Scalable Simplification of Reversible Circuits *

Vivek V. Shende, Aditya K. Prasad, Ketan N. Patel, Igor L. Markov and John P. Hayes
Department of EECS, University of Michigan, 1301 Beal Ave., Ann Arbor, MI 48109-2122
{vshende,akprasad,knpatel,imarkov,jhayes}@eecs.umich.edu

ABSTRACT

Reversible logic circuit synthesis has applications in various modern computational problems, low power design, and quantum circuit synthesis. Several algorithms for synthesis and simplification of reversible logic have been proposed recently; however, they tend to be infeasible for circuits of more than a handful of inputs. In our work, we examine scalable methods to reduce the gate count of a given reversible circuit. Theoretical considerations we take up suggest that *local optimization* – that is, the process of picking sub-optimal subcircuits, and replacing them with smaller counterparts – may be a fruitful approach. In practice, our methods work well on circuits with up to 30 inputs, and find reductions in gate count as large as 35% in randomly generated circuits. We conclude with an example of a circuit for which local optimization fails, and further directions for research.

1. INTRODUCTION

Many modern computational problems are inherently reversible in nature, meaning that information present in the input must be conserved by the computation and be recoverable from the output. Some fields in which such problems arise include cryptography, digital signal processing, and communications [8].

In addition, non-reversible circuits necessarily dissipate heat to compensate for the loss of information they incur [1]. It has been shown that some reversible circuits can be made asymptotically energy-lossless as their delay is allowed to grow arbitrarily large [15]. In fact, De Vos et al. have built reversible circuits of up to 384 transistors, powered only by the input. Figure 1 shows one of their circuits as seen by a scanning electron microscope [3].

Moreover, all quantum circuits are, by their nature, reversible. Purely quantum gates are necessary for the exponential speed-up enjoyed by many quantum algorithms, but generating classical reversible circuits is an important step toward quantum circuit synthesis. Many quantum computational applications call for large classical reversible sub-circuits: in particular, the textbook implementation of Grover's quantum search algorithm uses many CNT (CNOT, NOT, and TOFFOLI) gates [10].

*This work was partially supported by the Undergraduate Summer Research Program at the University of Michigan and by the DARPA QuIST program. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

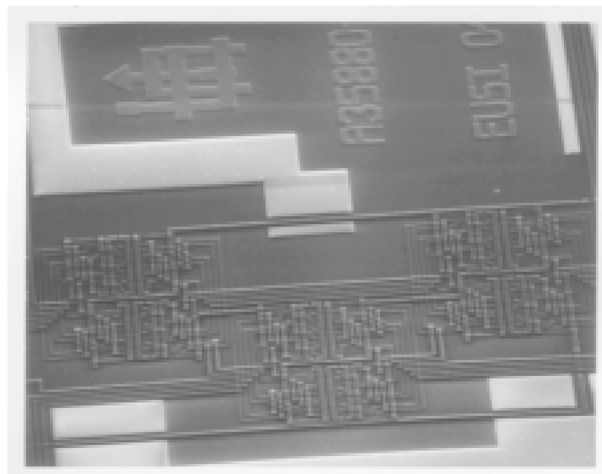
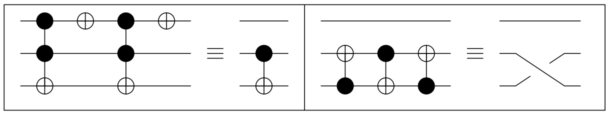


Figure 1: An image of a reversible CNT-circuit implemented in CMOS by De Vos et al. This particular circuit uses 144 transistors and no internal power supplies.

Some previous work has centered on generating circuits consisting entirely of CNT gates. Toffoli [13] showed that the CNT gate library is universal for the synthesis of reversible Boolean circuits. This has been recently extended, and in particular it has been shown that all even permutations can be synthesized with no temporary storage lines, and that odd permutations require exactly one extra line [12, 14]. Iwama et al. describe a simple but nontrivial set of local transformation rules for CNT-circuits [6]. Their work focuses on transforming circuits into a canonical form rather than with reducing circuit size, but indicate that the theory of reversible circuits would benefit from a more concrete heuristic for the latter.

Shende et al. describe a method for synthesizing optimal reversible circuits [12], which significantly outperforms the exhaustive search methods used by Kerntopf to tabulate statistics for small reversible circuits [7]. Still, even the improved algorithm was never called upon to synthesize circuits on more than three wires, and was never forced to output a circuit of more than twelve gates. It is evident that circuits on many more wires, containing many more gates, could not be generated in a reasonable amount of time by the algorithm presented there. They also suggest a heuristic which scales much better, but may yield very sub-optimal circuits.

The present work offers fast simplification of CNT-circuits rather than optimal methods that may not scale. We are interested in reducing the number of gates in a given circuit without increasing the number of bits on which the circuit operates – meaning that we allow no temporary storage, or constant inputs. This is motivated in part by the fact that in quantum computing applications, qubits are relatively expensive and gates are relatively cheap.



(a) (b)

Figure 2: Reversible circuit equivalences.

We use a greedy local search to find improvements that can be achieved by changing a few gates at a time while preserving the circuit function. Reversible circuits are particularly amenable to such optimization, which falls roughly into the paradigm of dynamic programming: large tables of circuit reduction rules that are precomputed by optimal methods [12] and can be stored in files. With appropriate data structures, optimization can be performed in linear time and is very fast in practice. It can be analogized with *peephole optimization* in compilers [9], where only several lines of code are optimized at a time.

