Data Structures and Algorithms for Simplifying Reversible Circuits [*]

Aditya K. Prasad[†], Vivek V. Shende, Ketan N. Patel[‡], Igor L. Markov and John P. Hayes
[†] Amazon.com Inc., 12th Ave South, Suite 1200, Seattle, WA 98144-2734
[‡] Qualcomm Inc., 645 Campbell Technology Parkway, Suite 200, Campbell, CA 95008
Department of EECS, University of Michigan, 2260 Hayward, Ann Arbor, MI 48109-2121
{akprasad,vivek.vijay.shende}@gmail.com,
kpatel@qualcomm.com, {imarkov,jhayes}@eecs.umich.edu

Abstract

Reversible logic is motivated by low-power design, quantum circuits and nanotechnology. We develop a compact representation of small reversible circuits to generate and store optimal circuits for all 40,320 three-input reversible functions, and millions of four-input circuits. This allows implementing a function optimally in constant time for use in peephole optimization of larger circuits produced by existing techniques, and guarantees that every three-bit subcircuit is optimal. To generate subcircuits, we use a graph-based data structure and algorithms for circuit restructuring. Finally, we demonstrate a suboptimal circuit for which peephole optimization fails.

Category: B.6.3 Design Aids, B.6.1 Design Styles
Terms: reversible logic, logic synthesis & optimization, algorithms, data structures
Keywords: circuit simplification, optimal subcircuit, circuit libraries

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 where such problems arise include cryptography, digital signal processing, and communications [14, 20].

Irreversible circuits necessarily dissipate heat to compensate for the loss of information they incur [3]. Recent work from Intel [25] derives physical limits for irreversible computation by systems that use electrons and energy barriers to store and manipulate binary values.

[*]This work was supported by the Undergraduate Summer Research Program at the University of Michigan, by the DARPA QuIST program and by the NSF. The views and conclusions contained herein are those of the authors and should not be interpreted as representing official policies or endorsements of employers or funding agencies.

Due to power-density constraints, these limits are likely to halt transistor shrinkage for all major circuit types and technologies by 2021. A potential solution is to recycle information and energy. To this end, reversible and asymptotically energy-lossless circuits have been proposed [24], but their delay can be arbitrarily large. Independently, De Vos et al. have built several reversible circuits of up to 384 transistors, powered only by their input signals. Figure 1 shows one of their circuits as seen through a scanning electron microscope [8].

Another novel computing technology that circumvents physical limits cited in [25] — quantum circuits, — also requires complete reversibility. Quantum circuits and algorithms offer additional benefits in terms of asymptotic runtime. While purely quantum gates are necessary to achieve quantum speed-up, variants of conventional reversible gates are also commonly used in quantum algorithms [2]. For example, the textbook implementation of Grover's quantum search algorithm uses many NCT (NOT, CNOT, and TOFFOLI) gates [19]. Hence, efficient synthesis with such gates is an important step toward quantum computation.

Toffoli [22] showed that the NCT gate library is universal for the synthesis of reversible Boolean circuits. This has been recently extended to show that all even permutations can be synthesized with no temporary storage lines, and that odd permutations require exactly one extra line [20, 23]. Optimal circuits for all three-bit reversible functions can be found in several minutes by dynamic programming [20]. This algorithm also synthesizes optimal four-bit circuits reasonably quickly, but does not scale much further. More scalable constructive synthesis algorithms [18, 1, 12] tend to produce suboptimal circuits even on three bits, which suggests iterative optimization based on local search.

The work in [11] describes a small set of local transformation rules for NCT-circuits, but focuses on transforming circuits into a canonical form rather than on reducing circuit size. A circuit simplifier proposed in [15, 16] tries to match a small set of pre-computed reducible circuit templates to subcircuits. Even with such simplification techniques, the only work to report optimal circuits for all three-bit functions so far is [20]. Our work extends such optimal methods to postprocess circuits produced by other techniques, so as to guarantee optimal gate counts in all three-bit subcircuits, and a large number of four-bit subcircuits. This can be contrasted with template-based simplification, which does not guarantee optimal subcircuits, but rather seeks to reduce larger sub-circuits that could be beyond the reach of optimal methods. As our empirical results show, the two approaches are complementary and should be employed together.

The method we present reduces the cost of a given circuit, e.g., the number of gates, without increasing the number of circuit inputs (no temporary storage or constant inputs). This is motivated in part by quantum computing applications, where qubits are relatively expensive and gates are relatively cheap. We also develop an exceptionally compact bit-based storage mechanism for small optimal circuits that supports finding an optimal implementation for a given function in $O(1)$ time. Such look-up libraries scale to millions of optimal circuits and can be constructed with surprising efficiency. When traversing a given large reversible

