A Variable-Length Chromosome Evolutionary Algorithm for Reversible Circuit Synthesis

XIAOXIAO WANG\textsuperscript{1,2,*}, LICHENG JIAO\textsuperscript{1}, YANGYANG LI\textsuperscript{1}, YUTAO QI\textsuperscript{1} and JIANSHE WU\textsuperscript{1}

\textsuperscript{1}Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education, School of Electronic Engineering, Xidian University, China
E-mail: lchjiao@xidian.edu.cn; yyli@xidian.edu.cn; qi_yutao@163.com; jshwu@xidian.edu.cn
\textsuperscript{2}School of Computer Science, Xi’an Shiyou University, China

Accepted: August 30, 2015.

A variable-length chromosome evolutionary algorithm for reversible circuit synthesis (VLEA\textsubscript{RC}) is presented to improve the quality of solutions in terms of quantum cost. The synthesis problem is formulated as a minimization problem with an equality constraint. To begin with, a modified stochastic ranking method for constraint handling is devised. This gives a better balance between decreasing the constraint violation and increasing the objective value through the use of parsimony pressure. Then, a periodic population update mechanism is applied when the evolution process stagnates. This mechanism employs heuristic information extracted from the positive polarity Reed-Muller expansion of the reversible specification. This can improve the feasible ratio and reduce the search space effectively. Our design is tested on several widely used benchmarks with circuit size varying from 4 to 30 inputs. The results show that the proposed method can find high quality solutions for the tested benchmarks as well as improve the circuit size that can be handled compared to previous evolutionary methods.

Keywords: Reversible circuit synthesis; variable-length chromosome; evolutionary algorithm; chromosome bloat; equality constraint;

1 INTRODUCTION

Nowadays, the field of reversible computing has received considerable attention in various research areas including low-power CMOS design, optical computing, and quantum computing [1].

\textsuperscript{*} Corresponding author: E-mail: xxwang@xsyu.edu.cn
Given a reversible specification, i.e. a permutation, reversible circuit (RC) synthesis is a process of finding a composition of reversible gates from a universal reversible gate set which minimizes cost while satisfies the reversible specification. Gate count (GC) or quantum cost (QC) is usually used as the cost metric to estimate the quality of a reversible circuit.

Existing RC synthesis algorithms are categorized into three classes: deterministic algorithms, heuristic algorithms and evolutionary algorithms.

Most deterministic algorithms can obtain feasible circuits efficiently, but these circuits often need to be optimized further. Transformation methods change the truth table [2] or Reed-Muller spectra [3] of a reversible function into that of identity function, and then optimize the circuit using template matching [4]. Binary Decision Diagram (BDD) based methods [5, 6] represent Boolean functions by BDDs, and then substitute each node with a cascade of reversible gates. Cycle-based algorithms [7, 8] decompose the permutation of a reversible function into a product of disjoint cycles, and then realize each cycle with a sequence of reversible gates. These algorithms can solve large scale problems rapidly, with template matching being used to further optimize the synthesis results. Other deterministic algorithms conduct exhaustive search to find the optimal solution. Some researches focus on minimizing gate count. Shende et al. [9] found optimal circuits for all 3-bit functions. Golubitsky et al. [10] have developed a tool capable of finding a CNP (CNOT, NOT, Peres) circuit with the minimal GC for any 4-bit permutation. D. Große et al. [11, 12] formulated the synthesis problem as a sequence of SAT problems and found optimal gate count reversible circuits for functions less than 5 bits. Other efforts have been made recently to reduce the QC. Szyprwowski et al. [13] employed the exhaustive method from [10] to find a local minimal-QC circuit for a range of GCs. However this method does not guarantee the exact minimum as it is time consuming to consider all different gate counts. Due to the super-exponential increase in time and memory requirements, exhaustive searches are impeded by hardware limitations when the problem size is more than 5 bits.

The classical heuristic algorithm, Reed-Muller Reversible Logic Synthesizer (RMRLS) [14], uses Reed-Muller expansions of reversible functions to construct priority-based search trees, and rapidly prunes the search space by using heuristic information. The greedy nature of the RMRLS heuristic affects the quality of its solutions.