The remainder of the paper is structured as follows. The necessary background is given in Section 2. Section 3 motivates the use of local optimization to reduce reversible circuits. Section 4 discusses the generation of circuit libraries for use in the reduction procedure, and Section 5 discusses the reduction procedure itself. We examine empirical results in Section 6 and some theoretical limitations to local optimization in Section 7. Finally, conclusions and on-going work are discussed in Section 8.

2. BACKGROUND

We are interested in reversible Boolean circuits. The condition for a (combinational) circuit to be reversible amounts to requiring each gate to be reversible, and requiring that no fan-out or feedback occur in the circuit. A reversible gate must compute a bijective function, and must therefore have the same number of inputs as outputs: this number is its width.

In other words: if we were to draw the circuit as a graph, with circuits as vertices, and directed edges representing wires (we allow more than one edge to connect two vertices), the graph is required to be acyclic, and every vertex must have as many edges entering as leaving. It follows that for any cross-section of the circuit, adding the number of wires cut to the widths of the gates cut yields a constant: its width. Clearly this is the same as the number of inputs to the circuit, or number of outputs.

Because the number of wires entering a gate is the same as the number of wires leaving, we may think of wires not as stretching merely from one gate to another, but going throughout the whole circuit, with gates appearing on them from time to time. Alternatively, we can think of bits in a register, on which we can perform reversible operations, but cannot physically move anywhere – this formulation corresponds to the quantum computing context.

2.1 The CNT Gate Library

The gates used in our reversible circuits are members of a larger family introduced by Toffoli [13]. The k-Controlled NOT gate, or k-CNOT, is a width $k+1$ gate. It leaves the first k inputs unchanged, and inverts the last iff all others are 1. The first three k-CNOT gates have special names. The 0-CNOT is just an inverter, or NOT gate (N). The 1-CNOT is just called a CNOT gate (C), and the 2-CNOT is called a TOFFOLI gate (T). These three gates comprise the CNT library, and a reversible circuit made up of only these gates is called a CNT-circuit. Our circuits will all be CNT-circuits: the CNT gate library is universal [13], and these particular gates appear often in the quantum computing context [10].

We draw CNT-circuits as arrays of horizontal lines representing wires, in which gates are represented by vertically-oriented symbols [4]. The $\oplus$ symbols represent inverters and the $\bullet$ symbols represent controls. A vertical line connecting a control to an inverter means that the inverter is only applied if the wire on which the control is set carries a 1 signal. The circuit in Figure 2(a) computes the same function as a single C gate. Such pairs of equivalent circuits are useful in optimization: we can replace the larger with the smaller to effect a circuit reduction.

2.2 Sub-circuits

Finding sub-circuits to reduce is central to the present work. We intuitively understand a sub-circuit to be a collection of gates with the property that we may "draw a box" around it and treat it as the inner workings of some larger gate. To make this precise, we consider the topological ordering of the gates in a circuit. For gates G,H in a circuit C, we say $H >_C G$ (or equivalently $G <_C H$) if there is some non-trivial path from H to G in the graph representing C. The $>_C$ relation is a strict partial order: from the definition of a path, it is clear that $F >_C G$, and $G >_C H$ imply $F >_C H$. In addition, since we require nontrivial paths, no gate follows itself and we cannot have both $G >_C H$ and $H >_C G$: otherwise, there would be a loop in the graph, which is required to be acyclic. But $>_C$ is not a total order: for example, in a two-wire circuit with one NOT gate on each wire, neither follows the other. Finally, we abbreviate $>_C$ to $>$ if it is clear which circuit we are talking about.

We say that a subset S of the gates in a circuit C is replaceable if, for all gates $G \in C$, and for all gates $F,H \in S, H > G > F \implies G \in S$. Given a replaceable subset of gates, their order relations, and which wires they operate on, we may recover all the details about how they are interconnected in the original circuit. We will henceforth understand that all gates "know" which wires they operate on; therefore, a sub-circuit is given by a replaceable subset of gates together with their order properties. We generally do not distinguish a sub-circuit from the underlying replaceable subset, and in fact denote them with the same letter: $S = (S, >_C|_S)$. Finally, a sub-circuit can be thought of as a circuit in its own right.

2.3 Circuit Encoding

It is not at all evident what sorts of data structures should be used to encode reversible circuits. Ideally, we want a data structure that captures the order properties of the circuit to be encoded. That is to say, we would like to encode a circuit C in a data structure D_C in such a way that the representation of the gate G occurs "after" the representation of a gate H iff $G > H$.

We can satisfy the "if" part of the condition by using an array like data-structure, for example, an STL `vector`. It suffices to proceed from the inputs of a circuit to the outputs, adding gates to the end of the array and removing them from the beginning of the circuit. When two gates are both situated at the inputs, we choose arbitrarily which to encode first. This procedure preserves the ordering: it is the text-book proof that a finite partially ordered set can be extended to a total order by depth-first search [2].

However, the result of the arbitrary choices made in the encoding procedure is that we cannot possibly ensure that the representation of G occurs after the representation of H only if $G > H$. In fact, it is clear even from the example of the two-wire circuit with a NOT gate on each wire that we cannot satisfy this condition in general: neither of these gates follows the other, but one must come first in the array.

We are left with a choice: either reject using the array in favor of something more complicated, or take pains to ensure that the arbitrary choice of encoding order does not affect the result of our algorithms. In this paper, we take the second option: the price is that locating all sub-circuits is a nontrivial task. However, the following characterization of sub-circuits is useful.

