffding's inequality

$$\mathbb{P}(|c_x - \hat{c}_x| \le \epsilon_d) \ge 1 - 2\exp\left(-2\lceil T \rceil \epsilon_d^2\right). \tag{11}$$

So the probability that for all of the $\lceil S \rceil$ samples of $x$ from the outer for-loop we have $|c_x - \hat{c}_x| \le \epsilon_d$ (this is precisely the event $A$) is

$$\mathbb{P}(A) \ge \left(1 - 2\exp\left(-2\lceil T \rceil \epsilon_d^2\right)\right)^{\lceil S \rceil}, \tag{12}$$

and we have

$$\begin{aligned} &\left(1 - 2\exp\left(-2\lceil T \rceil \epsilon_d^2\right)\right)^{\lceil S \rceil} \ge 1 - \frac{\epsilon_p}{2} \\ \Leftrightarrow\ & 1 - 2\exp\left(-2\lceil T \rceil \epsilon_d^2\right) \ge \left(1 - \frac{\epsilon_p}{2}\right)^{\frac{1}{\lceil S \rceil}} \\ \Leftrightarrow\ & \exp\left(-2\lceil T \rceil \epsilon_d^2\right) \le \frac{1}{2}\left(1 - \left(1 - \frac{\epsilon_p}{2}\right)^{\frac{1}{\lceil S \rceil}}\right) \\ \Leftrightarrow\ & -2\lceil T \rceil \epsilon_d^2 \le \log_e\left(\frac{1}{2}\left(1 - \left(1 - \frac{\epsilon_p}{2}\right)^{\frac{1}{\lceil S \rceil}}\right)\right) \\ \Leftrightarrow\ & \lceil T \rceil \ge \frac{1}{-2\epsilon_d^2}\log_e\left(\frac{1}{2}\left(1 - \left(1 - \frac{\epsilon_p}{2}\right)^{\frac{1}{\lceil S \rceil}}\right)\right), \end{aligned} \tag{13}$$

which is true by definition of $T$.

□

We are now in the position to prove Theorem 3.1.

Proof of Theorem 3.1. We first prove the theorem for the case $c_f < \epsilon_d$ and then for $c_f \ge \epsilon_d$.
If $c_f < \epsilon_d$, every $x$ that we sample in the outer for-loop will have the property $0 \le c_x \le c_f < \epsilon_d$. Since the estimators $\hat{c}_x$ from the MPS algorithm always satisfy $\hat{c}_x \ge 0$ we can see that if also the inequality $|c_x - \hat{c}_x| \le \epsilon_d$ holds, then we have $|c_f - \hat{c}_x| \le \epsilon_d$. So if all $x$ that we sample in the outer for-loop satisfy $|c_x - \hat{c}_x| \le \epsilon_d$ (which is precisely event $A$ of Lemma 3.2), then $|c_f - \hat{c}_x| \le \epsilon_d$ holds for all sampled $x$ and hence also $|c_f - \hat{c}| \le \epsilon_d$. But according to Lemma 3.2 this event has probability $\mathbb{P}(A) \ge 1 - \frac{\epsilon_p}{2}$ and hence $\mathbb{P}(|c_f - \hat{c}| \le \epsilon_d) \ge 1 - \frac{\epsilon_p}{2} > 1 - \epsilon_p$. This proves the theorem for the case $c_f < \epsilon_d$.

For the case $c_f \ge \epsilon_d$ appart from the event

$$A := \text{"for all } x \text{ that we sample in the outer for-loop we have } |c_x - \hat{c}_x| \le \epsilon_d\text{ "} \tag{14}$$

we also consider the event

$$B := \text{"some of the } x \text{ that we sample in the outer for-loop satisfies } c_x = c_f\text{ "}. \tag{15}$$

Clearly $A \cap B$ implies $|\hat{c} - c_f| \le \epsilon_d$. Having seen in Lemma 3.2 that $\mathbb{P}(A) \ge 1 - \epsilon_p/2$ it suffices to prove $\mathbb{P}(B) \ge 1 - \epsilon_p/2$, because then

$$\mathbb{P}(A \cap B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cup B) \ge 1 - \epsilon_p/2 + 1 - \epsilon_p/2 - 1 = 1 - \epsilon_p. \tag{16}$$