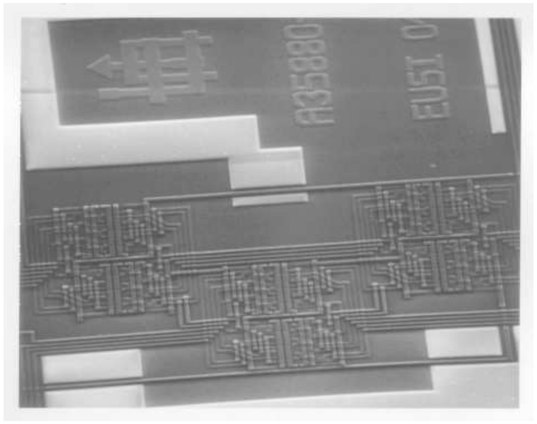

Figure 1: A reversible NCT-circuit with 144 transistors and no power rails, implemented in static CMOS by De Vos et al [8]. Simplifying such a circuit would reduce the required space, whereas in quantum-computing technologies the targets for improvement would be timing, the locality of qubit couplings, the probability of decoherence, and the required degree of error correction.

circuit, optimal libraries allow one to verify that the three- and four-bit subcircuits are implemented optimally. When a suboptimal subcircuit is found, it is replaced by an equivalent optimal subcircuit from the library. Such improvements alter only a few gates at a time, while preserving the overall circuit function. They can also be batched so that the overall runtime scales linearly, making it possible to reduce even very large circuits. With appropriate data structures, this technique is very fast in practice. Unlike previous work, it does not leave out any suboptimal three-bit subcircuits. Analogy can be made with *peephole optimization* in compilers [17], where only a few lines of code are optimized at a time. While such methods are well-known for simplifying irreversible circuits (e.g., the LSS system from IBM [7]), our approach can restructure a reversible circuit on the fly in order to find larger reducible subcircuits.

The remainder of the paper is structured as follows. The necessary background is given in Section 2. Section 3 introduces our compact data structure for reversible circuits. This data structure is used in Section 4, which discusses the generation of circuit libraries for use in the reduction procedure. Section 5 discusses the reduction procedure itself. We examine empirical results in Section 6 and exhibit theoretical limitations to peephole optimization. Conclusions are given in Section 7. The Appendix provides further arguments to support our empirical observations.

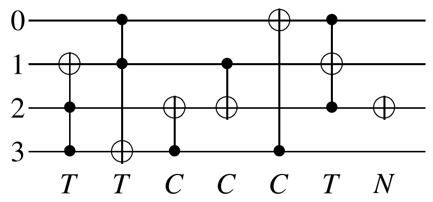

Figure 2: A reversible circuit of width four composed of seven NCT gates.

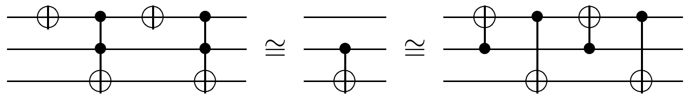

Figure 3: Sample reversible circuit equivalences.

2 Background and Basic Definitions

A logic gate is reversible if and only if it computes a bijective function. Therefore a reversible gate must have the same number of inputs as outputs, which we call its width, and the same applies to reversible circuits. In a (combinational) reversible circuit, all gates are reversible, and there is no fanout or feedback.

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

The NCT Gate Library. The gates used in our reversible circuits are members of a larger family introduced by Toffoli [22]. The k-Controlled NOT gate, or k-CNOT, has width $k+1$. 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 an inverter, or NOT gate (N). The 1-CNOT is called simply a CNOT gate (C), and the 2-CNOT is called a TOFFOLI gate (T). These three gates comprise the NCT library, and a reversible circuit with only these gates is called a NCT-circuit. Our circuits will all be NCT-circuits: the NCT gate library is universal [22], and these gates appear often in quantum computation [19].

We draw NCT-circuits as arrays of horizontal lines representing wires, in which gates are represented by standard symbols [9]. The $\oplus$ symbols represent inverters and the $\bullet$ symbols represent controls. A vertical line connecting controls to an inverter means that an inversion is applied iff all control lines carry 1s. A sample circuit containing NCT gates is depicted in Figure 2. We use the $\cong$ symbol to indicate that two circuits compute the same function. For example, the left- and right- hand circuits in Figure 3 both compute 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.

Representing Circuits by Graphs. One can model gates in a reversible circuit by vertices, and wires by directed edges (more than one edge may connect two vertices). If the gates are reversible, each vertex must have as many edges entering as leaving. Since no feedback is allowed, such graphs must be acyclic. The graph of a reversible circuit can be viewed as a partial ordering of gates: for gates G,H in C, we say $G >_C H$ (or equivalently $H <_C G$) if there exists a non-trivial path from H to G in the graph representing C. When C is clear from the context, we write $G > H$.

3 Data Structures for Representing Reversible Circuits

In electronic design automation, it is common to represent logic circuits as graphs or hypergraphs. However, we found that the regularity and ordering intrinsic to reversible circuits facilitate a more compact array-based representation (encoding) that is also more convenient for our purposes. In conventional circuit representations, all connections between individual gates are enumerated, and each gate stores indices of its incident connections. However, in a reversible circuit, one can distinguish a small number of wires going all the way through the circuit. In our encoding of a reversible circuit, those indices are maintained at individual gates, and the gates are stored in an array in an arbitrarily chosen topological order. Overlaid on this array is a redundant adjacency data structure (a graph) that allows one to look up the neighbors of a given gate. This representation is faithful; it is also convenient because each range in the array represents a valid sub-circuit. However, not every valid sub-circuit is represented by a range. In particular, any set of gates in a circuit that form an anti-chain (with respect to the partial ordering) will be ordered, obscuring the fact that any subset is a valid subcircuit.

