Reversible Logic Synthesis Through Ant Colony Optimization

Min Li, Yexin Zheng, Michael S. Hsiao and Chao Huang
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061, USA
{minli, yexin, mhsiao, chaoh}@vt.edu

Abstract—We propose a novel synthesis technique for reversible logic based on ant colony optimization (ACO). In our ACO-based approach, reversible logic synthesis is formulated as a best-path search problem, where artificial ants, starting from their nest (reversible function output), attempt to find the best path to the food source (reversible function input). The experimental results have demonstrated superior performance in terms of both synthesis quality and computation time. They also show that the proposed method is scalable in handling large reversible functions.

I. INTRODUCTION

Reversible logic has demonstrated promise in realizing zero-energy-consumption circuit design, quantum computation and nanotechnology [1, 2, etc]. Reversible logic was first related to energy when Landauer states that information loss due to function irreversibility leads to energy dissipation [3]. This principle is further supported by Bennett that zero energy dissipation can be achieved only when the circuit contains reversible gates [4].

Unfortunately, synthesis for conventional Boolean logic cannot be directly applied to reversible logic for the following intrinsic reasons: a) reversible primitive gates have an equal number of inputs and outputs; and b) fan-outs and feedback are not allowed in reversible logic, which leads to implementations in the form of cascaded reversible gates. Two major categories of reversible logic synthesis methods have been previously proposed. One of them is the exact approach which provides the optimal solution yet suffers from long computation time, although it can be achieved only when the circuit contains reversible gates [4].

Accordingly, for gate $g$, its functionality $\bar{g} = f_g(\bar{x})$ can be expressed as:

$$y_i = \begin{cases} x_t \oplus \bigwedge_{k=1}^{n} z_{2i-1} \& z_{2i}, & 1 \leq i \leq n, i \neq t \\ x_t, & i = t \end{cases}$$

This work is supported by NSF grant 0840936
Hence, \( g(\bar{c}, t) \) represents all the generalized Toffoli gates with the number of control bits no more than \( n \). For instance, suppose the first Toffoli gate \( TOF_2(x_1, x_3) \) in Fig. 1(b) is to be used in a different \( n \)-input reversible logic. This \( TOF_2(x_1, x_3) \) can be represented as \( g([0, 2, 2, \ldots, 2], 3) \) which indicates a target bit of \( x_3 \) and a positive control bit of \( x_1 \) in an \( n \)-bit reversible logic network. Similarly, the second Toffoli gate, \( TOF_1(x_2) \) (NOT gate), can be formulated as \( g([2, 2, \ldots, 2], 2) \), and the last \( TOF_3(x_2, x_3, x_1) \) can be represented as \( g([2, 0, 0, 2, \ldots, 2], 1) \) in any arbitrary \( n \)-bit reversible logic circuit.

\[
z_i = \begin{cases} x_i \oplus c_i, & \text{if } c_i \neq 2 \\ 1, & \text{otherwise} \end{cases} \tag{2} \]

**Definition 4:** For an \( n \)-bit reversible function \( f \), an initial pheromone map can be represented as a graph \( G_r(V, E) \), named \( \tau_{Graph} \), with the vertex set \( V = P \) and the edge set \( E = \{e_{pq} | p, q \in V\} \). For each edge \( e_{pq} \in E \), a set of pheromone values \( \tau_{pq} \) are established for the ants’ transition decisions. We define \( \tau_{pq} = \{\psi_{pq}, \phi_{pq}\} \), where \( \psi_{pq} = \{\psi_{pq}(t) | t = 1, 2, \ldots, n\} \) and \( \phi_{pq}(t, i, c_i) \) denotes the amount of pheromone for selecting bit \( t \) as the target bit if an ant wants to travel from state \( p \) to state \( q \), while \( \phi_{pq}(t, i, c_i) \) denotes the amount of pheromone to set control bit \( i \) as value \( c_i \) for target bit \( t \), and \( c_i \) is the control value of bit \( i \) in Definition 2.

For instance, Fig. 2(a) illustrates the \( \tau_{Graph} \) of three-bit reversible functions.

**Definition 5:** A weighted graph \( G(C, A) \) is defined as an ant system graph (ASGraph) where \( C = \{C_1, C_2, \ldots, C_n\} \) is a finite set of components (reversible function). Set \( C \) contains all the arcs (gates) connecting the components. Therefore, reversible function synthesis can be formulated as a minimization problem on an ASGraph where the weight of each arc \( w_{ij} \) is defined as the minimum cost of gates in \( a_{ij} \).

For example, Fig. 2(b) illustrates the ASGraph for the example reversible function \( f = [2, 6, 0, 5, 7, 3, 4, 1] \) in Fig. 1(a), with the corresponding Toffoli gates labeled on the arcs.