PROPOSITION 1. *A subset S of a circuit C is a sub-circuit if and only if there is an encoding E_C such that the gates from S form a contiguous block in E_C.*

Proof: The "if" part of the proposition is obvious. So suppose S is a sub-circuit of C. We claim that the set T, of gates in C which are neither in S nor follow any gate in S, forms a sub-circuit. Suppose $a, b \in T$, and $a < x < b$ for some gate x. If $x \in S$, we have $b > x$, which is a contradiction, and if for some gate $s \in S$, we have $x > s$, then $b > x > s \implies b > s$, which is again a contradiction. Let R be the set of gates that follow some gate in S, but are not in S; this is similarly a sub-circuit. We may consider R, S, T as circuits in their own right, and encode them in arrays E_R, E_S, E_T. We claim that the concatenation $E_C = E_T E_S E_R$ is a valid encoding for C.

To show this, we need to show that the order properties of C are preserved in E_C, and in fact it suffices to show the following claims: (1) No gate in S follows any gate in T, (2) No gate in R follows any gate in S, and (3) No gate in R follows any gate in T. Claims (1) and (2) are immediate from the definitions. Suppose some gate $r \in R$ is followed by some gate $t \in T$. By definition, there exists some $s \in S$ such that $t > s$, hence we have $r > t > s$, and r follows some gate in S, which is a contradiction. □

2.4 The Random Circuit Model

Reversible computation has yet to achieve sufficient popularity for benchmarking circuits to be publicly available. Moreover, we know of no tools to take a given permutation and return a reversible circuit computing it, although some methods of doing this have been suggested in the literature [12]. Therefore, in the interest of testing our methods, we generate random CNT-circuits, to simulate a simple but inefficient synthesis algorithm. The random circuit generating procedure constructs circuits sequentially by first selecting a gate type (C, N, or T) based on some probability distribution (p_C, p_N, p_T), and selecting wires for the gate inputs at random. The wires available for the next gate are the same as the wires for the previous, save that those wires which the previous gate took as inputs are replaced by wires corresponding to the outputs of the previous gate. For us, $p_C = p_N = p_T = \frac{1}{3}$.

We point out that the random circuit model should not be understood as representing circuits which occur in practice. Most functions require circuits of length $O(n2^n/log(n))$, for n the number of inputs [12]. It follows that most random circuits of this length will not be reducible to circuits of a reasonable length. Circuits used in practice, however, tend to be reducible to a non-exponential size. Therefore, while we use the random circuit model to give some preliminary data as to what sorts of reductions one can expect, the ultimate test of these methods needs to be against "real" circuits.

3. THEORETICAL MOTIVATION

Two consecutive inverters on the same wire can be canceled out. Similarly, if two CNOT gates appear consecutively on a pair of wires, and use the same wire for a control, they may be canceled out, and the same is true of matching TOFFOLI gates. These are the most elementary local reductions: clearly a single gate sub-circuit cannot be replaced with anything shorter, and it turns out that two gates can never be replaced by one. Before working out algorithms to deal with more complicated reduction schematics, let us determine how far cancellation alone can go. For the duration of this section, we consider cancellations on a long random circuit of k wires. We proceed through the circuit from inputs to outputs looking for duplicate gates that we can eliminate.

Let K be an arbitrary gate type which cancels itself out, like NOT, CNOT, or TOFFOLI. We are interested in the percent reduction that results from cancelling K-gates out. Because each cancellation removes 2 K-gates from the circuit, the percent reduction is twice the probability, $P(\text{red}_K)$, that a given K-gate is the first in a cancellation.

Let p_K be the probability that a given gate in the circuit is of type K. For an arbitrary K-gate A in the circuit, let P_K be the probability that a K-gate B we can cancel with A appears before an obstruction to cancellation. In general, P_K is a function of the probability distribution of the gates in the circuit. Clearly $P(\text{red}_K) < p_K P_K$, but this bound is not sharp: the first K-gate may have been eliminated in an earlier reduction. To eliminate this possibility, we ensure that the number of gates which precede the given gate and could cancel it is even. This probability is approximated by the following formula, which is precise in the limit of long circuits.

$$P(\text{red}_K) \approx p_K P_K (1 - P_K)(1 + P_K^2 + P_K^4 \cdots) = \frac{p_K P_K}{1 + P_K}$$

We now calculate this for the specific case of a CNT circuit, and for K=NOT, CNOT, and TOFFOLI. Let p_N, p_C and p_T denote the respective relative probabilities of these gates. A NOT gate may cancel with another NOT gate on the same wire so long as no gate between them is controlled on their wire. An inverter we may cancel appears with the same probability as a CNOT that obstructs cancellation, but TOFFOLI gates have two controls, so it is twice as likely that a TOFFOLI gate obstructs an inverter cancellation. Thus,

$$P_N = \frac{p_N}{p_N + p_C + 2 \cdot p_T}.$$

Similar calculations can be made for CNOT and TOFFOLI gates. One important difference is that for the NOT case we only had to consider the effect of a control occurring between two NOT gates; for the CNOT and TOFFOLI cases we also need to consider the effect of a target occurring between the gates' controls.

$$P_C = \frac{p_C}{p_N(k-1) + p_C(2k-2) + p_T(3k-5)}$$

$$P_T = \frac{p_T}{p_N(k^2 - 3k + 2) + p_C \frac{3k^2 - 11k + 10}{2} + p_T(2k^2 - 8k + 9)}$$

For equiprobable gate types ($p_C = p_N = p_T = 1/3$), we find that

$$P(\text{red}_N) \approx 1/15$$
$$P(\text{red}_C) \approx 1/(18k - 21)$$
$$P(\text{red}_T) \approx 2/(27k^2 - 99k + 102)$$

In contrast with the NOT case, the effectiveness of CNOT and TOFFOLI cancellations decreases as k increases. Intuitively this makes sense. Taking the example of the CNOT reduction, there is only one CNOT gate that can form a pair with another while there are $2k - 3$ CNOT gates that can eliminate the possibility of a pair, and it is correspondingly worse for TOFFOLI gates.