Example 1 *To encode the circuit in Figure 2, we number wires top-down from 0 to 3. Then the gates can be written as* $T(2,3;1)T(0,1;3)C(3;2)C(1;2)C(3;0)T(0,2;1)N(2)$. *The two gates* $T(0,1;3)$ *and* $C(1;2)$ *form a subcircuit (contiguous block), but do not form a range in the array.*

In Section 5 we develop algorithms for generating valid subcircuits, including the two-gate subcircuit from the previous example. These algorithms are based on the following characterization.

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

Section 4 describes algorithms to generate vast libraries of optimal reversible circuits based on the NCT gate library. To extend the practical scale of these libraries, we propose a compact bit-packing scheme that is also runtime-efficient. In the particularly convenient special case of 4 wires, each gate occupies just one byte. The gate type is stored in two bits since there are three options.[1] Each of the gate's operands is also stored in two bits, since each circuit operates on four wires. Since no gate in our library acts on more than three wires at a time, an entire gate can be encoded in *eight bits* (for NOT and CNOT gates, the excess operands are just ignored). Leveraging our array-based representation of reversible circuits, this allows one to encode any 4-wire NCT-based circuit with up to seven gates in only eight bytes, including the number of gates in the first byte.

Example 2 *The circuit from Figure 2 and Example 1 is encoded with 7 (size) in the first byte, and the remaining seven bytes encoding each gate individually. For example, the second byte corresponds to the gate $T(2,3;1)$ and starts with bits* 10 *for gate type. The remaining six bits in the second byte encode the three wire indices 2, 3 and 1. Other gates are encoded similarly, with code* 00 *for inverters and code* 01 *for C gates.*

Expressing such 64-bit representations using type `long long int`, enables very efficient handling on modern 32-bit and 64-bit workstations. We also store the function computed by the circuit. A reversible circuit on n wires permutes its 2^n possible input vectors. A permutation of 16 values can be captured with 4 bits per value, which we pack in two 32-bit integers to allow efficient hashing.[2] We multiply the low 32 bits by a prime number and add the result to the high 32 bits. According to our computational experience, less-optimized data structures may undercut the performance of library generation algorithms in Section 4 by more than a hundred times, and increase memory usage by more than 10 times.

4 Algorithms for Generating Optimal Circuit Libraries

Our circuit optimization relies on determining if a selected subcircuit can be re-implemented at lower cost. Here circuit cost is calculated as the sum of gate costs, where all gates of the same type contribute equally. Such circuit cost functions imply that any subcircuit of an optimal circuit is optimal, and our algorithms use this property.

For notational convenience, we describe below a special case where all gate types have equal costs. The more general case of potentially different gate costs requires that libraries should be expanded according to increasing circuit cost rather than increasing gate count. In particular, we never store optimal circuit costs, therefore considering floating-point gate costs will not affect the compact circuit encoding introduced in Section 3.

[1]If needed, one more gate can be added, e.g., the Fredkin gate or the SWAP gate.

[2]Hashing optimal circuits by their functions allows one to quickly check if a given trial subcircuit is optimal. For this, multiply out the gates of the trial subcircuit and look up the resulting function in an optimal circuit library. Then price the optimal circuit found and compare to the cost of the trial subcircuit.

Our algorithms build circuit libraries containing one representative optimal circuit for each function that may be computed in d or fewer gates on n wires. We term such a library an *optimal circuit representative library*, denoted $\mathrm{OCRL}(d,n)$. Observe that the first $d-1$ gates of an optimal d-gate circuit themselves form an optimal subcircuit. Therefore, to generate $\mathrm{OCRL}(d,n)$ from $\mathrm{OCRL}(d-1,n)$, we iterate through all $(d-1)$-gate circuits from $\mathrm{OCRL}(d-1,n)$, and add single gates to the end of each in all possible ways. If such a circuit computes a function that is not represented in the OCRL, we add it to the OCRL. Thus, only optimal circuits are added, at most one per function. The techniques described so far were first proposed in [20] to synthesize optimal reversible circuits using a depth-first search algorithm, accelerated by means of an OCRL. As an illustration, the authors of [20] built $\mathrm{OCRL}(3,3)$ and used it to synthesize optimal circuits on three wires for each of the $8! = 40,320$ reversible functions on three inputs. Generating $\mathrm{OCRL}(3,3)$ takes a negligible amount of time.

Using an optimal circuit library on four wires can lead to additional reductions, but there are $(2^4)!$, or more than 20 trillion, such functions. Memory constraints allow us to generate only about 40 million. Since there are approximately 26 million circuits of depth 6, we stop the generation algorithm at $\mathrm{OCRL}(6,4)$. Runtimes are reasonable, as shown in Table 3, but because of the limited library size our algorithm cannot always find and reduce suboptimal subcircuits with more than six gates. In practice, however, we observe that many circuits can be reduced to optimal by only considering subcircuits with six gates or fewer.

Table 3 shows that the time needed to generate circuit libraries is nearly linear in the number of circuits with about 200,000 circuits generated per second. This speed is due in part to the compactness of our data storage. Since each circuit is stored in only 16 bytes, all 26 million or so depth-6 circuits fit in under half a gigabyte of memory. In [20], generating and saving all functions on three wires takes 3.5 seconds; we require only 0.4 seconds.[3] More importantly, their storage method cannot accommodate such large libraries on four wires.