### III. Methodology

In this section, we describe the details of our reversible logic synthesis methodology.

**Algorithm 1 ACO-based reversible logic synthesis**

1. while \( ++k < G_{LOOP} \) do
2.     for all ants \( i = 1 \) to \( ANT\_NUM \) do
3.         initialize \( tour_i \), \( begin \) as the output state of \( f \)
4.     initialize \( dist_i = C(f) \)
5.     while \( dist_i \neq 0 \) do ++local iter < \( L\_LOOP \) do
6.         local_init_func = tour_i, \( end() \)
7.     for all local traces \( j = 1 \) to \( BREADTH \) do
8.         initialize local_trace_j
9.         \( CS_i = get\_next\_func(local\_init\_func) \)
10.        add \( CS_i \) to local_trace_j
11.       while \( + + \) trace_length < \( DEPTH \) do
12.          \( CS_i = get\_next\_func(CS_i) \)
13.          add \( CS_i \) to local_trace_j
14.       if \( C(CS_i) < dist_i \), then
15.           \( dist_i = C(CS_i) \)
16.           best_trace = \( j \)
17.       end if
18.     end while
19.     end for
20.    \( local\_trace_{j, best} = trim(local\_trace_{j, best}) \)
21.    \( local\_trace_{j, best} = del\_loop(local\_trace_{j, best}) \)
22.    append \( local\_trace_{j, best} \) to \( tour_i \)
23.    end while
24.    \( tour_i = del\_loop(tour_i) \)
25. end for
26. set \( min\_cost(tour_i) \)
27. update \( \tau_{Graph} \)
28. end while

The main procedure of our ACO-based reversible logic synthesis is illustrated in Algorithm 1. Parameters \( G_{LOOP} \) and \( L_{LOOP} \) are the upper bounds of iteration number for global and local search, respectively. A total of \( ANT\_NUM \) ants are employed. During a tour of each ant \( i \), the local search terminates when the current best distance \( dist_i = 0 \) which means the \( f_I \) is reached or the iteration index \( local\_iter \) reaches the upper bound \( L_{LOOP} \). The local search is implemented at lines 7–19, which nests depth-first searches in a breadth-first search (controlled by parameters \( BREADTH \) and \( DEPTH \), respectively). At line 9, the current state \( (CS_i) \) is updated after ant \( i \) selects a reversible gate based on our ACO stochastic model described in Section III-A. During the local search in
each iteration, ant $i$ evaluates BREADTH traces each with a length of DEPTH and picks the best trace, local_trace$_{i,best}$. The best trace has an ending state $CS_{best}$, which is closest to $f_t$, evaluated by the complexity function $C(f)$. The ending state $CS_{best}$ may sometimes appear multiple times in the middle of local_trace$_{i,best}$. All the state transitions after the first arrival of $CS_{best}$ in the trace thus become redundant. They are deleted by function trim at line 20. Function del_loop (line 21) removes circle traces which start and end at the same function. Now local_trace$_{i,best}$ is appended to the global tour $tour_i$. Once all the ants have finished their global searches, each global tour will be examined to remove loop tours and the $\tau_{Graph}$ is updated with new pheromone values. Finally, the tour with the minimum synthesis cost gives the output. We next discuss several key steps in the algorithm.

A. Stochastic gate selection model

During the local search, each ant selects a promising gate $g_i(c, t)$ to decrease the function complexity. This process is formulated as a stochastic model composed of two parts. The first is to decide the target bit governed by parameter $\psi$. In particular, at the $k^{th}$ iteration of the algorithm, the probability of selecting the $t^{th}$ bit as the target bit by ant $i$ at current state $CS_i = \{s_0, s_1, \ldots, s_N\}$ is calculated as:

$$p_t(CS_i, k) = \left\{ \begin{array}{ll}
\frac{W_t(CS_i, k)^\alpha \times (\eta_t)^\beta}{\sum_{j=0}^{N} W_t(CS_i, k)^\alpha \times (\eta_j)^\beta}, & \text{if } W_t(CS_i, k) > 0 \\
0, & \text{otherwise}
\end{array} \right. \tag{3}$$

where $W_t(CS_i, k) = \sum_{i=0}^{N} \psi_s, i(t)$ is the sum of pheromone for $t^{th}$ bit at each edge between patterns of $CS_i$ and $f_t$, $\eta_t$ is the best complexity value of next state when choosing bit $t$ as target bit. Parameters $\alpha$ and $\beta$ determine the relative impact of pheromone feedback and estimated evaluation of the transition on selection probability. In our experiment, we fix both $\alpha$ and $\beta$ value at one to simplify the computation.