As two gates are eliminated each time we apply a reduction, the expected percentage reduction from NOT cancellations alone is $2/15 \approx 13.3\%$. Note that cancelling inverters out is a special case of local optimization: we know that certain gate configurations are equivalent to an empty circuit, so we replace them as we find them. There are many more such reductions – but in general, we replace more gates with less gates rather than two gates with none. To make better reductions, we pre-compute a library of optimal circuits, with which we replace any longer, equivalent circuits we find.

4. CIRCUIT LIBRARY GENERATION

Our method for local optimization relies on our ability to determine whether or not a given circuit or sub-circuit is optimal – that is, whether or not there is an equivalent circuit that employs fewer gates. Any circuits or sub-circuits which are not optimal are called sub-optimal. Any sub-circuit of an optimal circuit is optimal.

One can check if a given circuit C on n wires is optimal as follows: build a library L of all optimal circuits on n wires, and check if $C \in L$. In fact, L need only contain the optimal circuits smaller than

C: if any circuit in L computes the same function as C, then C is suboptimal, and if no circuit in L computes the same function on C, then C is optimal by definition. Running through all the circuits in the library, determining the functions they compute, and comparing each with C is time consuming. Instead, we store the library as an STL `hash_set` of circuits, indexed by the (pre-computed) function they compute. Two circuits can have the same index, but it suffices to store only one optimal circuit for each function.

In short, we are interested in building circuit libraries containing one representative optimal circuit for each function that may be computed in d or fewer gates on n wires. We call such a library an $\text{OCRL}(d,n)$ (optimal circuit representative library). Observe that the first $d-1$ gates of an optimal d-gate circuit themselves form an optimal sub-circuit. Therefore, to generate an $\text{OCRL}(d,n)$ from an $\text{OCRL}(d-1,n)$, it suffices to iterate through $(d-1)$-gate circuits from the $\text{OCRL}(d-1,n)$, and add single gates to the end of each in all possible ways. It now suffices to iterate through the resultant circuits, adding them to the OCRL iff they compute a function which no circuit currently in the OCRL computes. This ensures that only optimal circuits are added, and that only one optimal circuit computing a given function is added.

The approach described above was taken in [12]. The authors of that paper were interested in the problem of synthesizing optimal reversible circuits, which they did using a depth-first search algorithm accelerated by means of an OCRL. To illustrate their methods, they built an $\text{OCRL}(3,3)$, and proceeded to synthesize optimal circuits of up to eight gates on three wires for each of the $8! = 40,320$ reversible functions on three inputs. They claim that generating the $\text{OCRL}(3,3)$ takes a negligible amount of time, and that synthesizing the remaining functions takes 215 seconds.

Here, however, we are interested in optimal circuits on four wires rather than three, for reasons that are clarified in Section 5. The number of such functions is $(2^4)!$, which is greater than 20 trillion. Memory constraints allow us to generate only up to approximately 40 million; in the interest of time and disk space, we choose to limit ourselves to circuits of depth 6, of which there are approximately 26 million. The generation of the $\text{OCRL}(6,4)$ is fast; details are listed in Table 3. Because our library contains only circuits of up to six gates, we cannot always determine the optimality of a sub-circuit with more than six gates. This is a tradeoff we have to deal with because of memory constraints. In practice, however, we find that most circuits are heavily populated with sub-circuits that can be reduced to six or fewer gates.

Our ability to generate and store very many circuits depends on a compact storage method we devised specifically for this purpose. Each gate occupies just one byte using the following bit-packing method. Because we only implement three gates, the gate's type can be stored in two bits. Each of the gate's operands can be represented by two bits as well, since we are only operating on four wires. Since no gate operates on more than three wires, the entire gate takes just eight bits (for NOT and CNOT gates, the excess operands are just ignored). Our representation of circuits for generation also packs data efficiently. We store up to seven gates, along with the number of gates – this takes eight bytes.

We also store the function the circuit computes. A reversible circuit on n wires permutes its 2^n possible input vectors. Therefore, on four wires, we store the function as a permutation of 16 values. Each value requires 4 bits, and as there are 16 of them, the whole function is stored in a 64-bit variable of type `long long int`. We can quickly hash these values by adding the high 32 bits to the low 32 bits times a prime number.