5 Algorithms for Reducing Reversible Circuits

To reduce NCT-circuits, we traverse small sections of these circuits and optimize them one by one. We say that a subset S of the gates in a circuit C is *replaceable* if, for any pair of gates $F,H \in S$ and for any gate $G \in C$, the relation $H > G > F$ implies that $G \in S$. Given a replaceable subset of gates, their order relations, and which wires they operate on, we can determine how they are interconnected in the original circuit — this completely determines a subcircuit. For example, the sets of gates highlighted in Figure 5 (b) and (c) are replaceable, but the one in Figure 5 (a) is not.

We now enumerate subcircuits of a given width in a circuit C, given the encoding array E_C (illustrated in Example 1). Recall that each subcircuit must be contiguous in some encoding

[3]All runtimes are for a single-processor implementation running on a 2GHz Pentium-4 Xeon workstation.

```
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


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

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

    return L
```

Figure 4: Pseudocode for subcircuit enumeration.

array E'_C, but trying all encoding arrays is impractical. Instead we make use of the overlay graph data structure (as explained earlier) and also consider all transformations that alter E_C without changing the function of the encoded circuit, and enumerate sets of gates that can be made contiguous by a sequence of such transformations.

Proposition 2 *If gates g, h share no wires and are consecutive in an encoding array, then in the original circuit g does not follow h, and h does not follow g. Swapping g and h will not affect the functionality of circuit.*

Proposition 3 *Assume that gates g, h are drawn from the NCT library and share some wires, but control wires of one gate are not connected to the target wires of the other gate. Then swapping g and h will not affect the functionality of the circuit.*

Given an encoding array E and a *pivot* gate, p, we can enumerate all maximal width-k subcircuits, containing p, that can be made contiguous without changing p's position in the encoding array. The algorithm proceeds as follows. Initialize L as an empty array of subcircuits, and add the one-gate subcircuit consisting of p to it. Now for each subcircuit S in L, iterate through all gates g from the right-most (in the encoding) gate of S, to the end of the circuit. For each gate g, check first if the total number of wires used by g and S together exceeds the maximal width k. If not, check if there are any gates x in the encoding array

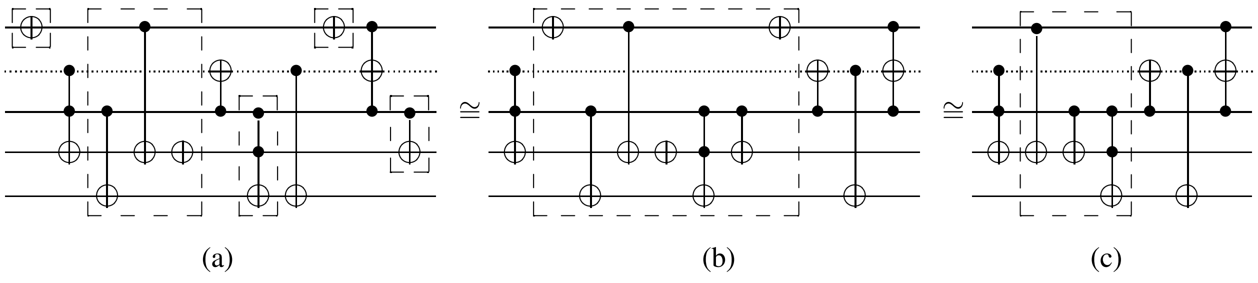

Figure 5: (a) The highlighted group of gates operates on the solid wires, (b) these gates can be consolidated into a contiguous subcircuit by reordering, (c) the subcircuit can be replaced by a smaller equivalent implementation.

between p and g satisfying $p < x < g$, but not in S, otherwise we know that $S \bigcup g$ forms a subcircuit. If S already operates on all the wires that g affects, then add g to S and begin again with the gate to the right of g. If not, form a new subcircuit S', consisting of S and g, put it at the end of the array, and continue looking for gates to add to S. After we search from the right-most gate to the end of the subcircuit, we search from the left-most gate to the beginning of the subcircuit. Then we consider the next subcircuit in L. At the end we have an array of subcircuits (L), each operating on a different set of wires, all with pivot p. Full pseudocode is presented in Figure 4, and the choice of the constant DISTANCE is discussed in Section 6. Note that calls to CAN_JOIN are expensive and therefore postponed as late as possible.

Finally, note that our algorithm is based on checking if two gates can move past each other in the encoding array without affecting circuit function. One way to ensure the function is not affected is to preserve the circuit structure. However, often gates can be moved past each other in the circuit itself without changing the computed function. In this case, we can also interchange them in the encoding array. We find such possibilities using commutativity rules from [20, Corollary 26]. Figure 5 shows an example.

6 Empirical Results and Implementation Details

In this section we highlight several implementation issues and empirically study the scalability of proposed algorithms as well as the impact of library size on performance.