Since $c_f \ge \epsilon_d$, the probability that a uniformly at random sampled $x \in \{0,1\}^n$ satisfies $c_x = c_f$ is at least $\epsilon_d$. So the probability that none of the $x$ that we sample in the outer for-loop satisfies $c_x = c_f$ is at most

$$(1 - \epsilon_d)^{\lceil S \rceil} \le (1 - \epsilon_d)^{\log_2(\epsilon_p/2)/\log_2(1-\epsilon_d)} = \frac{\epsilon_p}{2} \tag{17}$$

and hence $\mathbb{P}(B) \ge 1 - \epsilon_p/2$, which proves the theorem for $c_f \ge \epsilon_d$. The runtime of this algorithm can be found by multiplying the lengths of the 2 for-loops with the runtime of the computations that we do inside of the for loops. The latter is given by $\Theta(n + m + ET(f))$ where $n$ accounts for generating a random binary string of length $n$ and $m$ accounts for comparing 2 outputs. □

## 3.2 Experimental results and examples

We first show an example that illustrates a problem into which our algorithm can run and we explain how to solve it: When $c_f$ is very small, then due to the logarithmic relation between $c_f$ and the number of lines, estimation errors of $c_f$ can lead to large estimation errors of the number of lines.

**Example 1** If the MPS algorithm returns for $f : \{0,1\}^{10} \to \{0,1\}^6$ with $\epsilon_p = 0.01$ and $\epsilon_d = 0.1$ the value $\hat{c} = 0.65$, then with probability at least 0.99 we need $10 = \lceil \log_2(2^{10} \cdot (0.65 - 0.1)) \rceil = \lceil \log_2(2^{10} \cdot (0.65 + 0.1)) \rceil$ additional output lines to synthesize $f$. If $\hat{c} = 0.2$, we need between $7 = \lceil \log_2(2^{10} \cdot (0.2 - 0.1)) \rceil$ and $9 = \lceil \log_2(2^{10} \cdot (0.2 + 0.1)) \rceil$ additional output lines.

In general the amount of numbers of the form $\frac{1}{2^i}$ in our confidence interval equals the uncertainty about the number of lines that we need. If this number is too big we may run the algorithm again with a smaller $\epsilon_d$. The smaller the actual $c_f$ (and hence the expected $\hat{c}$) is, the more uncertainty about the number of lines we will have. However, since $c_f \geq 2^{-m}$ (this follows from the pigeon hole principle), the number of times that we have to repeat that process is bounded: Any interval of length $2 \cdot 2^{-(m+1)}$ which is contained in $(2^{-m}, 1]$, has at most one point of the form $\frac{1}{2^i}$. So choosing $\epsilon_d = 2^{-(m+1)}$ we are left with an uncertainty of at most 1 line.

**Example 2:** Let $f : \{0,1\}^{100000000} \to \{0,1\}^3$ defined by

$$f(x) = Dec2Bin\left(p\left(Bin2Dec(x)\right) \mod 8\right), \tag{18}$$

where $p(x) = x^3 + x$ is a polynomial. With some simple number-theoretic arguments one can check that $c_f = 3/8 = 0.375$: Since addition and multiplication commute with the mod operation, all inputs $x$ with $Bin2Dec(x) \mod 8 = 0$ are mapped by $f$ to $[0,0,0]$, those with $Bin2Dec(x) \mod 8 = 1, 2$ or $5$ are mapped to $[0,1,0]$, those with $Bin2Dec(x) \mod 8 = 3, 6$ or $7$ are mapped to $[1,1,0]$ and those with $Bin2Dec(x) \mod 8 = 4$ are mapped to $[0,1,0]$. Hence the outputs $[0,1,0]$ and $[1,1,0]$ have the largest preimages, each containing $\frac{3}{8}$-th of the input space.
This is a function of a size far beyond the scope of what any of the existing algorithms can compute. We ran the MPS-algorithm on $f$ for different values of $\epsilon_p$ and $\epsilon_d$ and summarized the results in Figure 3. With $\epsilon_d = 0.01$ and $\epsilon_p = 0.01$ we obtained $\hat{c} = 0.38081$ which indeed is 0.01-close to $c_f$. This means that to implement this function with a reversible circuit with a probability of at least 0.99 we need 100000002 lines. Also with all the other values of $\epsilon_p, \epsilon_d$ we have estimated the correct number of lines.