In Table 3, we see that the time required to generate circuit libraries is nearly linear in the number of circuits: we generate about 200,000 circuits per second. This speed is in part attributable to the compactness of our data storage. We store each circuit is described

```
bool CAN_JOIN(subcirc S, circ C, gate g)
   for each h from g to S.pivot
     if !(S.contains(h) or CAN_SWAP(g,h))
       return false
   return true


list FIND_SUBCIRCS(gate PIVOT, circ C)
   subcirc S ← {PIVOT}
   list L ← {S}
   i ← 0

   while (L[i]!= NIL)
     S ← L[i]
     for each g not in S
       if ON_SAME_WIRES(S,g)
         if CAN_JOIN(S, C, g)
           S ← S + g
       else if TOTAL_WIRES(S,g) ≤ k
         if CAN_JOIN(S, C, g)
           T ← S + g
             L.append(T)
     i ← i+1

   return L
```

Figure 3: Pseudo-code for sub-circuit enumeration.

in only 16 bytes, which allows us to fit all 26 million or so depth-6 circuits in under half a gigabyte of memory. In [12], generating all functions on three wires and saving them to disk took 3.5 seconds; we require only 0.4 seconds. More importantly, their storage method could not accommodate such large libraries on four wires.

5. CIRCUIT REDUCTION

Our goal is to reduce CNT circuits with many gates and wires. To accomplish this, we traverse small sections of these circuits and optimize them sequentially. This approach is reminiscent of *peephole* optimization often employed by modern compilers, which simplifies small sections of code at a time [9].

We are interested in replacing sub-circuits with smaller equivalent circuits. But the number of sub-circuits of a given circuit is exponential in the number of gates in the circuit; moreover, many of these yield no fruitful reductions. Sub-circuits which are too short are often already optimal, whereas sub-circuits which are too long, while probably sub-optimal, may not have an optimal realization small enough to be found in a pre-generated circuit library. Moreover, our method of generating circuit libraries requires that we pick the number of wires in advance – that is, we need to choose the width of the sub-circuits we plan to examine.

We are now faced with the task of enumerating the sub-circuits of a given fixed width that appear in a circuit C, given the encoded circuit, E_C. Recall that sub-circuits are characterized by the property that they are contiguous in some encoding array, E'_C. Listing all encoding arrays E'_C would take a long time. Instead we look at the transformations by which E_C may be altered without changing the circuit it encodes, and enumerate sets of gates which can be made contiguous by a sequence of such transformations.

PROPOSITION 2. *If g,h share no wires and are consecutive gates in an encoding array, g does not follow h, and h does not follow g. We can swap them without changing the circuit represented by the encoding array.*

Given an encoding array, E, and a *pivot* gate, p, we can enumerate all maximal width-k sub-circuits, containing p, which can be made

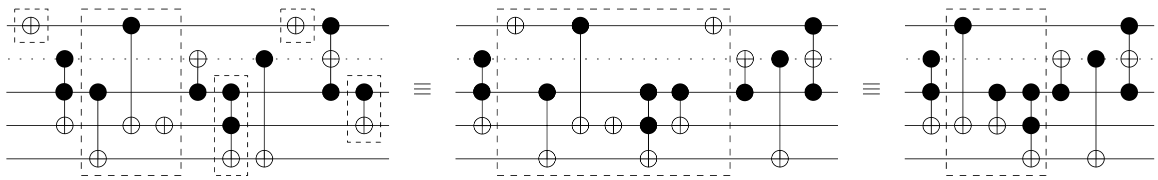


Figure 4: The highlighted gates in the left-most circuit form a sub-circuit on the solid wires. We can make them contiguous, as is shown in the middle circuit. This sub-circuit is suboptimal, and can be reduced, yielding the circuit on the right.

contiguous without changing the position of p in the encoding array. The algorithm proceeds as follows. Initialize L as an empty list of sub-circuits, and add to L the sub-circuit S_1 consisting of p alone. We now traverse the sub-circuits S of L, beginning with $S = S_1$.

For a given sub-circuit, S, with pivot p and right-most (in the encoding) gate g_r, we iterate through all gates from g_r to the end of the circuit. For each such gate g, we check first if the total number of wires used by g and by S exceeds the maximal width of sub-circuits we are interested in, k. If not, we check whether there are any gates x in the encoding array between p and g which satisfy $p < x < g$ but fail to be in S, otherwise we know that $S \bigcup g$ forms a sub-circuit. If S already operates on all the wires which g affects, then we add g to S and begin again with the gate to the right of g. If not, we form a new sub-circuit S', consisting of S and g, put it at the end of the list of sub-circuits, and continue looking for gates to add to S. Once we have finished looking between g_r and the end of the sub-circuit, we look between the leftmost gate g_l and the beginning of the sub-circuit, and when this is finished, we move on to the next sub-circuit in the list. By the end, we have a list of sub-circuits (L), each operating on a different set of wires, all employing P as their pivot. Pseudo-code is given in Figure 3.

Finally, note that in searching for sub-circuits, the only question we ever ask of two gates is whether they can move past each other in the encoding array. Originally, we were asking whether they could move past each other in the encoding array without changing the structure of the circuit represented. However, sometimes gates can be moved past each other in the circuit itself without changing the function being computed. In this case, we can also interchange them in the encoding array. We use commutability rules from the literature [12, Corollary 26]. An example is given in Figure 4

No matter how many sub-circuits are found, we may, in general, only optimize one of them. This happens because they may intersect in various ways, and after optimizing one, the others may no longer occur in the new circuit. In the instance that more than one of the sub-circuits we found are optimizable, we need a heuristic to decide which of these circuits we should optimize. We tried several different choices. Results appear in Section 6.

6. EMPIRICAL RESULTS

Our description of the algorithm in Section 5 leaves many parameters unspecified, and we would like to optimize them for both speed and performance by running empirical tests. Moreover, we would like to know what effect various factors such as circuit size, circuit library size, and limited gate libraries have on the efficacy of our algorithm. In this section, we investigate these questions.