Algorithmic details We now discuss how far to search from the pivot gate (DISTANCE) while locating subcircuits (Figure 4). For comparison, we first try DISTANCE = infinity; we call this version of the algorithm LocalA. We ran LocalA on randomly-generated circuits with 5, 10, and 20 wires, and 250, 500, and 1000 gates. While cost reductions achieved on such benchmarks are not representative of results on more structured circuits (discussed below), this experiment confirms the scalability of our techniques in terms of runtime and the

```
OPTIMIZE(circ C)
   bool FOUND_REDUCTION ← false
   gate CIRCUIT_POS ← C.first_gate
   while (CIRCUIT_POS != C.end)
     array SUBCIRCS ← FIND_SUBCIRCS(CIRCUIT_POS, C)
     if (SUBCIRCS != NIL)
       subcirc S ← CHOOSE_BEST_REDUCTION(SUBCIRCS, C)
       MAKE_CONTIGUOUS(S, C)
       permutation P ← COMPUTE_FUNCTION(S)
       subcirc R ← OCRL.find(P)
       REPLACE_SUBCIRCUIT(S,R,C)
       FOUND_REDUCTION ← TRUE
       CIRCUIT_POS ← R.first_gate
     else if ((CIRCUIT_POS = C.end) and FOUND_REDUCTION)
       CIRCUIT_POS ← C.first_gate
     else
       CIRCUIT_POS ++
```

Figure 6: Pseudocode for circuit reduction. OCRL is a pre-computed hash table of optimal circuit representatives, in which a circuit is hashed by the function it computes; see Section 4.

amount of reduction.

The greatest reduction occurred on circuits with 5 wires, and decreased significantly as the number of wires was increased. However, even at 20 wires, the program was able to reduce the gate count to 79.6% of its original value. The relative reduction was similar for circuits of very different gate counts. Results are given in Table 1.

As iterations are repeated for all gates in the circuit, LocalA runs in $\Omega(n^2)$ time, where n is the number of gates. In Table 1, we see the impact on runtime. We may obtain linear runtime by choosing any fixed value for DISTANCE. Table 1 compares LocalA to a DISTANCE = 30 implementation, LocalB, and shows only minimal performance degradation. However, LocalB was over 10 times faster for circuits on 20 wires with 1000 gates.

Reduction Versus Circuit Width. While evaluating the performance of LocalB on circuits with 5-30 wires, we ran it on inputs with a range of sizes. The average reduction percentages are shown in Table 2. The times listed in the table are for circuits with 250 gates. The reduction percentage decreased as circuit width increased, leveling to about 80% for 30 wires. Runtimes remained fairly low.

Reduction Versus Library Size. To test the efficacy of our circuit library, we compared it with several other optimal libraries with maximum depths ranging from zero gates — the identity function alone — up to a depth of 6 gates — the ordinary size of our circuit library and the largest we could store. The runtime differences for different library sizes were negligible for both 5 and 10 wires. Other empirical data are given in Table 3. Interestingly the amount of reduction degrades very slightly 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. To this end we observe that on 10 wires, more than 99% of the subcircuits found had fewer than 7 gates, and 98% had fewer than 6. Therefore, a depth-5 library is good enough for most subcircuits we find, and maintaining a depth-6 library should be more than sufficient.

The results for depth-0 are also interesting. Because the only function computable with zero gates is the identity function, gate cancellations account for almost all reductions, and empirically, most are inverter cancellations. The gate count is reduced by about 12% in both cases, almost matching the expected reduction of 13.3% computed in the Appendix. It is smaller for two reasons: (i) the 30-gate limit on the search distance from the pivot gate costs up to 2%, as seen in Table 1, and (ii) the analytical result assumes arbitrarily large circuits.

Circuits with Restricted Libraries. In [20, Theorem 33], a constructive synthesis procedure is given to decompose any permutation into an NCT-circuit consisting of four subcircuits, each with only one gate type: TOFFOLI, CNOT, TOFFOLI, and NOT, respectively. These subcircuits contain up to $3(2^n - n)(3n - 7)$, n^2, $3(2n+1)(3n-7)$, and n gates, respectively [20]. We now examine how well local optimization works on circuits with only TOFFOLI or only CNOT gates.

First, we build an optimal circuit library with circuits that only use the gate in question. We fix the subcircuit width to 4. For CNOT circuits, we can store the full OCRL(9,4) with all 20,160 4-wire CNOT functions. For TOFFOLI circuits, we store 20 million circuits, which is halfway between the sizes of OCRL(9,4) and OCRL(10,4). The first library takes less than 1 sec to build, and the second takes about 5 min. For 5-wire, 1000-gate random circuits, on average 25.6% of each CNOT circuit, and 88.6% of each TOFFOLI circuit remain after optimization.

We can estimate how much of the reduction is due to gate cancellations by the methods in the Appendix. For an all-CNOT circuit, we have $p_N = p_T = 0$ and $p_C = 1$, hence we expect cancellations to remove $2/(2k-1) = 22.2\%$ of the original gates. Thus, 52.2% of the CNOT gates were removed by more complicated reductions. On the other hand, we expect approximately 10% of the original gates in a random TOFFOLI only circuit to cancel, so only 1.4% of the original TOFFOLI gates were removed by more complicated reductions.

Results for Synthesized Benchmarks. We now test local optimization on circuits produced by constructive synthesis algorithms that take a permutation as input and return a circuit computing it. Such reversible synthesis algorithms are fairly recent [20, 18].