To compare the improvement of MPP estimates for a variation of the $(\epsilon_d, \epsilon_p)$ parameters, we estimated the MPP for the example function of Equation 18 as calculated from Algorithm 1. The resulting plot of Figure 3 shows how a decrease in the approximation parameter $\epsilon_d$ shifts the MPP estimates $\hat{c} = c_{\text{MPS}}$ $\epsilon_d$-close to the true value. Note how the security parameter $\epsilon_p$ only marginally influences this shift, up to sample-based fluctuations within the allowed margin. The corresponding sample sizes and number of function evaluations are listed in the lower part (b) of Figure 3.

Another parameter-dependent run-time comparison for the example function '$co14$' of the RevLib [6] benchmark library is given in Figure 4(b). Note how it is a worst-case embedding, where $k$ takes its maximal value $n$.

In the previous example we were able to determine the exact values of $c_f$, because we had knowledge about the algebraic properties of the function. Of course in such cases it would not be necessary to use our algorithm and we just used this trivial example to demonstrate the correctness of the algorithm. In the following example we will compute $c_f$ for a difficult function for which we do not know the exact values.

**Example 3:** Consider the function $f : \{0,1\}^{100000000} \to \{0,1\}^4$ that is defined by

$$f(x) = \text{Dec2Bin}(\#\text{OfDifferentPrimeFactors}(Sum(x))), \tag{19}$$

so that for example any input $x$ with $sum(x) = 60$, has $f(x) = [0,0,1,1]$, because $60 = 2*2*3*5$ has three different prime factors. Applying our algorithm with $\epsilon_d = 0.1$ and $\epsilon_p = 0.05$, we obtained $c_f = 0.36907$, meaning that for this function we need 100000003 lines.

Functions of the form $f(x) = a^x \mod N$ play a fundamental role in Shor's algorithm. In the next example we test MPS on such a function. Again we choose a simple example of such a function for which we know the correct value. We also view this function as a function which is defined on a finite set of natural numbers (and not binary strings).

**Example 4:** Consider the function

$$\begin{aligned} f :& \{1, 2, \dots, 2^{40} - 1\} \to \{1, 2, \dots, 2^{40} - 1\} \\ & x \mapsto 16^x \mod 2^{40} - 1. \end{aligned} \tag{20}$$

Here the correct value is again easy to see: Since this function is 10-periodic we have $c_f = 0.1$ and we need $\lceil \log_2(2^{40} \cdot 0.1) \rceil = 37$ garbage bits. We ran the MPS algorithm with $\epsilon_p = 0.1$ and $\epsilon_d = 0.1$ and obtained $\hat{c} = 0.12644$ resulting in 1 qubit more than optimal. For $\epsilon_p = 0.1$ and $\epsilon_d = 0.05$ we obtained $\hat{c} = 0.11605$, which gives us the correct minimal number of garbage bits $\lceil \log_2(2^{40} \cdot 0.11605) \rceil = 37$.

**Benchmarks:** We tested our algorithm on several irreversible RevLib benchmark functions and compared them to exact solutions of the MPP as given by the cube-based Sum-Of-Product (SOP) method or Binary Decision Tree (BDD) method from [5] and further also to an approach based on function matrices [9]. The number of additional lines $\hat{k}$ was correctly estimated for all functions with the parameter settings of only $(\epsilon_d, \epsilon_p) = (0.1, 0.1)$. For these parameters we obtained advantageous or comparable run-times, which are listed in the Table of Figure 4 (a).

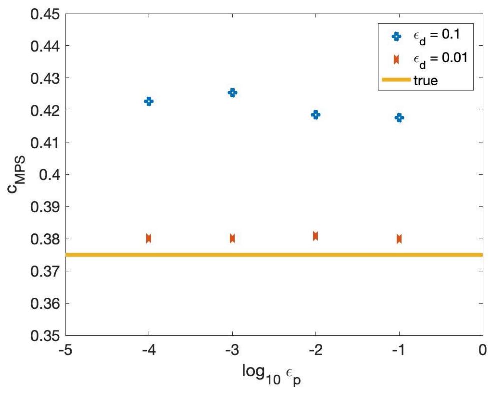
## 3.3 Implementation

All experimental results on benchmark functions were calculated on a 2.6 GHz Intel Core i7 with a main memory of 16 GB 2400 MHz DDR4 (Mac OS 10.14.1). We implemented and evaluated all benchmark functions in MATLAB® R2017b based on the circuit representation provided by RevLib. The code for both the algorithm and the benchmark functions are available on GitHub, as well as the .mat files of all evaluations.