Various evolutionary algorithms are already used in quantum circuit synthesis, such as GP [15, 16], GA [17–20], EP [21], HQEA [22], and in reversible circuit synthesis [23–25] due to their capability for global search. However, only small scale problems with less than 7 inputs have been tested in these algorithms, synthesizing circuits of at most 25 gates.
The quality of the results from previous applications of EAs to the RC synthesis has been limited, possibly due to the following difficulties. First and foremost, chromosome bloat, the uncontrolled growth of the average size of individuals, must be avoided in variable-length chromosome evolutionary algorithm (VLEA). Moreover, the population is full of infeasible solutions because the construction of feasible solutions is difficult and no effective mechanism to directly repair infeasible individuals has been proposed. Finally, the search is prone to falling into stagnation or premature convergence due to the highly epistatic nature of RC synthesis. That is, the contribution of a gene to the fitness of an individual is highly depending on the genes in the front always makes evolution converge to a partial solution. In order to advance the development of EAs in the field of reversible synthesis, we must address the above problems.

This paper proposes a novel variable-length chromosome evolutionary algorithm for RC synthesis (VLEA_RC). It is restricted to QC-minimization using Generalized Toffoli gates (GT) without adding an ancilla line. The algorithm starts with short chromosomes, and then tailors an existing variable length crossover operator [26] to automatically grow the chromosomes in a slow and controlled rate. As the algorithm sets a size limit for chromosomes to avoid bloat, this controlled rate of growth allows more generations with which to explore the global search space. Besides these techniques usually adopted by general VLEAs, our algorithm mainly employs two mechanisms to address the aforementioned difficulties. Firstly, a new constraint handling method, modified stochastic ranking (MSR), is proposed, which is based on stochastic ranking (SR) [27]. Unlike SR which randomly ranks two infeasible individuals according to objective value or constraint violation through a predefined probability, MSR ranks infeasible individuals according to their domination relationship in terms of constraint violation (CV) and objective value. CV herein is the error of a circuit, and objective value means the quantum cost of a circuit. When two infeasible ones do not dominate each other, MSR balances between the two contradictory aspects: decreasing CV versus minimizing QC. It computes a ratio of the difference between objective values to the difference between CVs. MSR makes rank according to the value of the ratio and a predefined probability. In addition, a new population update mechanism similar to a hyper mutation is applied when evolution stagnates. The selected individuals are updated by appending several gates extracted from the positive polarity Reed-Muller (PPRM) expansion. The extraction method is same as that of RMRLS [14], but the appending and optimization mechanism is different. RMRLS uses a greedy method to score each extracted gates and append the best one, while our method appends randomly selected gates from the extracted ones and optimize through evolution. The mechanism can
extract the evolution from stagnation, increase the feasible ratio and improve the convergence speed.

VLEA_RC improves the problem scale that EAs could possibly handle. The quality of its solutions are comparable to the best known ones obtained from EAs and other methods as of 2013.

The rest of this paper is organized as follows: Section II introduces some basic concepts. Section III gives related works and motivation. Section IV describes VLEA_RC and its mechanics. Section V shows the experimental results. Finally, Section VI concludes this paper and outlines the possible directions for future research.

2 FUNDAMENTALS OF REVERSIBLE LOGIC CIRCUITS

Here we introduce the fundamentals of reversible circuit synthesis.

2.1 Reversible Functions and Reversible Gates

Definition 2.1. A function $f : B^n \rightarrow B^n$ with $n$ inputs and $n$ outputs is reversible, if and only if it is a one-to-one mapping between a set of $B^n$ input vectors and a set of $B^n$ output vectors.

Definition 2.2. Let $X = \{x_1, x_2, .., x_n\}$ be the set of domain variables. Generalized Toffoli gate (GT), also called multiple-control Toffoli, has the form $T(C, t)$, where $C$ is the set of control lines with $C \subseteq X$ and $t$ is the target line with $t \in X$ but $t \notin C$. The value of the target line is inverted if all control lines are assigned to 1. For $|C| = 0$ and $|C| = 1$, the gates are NOT and CNOT respectively. For $|C| = 2$, the gates are called Toffoli. See Figure 1.