6.1 Algorithmic Improvements

The discussion in Section 5 left unspecified various implementation details. These include: how far from the pivot gate we search for sub-circuits, how many sub-circuits we collect in the list before trying to optimize them, and how we choose which sub-circuit to optimize when we are ready to do so. Here, we implement and compare several different algorithms. Our first method, LocalA, is as follows. For each pivot, look for sub-circuits all the way from one end of the circuit to the other, collect them, reduce whichever one can be reduced by the largest amount, and then proceed to the next pivot.

We ran LocalA on random circuits with 5, 10, or 20 wires, and 250, 500, or 1000 gates. We found that the greatest reduction occurred on circuits with 5 wires. As the number of wires grows, this reduction factor decreases considerably. However, even on 20 wires, the program was able to reduce the gate count to 79.6% of its original value. The reductive efficiency was relatively constant as the initial gate count varied. Results are given in Table 1.

We made several modifications to this basic algorithm, both to improve its reduction ability, and to decrease runtime. One alternative: instead of reducing sub-circuits after each pivot, traverse the entire circuit to find the most reducible sub-circuit, reduce it, and continue. This variation took about 35 times longer than LocalA. Because it only made one reduction per pass through the circuit, it had to make very many passes before it could find no more. However, we found that it offered no reduction increase on average, and actually did worse in some cases. We tried the opposite: instead the most reducible sub-circuit, search for the least. This took as long without producing any benefits.

We did glean some interesting information from these versions, however. For the greedy version, we found that the maximum reduction almost always diminished with each pass. On circuits with 5 wires and 1000 gates, the first pass through often found reductions of up to 15 or 20 gates at a time. Within a few passes, however, it dropped below 10. On average, the algorithm made 100 passes through the circuit before it could not find any more reductions.

Based on the evidence gathered from the last two algorithms, we tried the following: instead of searching through all of the sub-circuits for a given pivot before reducing one, we immediately reduced the first one we could. We found that the great majority of pivots had no reducible sub-circuits; therefore, this improvement took away no ability in reduction. However, it produced nearly a 40% reduction in time.

The next improvement we found was even more significant. In LocalA, we pick a pivot and examine the circuit from end to end to find sub-circuit. This takes time at least proportional to the circuit size. We repeat this process for all gates in the circuit, so LocalA runs in $\Omega(n^2)$ time, where n is the number of gates. In Table 1, we see the impact this has on runtime. However, we know intuitively that the algorithm should not be worse than linear: we could run it

	Runtime of Local A, sec			Runtime of Local B, sec		
#gates	5 wrs	10 wrs	20 wrs	5 wrs	10 wrs	20 wrs
1000	3	20	106	1.7	3.8	10.1
500	1	9	45	.8	1.6	3.5
250	0	4	21	.2	.5	1.1
%Rdx	35.2	25.6	23.0	37.3	25.6	21.0

Table 1: Runtimes and performance of our algorithms. All tests performed on a 2GHz Pentium-4 Xeon workstation. Performance is measured in percent reductive efficiency, 100 **times the change in circuit size, divided by the original circuit size.**

on the first 100 gates, and repeat for the remaining sets of 100 gates, and should obtain similar results in linear time.

An empirical observation shows how to make this possible. Although gates which constitute a given sub-circuit may theoretically appear arbitrarily far away from each other in an encoding array, this is unlikely to happen in practice. Empirically, none of the sub-circuits on 5 to 20 wires ever extend more than 50 gates from the pivot. In fact, the overwhelming majority stay within 30 gates. We adjusted the loop accordingly: whereas before we examined all gates from the pivot to both ends, we now examine only those that are up to 30 gates away from it. As this yields significant savings in runtime while sacrificing little reductive efficiency, we use this modification in all later variants.

We tested the new algorithm (LocalB) on many circuits, of the same size we tested LocalA with. The improvements in runtime over LocalA were significant. LocalB was over 10 times faster for 20 wires and 1000 gates. More results are given in Table 1.

6.2 Reduction Versus Circuit Size

To determine the performance of LocalB on circuits of different widths, we tried varying the wire count from 5 wires to 30 wires. We ran it with circuits of varying sizes again, and averaged their reduction amounts. The times listed in Table 2 are for circuits of 250 gates. As shown by the table, the reduction ability of the algorithm becomes worse as circuit width increases, leveling to about 80% with 30 wires. Runtimes remain fairly low.

6.3 Reduction Versus Library Size

To test the efficacy of our circuit library, we created several more optimal libraries to test it against. We generated libraries with maximum depths ranging from 0 gates – the identity function alone – up to a depth of 6 gates – the ordinary size of our circuit library. Table 3 shows the sizes of the various gate libraries we tested. A depth of 6 was the greatest we could store in memory.

The runtime differences between different library sizes were negligible for both 5 and 10 wires. What's interesting is that the reduction performance degradation is very slight from depth 6 to depth 5, despite the fact that there are more than 10 times as many circuits of depth 6 as depth 5. A further look into the size distributions made the reason for this evident. On 10 wires, more than 99% of the sub-circuits found had gate counts less than 7, and 98% had fewer than 6. A depth 6 library, therefore, is sufficient for the large majority of sub-circuits we find.

The data for depth-0 circuits is also interesting. Because the only function computable with 0 gates is the identity function, gate cancellations account for almost all reductions which may be achieved with this library, and empirically, the vast majority of these were inverter cancellations. The program was able to reduce the gate count by about 12% in both cases, which almost matches the expected reduction value of 13.3% computed in Section 3. It falls short for two reasons: the thirty gate bound on the search distance from the pivot gate costs up to 2% of reductions, as can be seen in Table 1, and the value of 13.3% applied to arbitrarily large circuits.