When the target bit $t$ is determined, we will decide the value for each control bit $m$, governed by parameter $\phi(t, m, c_m)$. In particular, the probability for the control bit $m$ to be $c_m$ is:

$$p_c(CS_i, k) = \left\{ \begin{array}{ll}
\frac{W_c(CS_i, k)^\alpha \times (\eta_c)^\beta}{\sum_{j=0}^{N} W_c(CS_i, k)^\alpha \times (\eta_j)^\beta}, & \text{if } W_c(CS_i, k) > 0 \\
0, & \text{otherwise}
\end{array} \right. \tag{4}$$

where $W_c(CS_i, k) = \sum_{i=0}^{N} \psi_s, i(t), m, c_m$ is the sum of pheromone for the $m^{th}$ bit set as value $c_m$ when the $t^{th}$ bit is the target bit.

B. Initialization of $\tau_{pq}$

In order to optimize the memory usage, a hash table is employed to store the updated pheromone values. Therefore, when a $\tau_{pq}$ is required for decision making, the hash table is checked first. If it fails, we know that the $\tau_{pq}$ value has not been updated yet and a default value will be assigned according to the procedure summarized as follows. If $\vec{p}$ and $\vec{q}$ differ on bit $t$ ($p_t \neq q_t$), we can always find a Toffoli gate with target bit $t$ that outputs $\vec{q}$ from input $\vec{p}$. Therefore, we assign value $\psi_{pq}(t)$ as a positive constant $\tau_{T}$ to favor the selection of $t$ as the target bit; otherwise, it is set to zero. For the same vertex pair $p$ and $q$, and same target bit $t$, $\phi_{pq}(t, i, c_t)$ is assigned to $\tau_{P}$ (a positive constant, $P$ for positive) if $p_t \oplus c_t = 1$, and $\tau_{N}$ (a negative constant, $N$ for negative), if $p_t \oplus c_t = 0$. The reason for this assignment is because according to Equation 1, we can flip bit $t$ if $z_t = p_t \oplus c_t = 1$, which should be favored when determining the control value $c_t$. For $c_t = 2$, a positive constant $\tau_{D}$ (D for don’t care) is used, as a gate can still flip target bit $t$ if bit $i$ is not a control bit. The relationship of the three constant values used is $\tau_{P} > \tau_{D} > |\tau_{N}|$ to reflect the preference during gate selection process to maximize the decrease of function complexity. But for target bit with $p_t = q_t$, we assign $\phi_{pq}(t, i, c_t)$ as $\tau_{N}/n$ if $z_t = p_t \oplus c_t = 1$, $\tau_{P}/n$ if $z_t = p_t \oplus c_t = 0$, and $-\tau_{D}/n$ if $c_t = 2$. The values are assigned in the opposite way as when $p_t \neq q_t$ to eliminate state transition by flipping bit $t$. The constant values are then divided by the number of bit $n$ such that these unnecessary transitions will not overweight the impact of those favored. Finally, when $i$ equals $t$, the $\phi$ values are set to zero.

C. Pheromone update

After all the ants have completed their tours, the pheromone graph is updated as:

$$\tau_{pq}(g, k+1) = (1-\rho) \times \tau_{pq}(g, k) + \sum_{i=1}^{\text{ANT.NUM}} \Delta \tau_{pq}(g, k) \tag{5}$$

where $\rho$ is the pheromone evaporation rate (less than one). The evaporation process can avoid unlimited accumulation of pheromone and alleviate the influence of some bad decisions. $\Delta \tau_{pq}(g, k)$ is the amount of pheromone that ant $i$ deposits at iteration $k$. It is defined as:

$$\Delta \tau_{pq}(g, t) = \left\{ \begin{array}{ll}
\frac{Q_t(tour_i)}{Q} \times \text{pos}(g), & \text{if } g \in tour_i \\
0, & \text{otherwise}
\end{array} \right. \tag{6}$$

where $Q$ is the total amount of pheromone that one ant can release, and $C$ represents the total cost of the tour. The
reciprocal relation ensures that the gates on the tour with less costs will receive more pheromone. More specifically, if $g_q(t, \tilde{t})$ is selected at state $CS = [s_0, s_1, \ldots, s_N]$ along the travel, $\psi_{s_q}(t)$ and $\delta_{s_q}(t, m, c_m)$ will be reinforced. $pos(q)$ tracks the relative position of gate $g$ with respect to $tour_{begin}$. Hence, the gates closer to the start of the tour will receive more pheromone. The reason is because, generally at the beginning of the tour, the complexity of the state is higher, which usually results in an increasing number of candidate gates. Therefore, gate selection at an early stage is more difficult yet more important than a later stage. Early selection should be favored if the intermediate result is good.