Shende et al. give a synthesis algorithm that takes an even permutation as input and yields an NCT-implementation [20]. Their algorithm decomposes the permutation into disjoint pairs of 2-cycles that are then individually implemented, mostly with TOFFOLI gates. Thus, the resulting circuits are dominated by TOFFOLI gates, which as we illustrated in our results for random circuits may be less amenable to local optimization than those with more CNOT and NOT gates. However, many TOFFOLI gates in the algorithm can be replaced by NOT and CNOT gates. For example, the first step of implementing a disjoint pair of 2-cycles in the algorithm is to transform the first index to $10\cdots0100$ using at most n TOFFOLI gates; this

	Runtime of Local A (sec)			Runtime of Local B (sec)		
No. of 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
%reduction	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. %reduction is measured by dividing the change in circuit size by the original circuit size.

Input Circuits of Different Widths							
No. of wires	5	7	10	15	20	25	30
%reduction	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 d	0	1	2	3	4	5	6
%reduction, 5 wires	12.4	18.3	26.4	30.8	33.5	35.5	37.0
%reduction, 10 wires	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: Properties of an OCRL(d,4) for various d: reduction efficiency on 5, 10 wires, number of circuits in OCRL, and build-time.

can be done with at most n NOT gates. In the second step, instead of using up to n TOFFOLI gates to transform the second index to $10\cdots01$ (without affecting the previously transformed index), we can do the same with n CNOT and two NOT gates. Most of the remaining steps can be similarly simplified to use fewer TOFFOLI gates. We call this modified algorithm Synthesis 1. Miller et al. [18] give another synthesis algorithm that we refer to as Synthesis 2. These two synthesis techniques are used to illustrate our optimization only —- we do not intend to compare them to each other or perform comprehensive empirical studies on reversible synthesis.

Benchmark	Synthesis 1 [20]			Synthesis 2 [18]			Best result	
Function	—	LocB	Tmpl	—	LocB	Tmpl	Method	size
hwb4	116	60	76	24*	22	23*	2L	22
hwb5	394	196	277	135*	134	116*	2TL	115
hwb6	894	526	670	539*	533	468*	2TLTL	457
mod5	56	21	22	9	8	9	2L	8
bch7	3504	1867	2429	38	36	36	2L,2T	34
rd53(1)	776	424	553	804	804	700	1L	424
rd53(2)	2748	1062	461	62	60	57	2LT	50

Table 4: NCT gate counts for circuits synthesized using two existing algorithms and postprocessed by LocalB and Templates. Both Synthesis 2 and Templates return circuits with generalized TOFFOLI gates that we decompose into NCT circuits. In cases labeled with asterisks such decomposition requires an extra wire.

We synthesize several benchmark functions using algorithms Synthesis 1 and 2, and then apply two optimization algorithms: Local B and template-based optimization from [18, 15]. Both Synthesis 2 and template-based optimization can yield circuits with generalized TOFFOLI gates that act on more than three wires; using the best known decompositions [2] we break down these larger gates into 3-input TOFFOLI gates. However, in some cases the generalized TOFFOLI gates span all wires, and thus an additional wire is needed to decompose the original gate.

For our first set of benchmarks we use a reversible generalization of the hidden weighted bit (hwb) function; the input word is circularly shifted by its weight, i.e., the number of 1s it has. We synthesize this function for input sizes of 4, 5 and 6 bits. The next benchmark, mod5, is a 5-wire Grover oracle circuit [19, 20] that takes the first four wires as inputs leaving their values unchanged at the output while inverting the value of the last wire if the first four wires represent an integer divisible by 5. The next benchmark, bch7, is a 7-input function that decodes a 7-bit single error-correcting BCH code [13, pp. 23–26]. The final set of benchmarks are versions of rd53, a 7-bit encoder function [18] that uses the first five wires as inputs and the last three as outputs with one shared bit. The output is the binary representation of the number of ones on the inputs. The difference between the two versions of rd53 is simply the ordering of the output wires. All benchmarks can be found online at

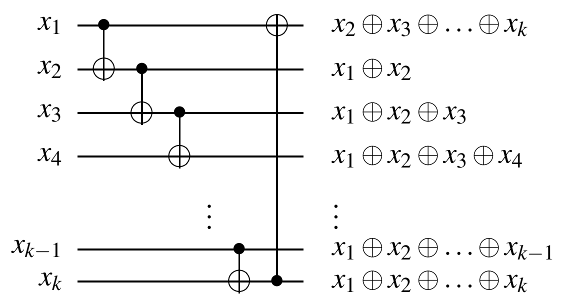
x_1
x_2
x_3
x_4
x_{k-1}
x_k
$x_2 \oplus x_3 \oplus \ldots \oplus x_k$
$x_1 \oplus x_2$
$x_1 \oplus x_2 \oplus x_3$
$x_1 \oplus x_2 \oplus x_3 \oplus x_4$
$x_1 \oplus x_2 \oplus \ldots \oplus x_{k-1}$
$x_1 \oplus x_2 \oplus \ldots \oplus x_k$

Figure 7: An optimal circuit used to illustrate limitations of peephole optimization.