2.2 Positive Polarity Reed-Muller Expansion of Reversible Functions

Any Boolean function can be described using an EXOR-sum of products (ESOP) expansion. The PPRM expansion is an ESOP expression which uses only uncomplemented variables and can be generated from a truth table.
Every output of a truth table of a reversible function can be converted into a PPRM expression.

For example, the permutation of the 3-bit reversible function 3, 17 is [7, 1, 4, 3, 0, 2, 6, 5]. Its PPRM expansion for each output are given in (1).

\[
\begin{align*}
  a_o &= 1 \oplus b \oplus ab \oplus c \oplus bc \\
  b_o &= 1 \oplus a \oplus b \oplus c \\
  c_o &= 1 \oplus a \oplus c \oplus ac \oplus bc
\end{align*}
\]  

(1)

Maslov et al. [3] give an efficient technique for transforming the truth table into its PPRM expansion.

### 2.3 Cost of Reversible Toffoli Networks

There are many cost metrics to measure a reversible circuit, such as QC, GC, and Linear Nearest Neighbor Cost (LNN). In this paper, we use QC as the measurement of a reversible circuit.

The QC of a reversible circuit \(rc\) is the sum of the QC of each reversible gate \(g_i\) constituting the circuit.

\[
qc (rc) = \sum_{i=1}^{\text{len}(rc)} gqc (g_i)
\]  

(2)

Where \(\text{len}(rc)\) is the number of gates constituting \(rc\) and \(gqc(g_i)\) represents the quantum cost of a GT gate \(g_i\). It is calculated according to the mechanics described at http://webhome.cs.uvic.ca/dmaslov/definitions.html; see (3).

\[
gqc (T (C, t)) = \begin{cases} 
  1 & |C| = 0 \\
  2^{|C|+1} - 3 & |C| \geq 1
\end{cases}
\]  

(3)

### 2.4 Constraint Violation

In RC synthesis, CV is often measured by the Hamming distance [15, 19] or the matrix trace distance [21] between the target matrix \(O\) corresponding to the reversible specification and the matrix \(S\) representing the synthesized circuit. The computation of matrix \(S\) requires time consuming Kronecker and standard matrix multiplication. Here we propose a new method, drawing inspiration from RMRLS [14], to calculate the CV of the circuit \(rc\), which is defined in (4).

\[
\text{cv}(rc) = \text{diffTerm} (\text{reduced PPRM})
\]  

(4)

The function \text{diffTerm}() returns the number of different terms between the \text{reduced PPRM} and the PPRM of the identity function with the same
size. The reduced \textit{PPRM} is obtained through substituting the sequence of reversible gates of \textit{rc} successively into the PPRM of a reversible function. For example, for function 3_17, its PPRM is shown in (1). Substituting \( a \oplus 1 \) into each instance of \( a \) in (1), we can obtain \textit{reduced PPRM} in (5).

\begin{align*}
  a_o &= 1 \oplus ab \oplus c \oplus bc \\
  b_o &= a \oplus b \oplus c \\
  c_o &= a \oplus ac \oplus bc.
\end{align*}

The number of different terms between (5) and the PPRM of the identity function is 11. Therefore the CV of the circuit with only one gate NOT(\( a \)) is 11.

\section{3 MOTIVATION AND RELATED WORK}

In this section, we introduce the involved techniques involved in and the motivation for VLEA RC.

\subsection{3.1 Equality Constraint Handling}

Early evolutionary algorithms for quantum and reversible logic synthesis often model the problem as a correctness maximization problem without constraints [15, 18]. Then, some researchers focus on minimizing a weighted sum of the circuit cost and the error (reciprocal relation to correctness) [17, 19, 22, 23]. In order to find correct circuits, we formulate the problem as a minimization problem with equality constraint, that is, to ensure correctness whilst minimize QC.

Existing constraint handling techniques can be grouped as: penalty functions, repair algorithms, separation of objectives and constraints [27], multi-objective based techniques [28] and hybrid methods [29]. The handling of equality constraints has long been a difficult issue for evolutionary optimization methods, on account of feasible space being very small compared to the entire search space. Recently appeared algorithms for equality constraint handling are specialized for continuous functions, such as the local search methods for feasibility reparation [30–32] and the hybrid algorithm [33].