6.4 Circuits with Restricted Libraries

In [12, Theorem 33], a constructive synthesis procedure is given to decompose an arbitrary permutation into a CNT-circuit. In fact, the resultant circuit breaks down into four sub-circuits, each of which has only one gate type: the first contains only TOFFOLI gates, the second only CNOTs, the third only TOFFOLIs, and the fourth only inverters. These sub-circuits contain up to $3(2^n - n)(3n-7)$, $n^2/\log n$, $3(2n+1)(3n-7)$, and n gates, respectively [12, 11]. We now examine how well local optimization works on circuits comprised only of TOFFOLI, or only of CNOT gates.

Input Circuits of Different Widths							
# wires	5	7	10	15	20	25	30
%Rdx	37.5	29.4	24.9	22.9	21.7	20.5	19.5
Time, sec	.2	.4	.6	1.0	1.5	1.9	3.1

Table 2: LocalB applied to circuits of various widths.

Circuit Libraries of Different Depths							
Depth	0	1	2	3	4	5	6
%Rdx, 5wrs	12.4	18.3	26.4	30.8	33.5	35.5	37.0
%Rdx, 10wrs	11.6	15.7	21.9	24.2	25.0	25.9	25.9
# Circuits	1	29	605	10K	158K	2.1M	26M
Time (sec)	0	0	0	0	1	10	152

Table 3: Characteristics of an OCRL(n,4) for various n: reduction efficiency on 5, 10 wires, number of circuits in the OCRL, and build-time.

First, we build an optimal circuit library specific to the problem, that is, consisting of only circuits that only use the gate in question. The sub-circuit width we fix at 4 wires. For CNOT gates, we can store the full OCRL(9,4), capturing all 20160 functions computable with CNOT gates on 4 wires. For TOFFOLI gates, we store 20 million circuits, which turns out to be halfway between the OCRL(9,4) and the OCRL(10,4). The first of these libraries took less than a second to build, while the second took approximately 5 minutes. Experiments were performed on a circuit on 5 wires and containing 1000 gates. On average, 25.6% of the CNOT only circuit remained, and 88.6% of TOFFOLI gates remained.

We can compute how much of the reduction was likely due to gate cancellations by the methods in Section 3. For a CNOT only circuit, we have $p_N = p_T = 0$ and $p_C = 1$, hence $P(\text{red}_C) = \frac{1}{2k-1}$. We expect CNOT cancellations to remove $\frac{2}{2k-1} = 22.2\%$ of the original gates. Therefore, 52.2% of the CNOT gates were removed by more complicated reductions. On the other hand, we would expect that an average of 10% of the original gates in a random TOFFOLI only circuit would cancel, so only 1.4% of the original TOFFOLI gates were removed by more complicated reductions.

7. LIMITATIONS

Our empirical results for random circuits show that significant improvements are possible using local optimizations, however there are some theoretical limitations to this approach. We know that any irreducible CNT-circuit can have no reducible sub-circuits. We can also show that the converse is not true in the following strong sense.

PROPOSITION 3. *For any d there is a reducible CNT-circuit with depth $\geq d$ and no reducible proper sub-circuits.*

Proof: The proof is by construction. For a given d we first construct the circuit shown in the shaded box in Figure 5 with $k = d$. This CNT-circuit has k gates and depth d. It computes a function that changes the values of k wires, since each wire has exactly one CNOT that acts on it. Since the circuit has only k gates, it must be irreducible; otherwise there would exist a CNT-circuit that used fewer than k gates to modify the values of k wires, which is not possible since each CNT-gate has only one target. Because the circuit is irreducible, it follows that its sub-circuits must also be irreducible.

Now we repeat the pattern in our construction, adding one gate at a time, as shown on Figure 5, and stopping once we have a reducible circuit. This circuit can have no reducible sub-circuits, because of the cyclic structure of the construction. In particular, suppose the sub-circuit formed between gate numbers i and j inclusively was reducible, then this would imply that the sub-circuit formed by the first $j-i+1$ gates was also reducible since the two sub-circuits are identical up to a relabeling of the inputs. However, this would be a contradiction since we stopped adding gates as soon as we had a reducible circuit. Therefore, we have constructed a reducible circuit

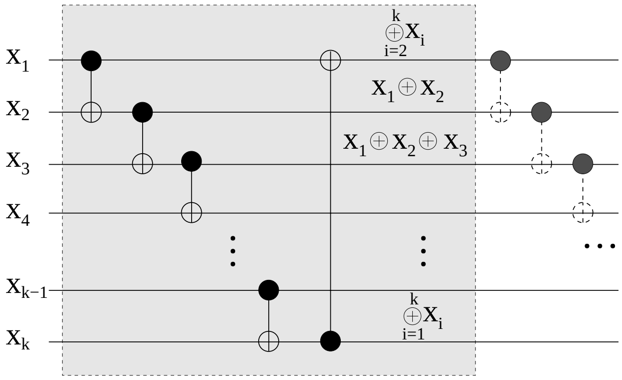


Figure 5: Example showing that for any d there is a reducible CNT-circuit with depth$\geq d$ and no sub-optimal proper sub-circuits.

with depth $\geq d$, that has no reducible proper sub-circuits. □

One consequence of this proposition is that no matter how large a library of local reducibility relations we have, there are circuits that cannot be reduced using our local reductions. However, we note that the above example is only guaranteed to use $k+1$ gates, where k is the number of wires. In fact, because the CNOT is the only gate used, the circuit cannot have length greater than $O(k^2)$ [12]. On the other hand, one can repeat this construction with the TOFFOLI gate, and it is not obvious that the circuit constructed need be short – but it is also not obvious that the circuit constructed need be long.