`http://www.cs.uvic.ca/~dmaslov/`

In Table 4, the last two columns show *how* the shortest circuit was produced for each benchmark, and its size. For example, the best result for hwb5 was found by using Synthesis 2, applying the templates technique, then applying Local B. In most cases Synthesis 2 contributed to best results when used with template-based optimization and LocalB. For synthesized benchmarks, a combination of Local B and template-based optimization achieved an overall 18% reduction.

Limitations of Peephole Optimization. As seen from our empirical results, peephole optimization shows significant improvements when applied to both structured and randomly-generated circuits. However this approach also has limitations. While no subcircuit of an optimal NCT-circuit can be reduced, the converse is false.

For example, consider the k-gate, depth-k NCT-circuit shown in Figure 7. The function it computes modifies each bit. Therefore, if it is computed with gates from the NCT gate library, at least one "$\oplus$" symbol must appear on each wire, hence at least k gates are required. The circuit given is optimal, hence its subcircuits must also be optimal. Now suppose we continue the pattern, adding one gate at a time, and stopping once the circuit first becomes suboptimal for the function it computes. We note that no suboptimal subcircuits are formed: if the subcircuit from gate i to gate j (inclusive) is suboptimal, then the subcircuit formed by the first $j-i+1$ gates is also suboptimal, since the two subcircuits are identical up to a relabeling of the inputs. This cannot be, as we stop adding gates upon first finding a reducible circuit.

Proposition 4 *For any d there is a reducible NCT-circuit with depth $\geq d$ and no reducible proper subcircuits.*

Proposition 4 implies that no matter how large our library of local reductions may be, some circuits cannot be reduced by local optimization. The above example is only guaranteed to use $k+1$ gates, where k is the number of wires. However, other longer constructions with this property may exist using TOFFOLI gates.

7 Conclusions

We have described a compact storage mechanism for synthesizing reversible circuits that exploits libraries containing optimal reversible circuits. These libraries play a role similar to those of end-game databases in computer chess — they can be generated once and for all, and allow very efficient look-up. They can also be accessed in constant time by the functions they implement. We show that on a modest workstation, it is possible to generate optimal circuits for all 40,320 reversible functions on three bits in a blink of an eye. This is in contrast to recent publications [18, 16, 1, 12] which report suboptimal synthesis on three bits, typically requiring more time.

We developed a new use for large libraries of optimal reversible circuits — iterative local optimization of larger circuits. In particular, if our method is applied after those in [18, 16, 1, 12], it will guarantee optimal results for all three-bit functions. Indeed, we consider all possible three-bit subcircuits and replace those that are suboptimal with equivalent optimal variants from the library. Given that we can also store and use millions of optimal four-bit circuits, it would be very difficult for a human circuit designer to improve the resulting circuits. We have shown empirically that such optimization scales well for large reversible logic circuits. Runtimes are in seconds even for circuits with 30 inputs and 1000 gates, and local optimization is equally fast when applied to both randomly generated and synthesized circuits. Our current techniques do not exhaust all available memory, but we have not been able to find noticeable improvements by storing larger libraries. We have shown that the additional use of template-based simplification improves results, however templates do not require much memory and do not need to be traded off for libraries.

References

[1] A. Agrawal and N. K. Jha, "Synthesis of Reversible Logic," *Proc. DATE* 2004, pp. 1384-1385.

[2] A. Barenco et al., "Elementary Gates for Quantum Computation", *Physical Review A* **52**, 1995, pp. 3457-3467.

[3] C. Bennett, "Logical Reversibility of Computation," *IBM J. of R. & D.*, **17**, 1973, pp. 525-532.

[4] J. Cong, M. Romesis, and M. Xie, "Optimality, Scalability and Stability Study of Partitioning and Placement Algorithms," *Proc. ISPD* 2003, pp. 88-94.

[5] T. Cormen et al., *Introduction to Algorithms, 2nd ed.*, MIT Press, 2001.

[6] J. Darnauer and W. W. Dai, "A Method for Generating Random Circuits and its Application to Routability Measurement,"*Proc. FPGA* 1996, pp.66-72.

[7] J. A. Darringer, D. Brand, J. V. Gerbi, W. H. Joyner Jr., L. Trevillyan, "LSS: A System For Production Logic Synthesis," *IBM Journal of Research and Development*, vol. 28, no. 5 (September 1984), pp. 537-545.

[8] B. Desoete and A. De Vos, "A Reversible Carry-Look-Ahead Adder Using Control Gates," *Integration*, **33**, 2002, pp. 89-104.

[9] R. Feynman, "Quantum Mechanical Computers," *Optics News*, **11**, 1985, pp. 11-20.

[10] L. K. Grover, "A Framework For Fast Quantum Mechanical Algorithms," *Proc. STOC*, 1998.

[11] K. Iwama et al., "Transformation Rules For Designing CNOT-based Quantum Circuits," *Proc. DAC* 2002, pp. 419-425.

[12] P. Kerntopf, "A new heuristic algorithm for reversible logic synthesis," *Proc. DAC* 2004, 834-837.

[13] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, 1977.

[14] J. P. McGregor and R. B. Lee, "Architectural Enhancements for Fast Subword Permutations with Repetitions in Cryptographic Applications," *Proc. ICCD*, 2001, pp. 453-461.