## 4 CONCLUSIONS AND FUTURE WORK

We presented a probabilistic algorithm which efficiently approximates the minimal number of lines that are needed to implement a given function $f : \{0,1\}^n \to \{0,1\}^m$ in a reversible circuit. By sampling at random from the input space, the algorithm estimates how many inputs are mapped to the same value as a given $x$ and doing this for several $x$ the algorithm estimates the size of the largest subset of the input space on which $f$ is constant. On the other hand, determining the size of the largest set on which $f$ is constant is mathematically equivalent to determining the number of lines that are needed to implement $f$ with a reversible circuit. Using Hoeffding's inequality we are able to choose the lengths of the for-loops in such a way that the estimate of the algorithm differs from the correct value not more than $\epsilon_d$ with a probability of at least $1 - \epsilon_p$, where $\epsilon_d, \epsilon_p > 0$ are constants that can be chosen freely.

Our algorithm can be used in conjunction with existing methods that tackle the more general embedding (synthesis) problem [2], [9], [5]. Those often consist of two largely autonomous steps: First find the minimal number of lines or bits and secondly generate a cascade of gates or a function. Our algorithm can be used to do the first step more efficiently.



**(a)**

| $(\epsilon_d, \epsilon_p)$ | $c_{\mathrm{MPS}}$ | $S$ | $T$ | $\#func\_eval$ |
|---|---|---|---|---|
| (0.1,0.1) | 0.41761 | 29 | 352 | $10^4$ |
| (0.1,0.01) | 0.41851 | 51 | 497 | $2.5 \cdot 10^4$ |
| (0.1,0.001) | 0.42540 | 73 | 630 | $4.6 \cdot 10^4$ |
| (0.1,0.0001) | 0.42272 | 94 | 757 | $7.1 \cdot 10^4$ |
| (0.01,0.1) | 0.37999 | 299 | 46820 | $1.4 \cdot 10^7$ |
| (0.01,0.01) | 0.38081 | 528 | 61291 | $3.2 \cdot 10^7$ |
| (0.01,0.001) | 0.38015 | 757 | 74616 | $5.6 \cdot 10^7$ |
| (0.01,0.0001) | 0.38006 | 986 | 87452 | $8.6 \cdot 10^7$ |
| (0.001,0.01). | - | $7.3 \cdot 10^6$ | $5.3 \cdot 10^3$ | $3.9 \cdot 10^{10}$ |

**(b)**

**Figure 3: (a) MPS estimates for the embedding of example function (18) for different combinations of the parameters $(\epsilon_d, \epsilon_p)$, calculated from Algorithm 1. (b) Corresponding sample sizes and number of function evaluations.**

Our results raise many questions for future research: Can our probabilistic approach be extended to tackle the synthesis problem directly? Could derandomization techniques be used to make it into a deterministic method?

## A BINOMIAL DISTRIBUTIONS

A random variable $X$ with binomial distribution $B(n, p)$ takes values in $\{0, \ldots, n\}$ and has expected value $p \cdot n$. By Hoeffding's inequality [1] applied to $B(n, p)$ we have that for $\epsilon > 0$

$$\mathbb{P}\left(|\frac{1}{n} \cdot X - p| > \epsilon\right) \leq 2 \exp\left(-2n\epsilon^2\right). \tag{21}$$

## ACKNOWLEDGMENTS

Thanks to Amit Kulkarni for helpful discussions of LUT mappings. Thanks to Jakob Oesinghaus for discussing implementation issues. Many thanks to Ce Zhang for his support.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 programme (grant agreement DAPP, No. 678880).