8. CONCLUSIONS & FURTHER WORK

We have shown that local optimization is a scalable tool for reversible logic circuit optimization. Inverter cancellations alone can provably achieve up to a 13% reduction, and large tables of circuit equivalences empirically result in reductions of up to 37% for five-wire circuits. Even on circuits with 30 inputs and a thousand gates, runtimes are measured in tens of seconds.

However, random circuits may not be representative of reversible circuits relevant to applications. The real test of local optimization will come only after known reversible logic synthesis algorithms – which take a permutation and return a circuit computing it – are implemented. It may be the case that synthesis tools will produce circuits which are unsuitable for random optimization; however, this seems unlikely given the fact that known algorithms produce exponentially long circuits even when shorter are possible [12].

Directions for further work in local optimization of reversible circuits fall into three general headings.

Storage of the optimal circuit library. The need for a larger gate library is best evidenced by the poor performance of our symbolic reduction algorithm on TOFFOLI-only circuits: most reductions were trivial gate cancellations. The problem is that since a TOFFOLI gate occupies 3 wires by itself, it is hard to find sub-optimal 4-wire sub-circuits. However, it is currently impossible to store a useful 5- or 6-wire optimal circuit library. One idea to improve storage is to store optimal circuits up to relabelling of wires: this would reduce memory requirements by a factor of $k!$ for circuits of width k. However, care will have to be taken to avoid increasing runtime by the corresponding amount. Another idea to extend our circuit libraries is to store sub-optimal circuits which cannot be reduced by local optimization, and their optimal counterparts.

Improving sub-circuit enumeration. Our current sub-circuit enumeration misses some sub-circuits. However, a more exhaustive enumeration would require more subtle data-structures. Additionally, our ability to find sub-circuits depends on our knowledge of commutability rules: as it is, we only know one [12, Corollary 26]. It would be advantageous to know more. Moreover, given that commutability seems central to finding sub-circuits, perhaps we should (1) extend our gate library to include more commutable gates, and (2) in the optimal circuit library, prefer more commutable sub-circuits.

Non-local optimizations. Two consecutive CNOT gates occurring on the same wires with the same orientation cancel out, but if they occur with opposing orientations, they may instead be replaced by a single CNOT and a wire swap. Wire swaps may all be pushed to the end of the circuit; doing so introduces no new gates. Moreover, because any permutation of n wires may be accomplished in $n-1$ transpositions, at most $n-1$ swaps are required in all, each of which cost 3 CNOT gates. Novel techniques for CNOT-circuit synthesis can be applied at this point for further potential optimizations [11]. The methods of Section 3 indicate that the reduction achieved using this technique will be comparable to that offered by canceling CNOT gates. We note that this is not a local optimization: it collects CNOT gates from all over the circuit, groups them into wire swaps, and moves the wire swaps to the end of the circuit, where they can be cancelled. There may be similar groupings of larger numbers of gates that also allow for non-local optimization.

9. REFERENCES

[1] C. Bennett, "Logical Reversibility of Computation," *IBM J. of R. & D.*, **17**, 1973, pp. 525-532.
[2] T. Cormen et. al., *Introduction to Algorithms, 2nd ed.*, The MIT Press, 2001.
[3] B. Desoete and A. De Vos, "A Reversible Carry-Look-Ahead Adder Using Control Gates," Integration, the VLSI Journal, **33**, 2002, pp. 89-104.
[4] R. Feynman, "Quantum Mechanical Computers," *Optics News*, **11**, 1985, pp. 11-20.
[5] L. K. Grover, "A Framework For Fast Quantum Mechanical Algorithms," Symp. On Theory of Computing, 1998.
[6] K. Iwama et al., "Transformation Rules For Designing CNOT-based Quantum Circuits," DAC 2002, pp. 419-425.
[7] P. Kerntopf, "A Comparison of Logical Efficiency of Reversible and Conventional Gates," IWLS 2000, pp. 261-269.
[8] J. P. McGregor and R. B. Lee, "Architectural Enhancements for Fast Subword Permutations with Repetitions in Cryptographic Applications," ICCD, 2001, pp. 453-461.
[9] W. McKeeman, "Peephole Optimization," *Communications of the ACM*, **8**, July 1965, pp. 443-444.
[10] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, 2000.
[11] K. N. Patel et al., "Efficient Synthesis of Linear Reversible Circuits," 2003. `http://xxx.lanl.gov/abs/quant-ph/0302002`
[12] V. V. Shende et al., "Synthesis of Reversible Logic Circuits," to appear in *IEEE Trans. on CAD*, 2003. `http://xxx.lanl.gov/abs/quant-ph/0207001`
[13] T. Toffoli, "Reversible Computing," *Tech. Memo MIT/LCS/TM-151*, MIT Lab for Comp. Sci., 1980.
[14] A. De Vos, B. Raa, and L. Storme, "Generating the Group of Reversible Logic Gates," Journal of Physics A: Mathematical and General, **35**, 2002, pp. 7063-7078.
[15] S. Younis and T. Knight, "Asymptotically Zero Energy Split-Level Charge Recovery Logic," *Workshop on Low Power Design*, 1994.