[15] D. Maslov, G. W. Dueck, D. M. Miller, "Simplification of Toffoli Networks via Templates," *Symp. on Integrated Circuits and System Design* 2003, Sao Paulo, Brazil, pp. 53-58.

[16] D. Maslov, G. W. Dueck, D. M. Miller, "Fredkin/Toffoli Templates for Reversible Logic Synthesis," *Proc. ICCAD* 2003, 256-261.

[17] W. McKeeman, "Peephole Optimization," *Communications of the ACM* **8**, 1965, pp. 443-444.

[18] D. M. Miller, D. Maslov and G. W. Dueck, "A Transformation Based Algorithm for Reversible Logic Synthesis," *Proc. DAC* 2003, pp. 318-323.

[19] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, 2000.

[20] V. V. Shende et al., "Synthesis of Reversible Logic Circuits," *IEEE Trans. on CAD*, vol 22(6), June 2003, p. 710-722.

[21] D. Stroobandt, P. Verplaetse, J. Van Campenhout, "Towards synthetic benchmark circuits for evaluating timing-driven CAD tools," *Proc. ISPD* 1999, pp. 60-66.

[22] T. Toffoli, "Reversible Computing," *Tech. Memo MIT/LCS/TM-151*, MIT Lab for Comp. Sci., 1980.

[23] A. De Vos, B. Raa, and L. Storme, “Generating the Group of Reversible Logic Gates,” *Journal of Physics A*, **35**, 2002, pp. 7063-7078.

[24] S. Younis and T. Knight, “Asymptotically Zero Energy Split-Level Charge Recovery Logic,” *Workshop on Low Power Design*, 1994.

[25] V. V. Zhirnov et al., “Limits to Binary Logic Switch Scaling — A Gedanken Model,” *Proc. IEEE* **91**, 2003, pp. 1934-1939.

Appendix: Analytical Statistics for Random Circuits

Classical VLSI circuits are sometimes modeled by randomly-generated synthetic netlists that preserve the same statistical characteristics [6, 21]. Indeed, having families of increasing circuits with similar structure is particularly convenient when estimating the scalability of optimization heuristics and available room for improvement [4]. However, faced with the lack of comparable studies on the statistical behavior of practical reversible circuits, we are going to use uniform distributions as a first-order approximation and a likely optimistic upper bound. To evaluate the scalability of our reduction techniques, we construct a random circuit in steps, by first selecting a gate type (N, C or T) based on some probability distribution (p_C, p_N, p_T). The wires for the gate inputs are then chosen at random. The wires available for the following gate are the same, except that those at the inputs of the current gate have been replaced by those at its outputs.

Each NCT gate is its own inverse, therefore two identical gates appearing next to each cancel In this section, we analytically estimate the rates of such cancellations in long random circuits on k wires, assuming a single pass through the circuit from inputs to outputs.

Let K be an arbitrary idempotent gate type, such as NOT, CNOT, or TOFFOLI. Since two gates are eliminated with each cancellation, the reduction in the size of the circuit resulting from K-gate cancellations is $2 \times P(\text{red}_K)$, the probability that a given K-gate appears in the first position among cancelling gates. Let p_K be the probability a given gate in the circuit is of type K. We now view gates one by one from left to right and consider them as random variables. For an arbitrary K-gate in the circuit, let P_K be the probability that another K-gate that can cancel it appears before other gates can block the 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 tight: the first gate may have been eliminated in an earlier reduction. To eliminate this possibility, we ensure that the number of gates that precede the first gate and could cancel it is even. This probability is approximated by the following formula.

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

We now calculate this for the specific case of a NCT-circuit, and for NOT, CNOT, and TOFFOLI (viewed as K). Let p_N, p_C and p_T denote the respective relative probabilities of these gates. Two NOTs on the same wire can cancel if no gate between them is controlled by that wire. A NOT that cancels appears with the same probability as a CNOT that obstructs cancellation; since a TOFFOLI gate has two controls, it is twice as likely to obstruct the cancellation. Thus,

$$P_N \quad = \quad p_N/(p_N + p_C + 2 \cdot p_T).$$

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.

$$\begin{aligned} P_C &= \frac{p_C}{p_N(k-1) + p_C(2k-2) + p_T(3k-5)} \\ P_T &= \frac{p_T}{p_N(k^2-3k+2) + p_C\frac{3k^2-11k+10}{2} + p_T(2k^2-8k+9)} \end{aligned}$$

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

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

In contrast with the NOT case, the reduction from CNOT and TOFFOLI cancellations decreases as k increases. Intuitively this makes sense. Consider CNOT reductions: only one CNOT gate can form a pair with another while $2k-3$ CNOT gates can eliminate the possibility of a pair; it is correspondingly worse for TOFFOLI gates.

The expected percentage reduction from NOT cancellations alone is $2/15 \approx 13.3\%$. Note this is a special case of local optimization: certain gate configurations are equivalent to an empty circuit, and so can be eliminated. There are many more such reductions—but in general, we replace more gates with fewer rather than two gates with none. To make greater reductions, we pre-compute a library of optimal circuits, that we use to replace any longer, equivalent circuits we find.