## REFERENCES

[1] Wassily Hoeffding. 1963. Probability Inequalities for Sums of Bounded Random Variables. *J. Amer. Statist. Assoc.* 58, 301 (March 1963), 13–30. http://www.jstor.org/stable/2282952?

| Number of lines and runtime for embedding large f | | | | | | |
|---|---|---|---|---|---|---|
| benchmark | lines | | run-time [s] | | | |
| RevLib | | | ESOP | BDD | FM | MPS |
| [6] | (n,m, k) | ĉ | [5] | [5] | [9] | |
| co14 | (14,1,14) | 1.0 | 72.8 | 0.0 | 0.2 | 0.0375 |
| t481 | (16,1,16) | 0.70 | 1717.8 | 0.0 | 0.2 | 0.2084 |
| cmb | (16,4,16) | 1.0 | 7.1 | 0.0 | 0.2 | 0.0224 |
| pcler8 | (16,5,16) | 0.54 | 28.1 | 0.0 | 0.2 | 0.2199 |
| cm150a | (21,1,21) | 0.80 | >50000 | 0.1 | 2.7 | 0.2850 |
| mux | (21,1,21) | 0.81 | >50000 | 0.1 | 2.5 | 0.2938 |

**(a)**

| Probabilistic guarantees for the benchmark function co14 | | | | |
|---|---|---|---|---|
| $(\epsilon_d, \epsilon_p)$ | ĉ | k | $\hat{k}$ | run-time [s] |
| (0.1,0.1) | 1 | 14 | 14 | 0.0312 |
| (0.1,0.01) | 0.99799 | 14 | 14 | 0.0375 |
| (0.01, 0.1) | 0.99908 | 14 | 14 | 1.2367 |
| (0.01, 0.01) | 0.99935 | 14 | 14 | 1.5909 |
| (0.01, 0.001) | 0.99940 | 14 | 14 | 1.91843 |
| (0.001, 0.01) | 0.99928 | 14 | 14 | 320 |

**(b)**

**Figure 4: (a) Run-time comparison of MPS-estimated minimal number k of required garbage lines of irreversible n-to-m boolean functions (exact and MPS estimate), selected from RevLib [6]. The estimate $\hat{c} = c_{\mathrm{MPS}}$ was evaluated for all combinations of $\epsilon_d, \epsilon_p \in \{0.1, 0.01\}$. Here we only list $\hat{c}$ for $(\epsilon_d, \epsilon_p) = (0.1, 0.1)$, since all instances returned the correct number $k = \hat{k} = \lceil n + \log \hat{c} \rceil$ of additional lines. (b) Evaluation of different approximation guarantees $(\epsilon_d, \epsilon_p)$ for an arbitrary example function co14. Legend: ESOP: Exclusive-Or Sum-Of-Products, BDD: Binary decision diagram, FM : function matrices.**

[2] Dmitri Maslov and Gerhard W. Dueck. 2004. Reversible cascades with minimal garbage. *IEEE Trans. on CAD of Integrated Circuits and Systems* 23, 11 (2004), 1497–1509.

[3] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (Oct. 1997), 1484–1509. https://doi.org/10.1137/S0097539795293172

[4] Mathias Soeken, Robert Wille, Christoph Hilken, Nils Przigoda, and Rolf Drechsler. 2012. Synthesis of reversible circuits with minimal lines for large functions. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*. IEEE, 85–92.

[5] Mathias Soeken, Robert Wille, Oliver Keszocze, D. Michael Miller, and Rolf Drechsler. 2015. Embedding of Large Boolean Functions for Reversible Logic. *J. Emerg. Technol. Comput. Syst.* 12, 4, Article 41 (Dec. 2015), 26 pages. https://doi.org/10.1145/2786982

[6] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. 2008. RevLib: An Online Resource for Reversible Functions and Reversible Circuits. In *Int'l Symp. on Multi-Valued Logic*. 220–225. RevLib is available at http://www.revlib.org.

[7] Robert Wille, Oliver Keszocze, and Rolf Drechsler. 2011. Determining the minimal number of lines for large reversible circuits. *2011 Design, Automation and Test in Europe* (2011), 1–4.

[8] Robert Wille, Mathias Soeken, and Rolf Drechsler. 2010. Reducing the Number of Lines in Reversible Circuits. (2010), 647–652. https://doi.org/10.1145/1837274.1837439

[9] Alwin Zulehner and Robert Wille. 2017. Make it reversible: Efficient embedding of non-reversible functions. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*. 458–463. https://doi.org/10.23919/DATE.2017.7927033