### IV. EXPERIMENTAL RESULTS

We evaluated the proposed method in C++ on an Intel Xeon 3GHz workstation with 2GB memory, running Red Hat Linux operation system.

The ACO algorithm parameters are configured as follows. Both $\alpha$ and $\beta$ are fixed at one. We use ten ants ($ANTS = 10$) and the global iteration upper bound is $G_LOOP = 10$. Local search parameters $BREADTH$ and $DEPTH$ are set to three and seven, respectively. All the parameters used here are experimentally determined. And we believe that a proper tuning of these parameters according to the problem size may improve the synthesis result.

#### A. Comparisons with other heuristics

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 0, 1, 2, 3, 4, 5, 6]</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>[0, 1, 2, 3, 4, 5, 6, 7]</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]</td>
<td>7</td>
<td>9</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>[3, 2, 1, 0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>[3, 4, 0, 2, 5, 6, 1]</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>[2, 3, 4, 5, 6, 1, 0]</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>[8, 2, 14, 15, 1, 11, 10, 0, 5, 8, 1, 15, 12, 4, 9]</td>
<td>14</td>
<td>19</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>[2, 9, 7, 13, 10, 4, 14, 3, 0, 12, 6, 8, 15, 11, 15]</td>
<td>14</td>
<td>23</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>[6, 4, 11, 0, 9, 8, 12, 2, 15, 5, 3, 7, 10, 14, 11]</td>
<td>17</td>
<td>21</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>[13, 1, 4, 0, 2, 15, 6, 12, 8, 11, 3, 4, 5, 7, 10]</td>
<td>14</td>
<td>29</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Average gate count</td>
<td>7.44</td>
<td>9.81</td>
<td>6.06</td>
<td></td>
</tr>
<tr>
<td>Total time (seconds)</td>
<td>44.23</td>
<td>-</td>
<td>1.19</td>
<td></td>
</tr>
</tbody>
</table>

Table I reports the synthesis results of the benchmark circuits discussed in [7]. The results are compared with two state-of-the-art heuristics, namely, the PPRM expansion method [6] and non-search method MOSAIC [7]. It can be observed that the proposed ACO method outperforms both of them in terms of gate count. Compared with MOSAIC, on average, ACO can achieve a 38% reduction on gate count. Note that no execution time is reported in [7]. Although our synthesis results are comparable to PPRM, our method only takes a fraction of computation time (totaling only 1.19s) to synthesize all the 16 reversible functions, while PPRM takes 44.23s. This is more than $37 \times$ speedup.

#### B. Scalability analysis

To analyze the scalability of the proposed algorithm, we evaluate with random reversible functions of input size from 9 to 16 bits. The resultant gate count, computation time, and memory usage are listed in Table II.

**TABLE II: SCALABILITY ANALYSIS OF LARGE INPUT SIZES**

<table>
<thead>
<tr>
<th>Input number</th>
<th>Gate count</th>
<th>Time (seconds)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>159</td>
<td>1.8</td>
<td>66</td>
</tr>
<tr>
<td>10</td>
<td>491</td>
<td>38.6</td>
<td>132</td>
</tr>
<tr>
<td>11</td>
<td>277</td>
<td>16.6</td>
<td>98</td>
</tr>
<tr>
<td>12</td>
<td>346</td>
<td>64.4</td>
<td>134</td>
</tr>
<tr>
<td>13</td>
<td>428</td>
<td>80.6</td>
<td>120</td>
</tr>
<tr>
<td>14</td>
<td>922</td>
<td>149.2</td>
<td>128</td>
</tr>
<tr>
<td>15</td>
<td>782</td>
<td>379.0</td>
<td>168</td>
</tr>
<tr>
<td>16</td>
<td>1153</td>
<td>595.1</td>
<td>204</td>
</tr>
</tbody>
</table>

Due to the nature of swarm intelligence, the ACO algorithm explores only a small subset of the entire search space, and hence can significantly reduce the solving efforts to achieve optimal or near-optimal solutions. It also has the flexibility to trade off optimization degree with synthesis time through tuning the ACO parameters. Therefore, the proposed ACO-based method points at a promising direction of time- and memory-efficient synthesis especially for large reversible functions.

#### V. CONCLUSIONS

We have proposed a novel ACO-based reversible logic synthesis approach. A new probabilistic transition model is formulated, and the corresponding graph based on the characteristics of reversible gates eases the ACO-based search. Compared with existing techniques, our method has superior performance in both gate count and computational cost.

**REFERENCES**