For RC synthesis, feasible solutions are difficult to build and no effective mechanism to directly repair infeasible solutions has been proposed. Consequently, the evolving population is full of infeasible ones and the ranking of infeasible solutions needs to be paid more attention to. This paper employs the separation of objectives and constraints mechanism to solve equality constraints, and emphasizes the comparison of undominated infeasible individuals.
The methods belonging to this category include: Superiority of Feasible Solutions (SF) [34], $\varepsilon$-constraint [35], Stochastic Ranking (SR) [27], and etc. Giving absolute priority to CV decreasing, just as SF and $\varepsilon$-constraint do, may lead solutions with large QC. We need an algorithm sufficiently considering both the CV and objective value sufficiently in the whole evolution process. SR seems to fit that bill, but the stochastic balance between CV and objective value may sometimes cause the degradation of evolution. Modified SR (MSR) is based on SR and introduces multi-objective idea and parsimony pressure into SR. It reduces the randomness by evaluating the ratio of the QC difference to the CV difference between two nondominated infeasibles. If the ratio is less than a predefined value $\rho$, that is to say, the decrease in CV only leads to a small increase in QC, the infeasible individual with smaller CV but larger QC ranks first. Otherwise, the ranking is based on the probability $p_f$ defined in [27]. As described in the next section, the MSR mechanism can also help avoid chromosome bloat.

3.2 Variable Length Evolutionary Algorithm

In RC synthesis, the length of the optimum solution is generally unknown beforehand, so we adopt a variable-length representation. For existing VLEAs, the primary issue to be addressed is how to avoid chromosome bloat, i.e. allele redundancy and undue growth.

Numerous methods for controlling bloat can be found in the GP literature [36]. We can classify the methods into two categories: explicit and implicit methods. Explicit methods often use individual size as an explicit index to control chromosome growth, such as depth limiting, Tarpeian, parsimony pressure, biased multi-objective, waiting room, double tournament, proportional tournament, and death by size. Implicit controls include nondestructive crossover, and recently, space structure with elitism [26, 33]. There are also several successful variable-length chromosome GAs, such as the messy GA, Chunking GA [37] and a progressive refinement VCLGA [38]. However, the existing VLEAs are not appropriate for the problem with unbounded size. In addition to the above, nondestructive crossover mechanisms are also used in VLEAs, such as SAGA [39], Virtual Virus (VIV) and SVLC [26]. In our algorithm, we use three mechanisms to control bloat: size limit, nondestructive crossover and MSR.

**Size Limit**

We set the maximum length of chromosomes according to the complexity of the RLC problem; see 4.2 and Table 1.

**SVLC**

SVLC is a nondestructive crossover operator which ensures that the average size of chromosomes will increase gradually. This affords the EA the
opportunity to search for solutions in a more correlated landscape, i.e. all chromosomes in nearby generations will be of similar lengths [40].

**MSR**

In our algorithm, tournament selection is based on the sorting conducted by MSR. MSR gives priority to individuals which decrease CV with only small increase in QC. Having a large QC often means that the individual has a large chromosome size, or that the individual consists of gates with large cost, so MSR can also play a role in controlling bloat.

### 3.3 Evolution Stagnation and population update

We perform an evolutionary experiment on a randomly selected reversible function, *ham*7. Figure 2 shows the variation of the average CV and average chromosome length with time under the effect of SVLC and MSR. In this initial phase, the average CV and the average length of chromosomes will both decrease rapidly and then level off with generations due to the high-epistasis characteristic of the problem. After this initial phase, the average length will grow in gradual incremental steps, with the CV decreasing slowly under the effect of SVLC and selection bias. Figure 3 shows the increasing trend of the average length will hit the wall at last. The inherent difficulty of equality constraint and the population convergence may sometimes impede the necessary growth of the chromosomes to find the optimal solution. The slow growth in size does not allow the optimal solution within the simulation time. We can see in Figure 4 that feasible solutions can not be obtained through one hundred thousand generations.
To detect the changed search space, extract evolution from stagnation and increase the diversity of population, a special operation called population update is executed iteratively every few generations. It appends a specific number of reversible gates to a selected chromosome. The appended gates are generated randomly or chosen from the preferential gate library of corresponding chromosome. The preferential gate library consists of gates extracted from the PPRM expansion of the current chromosome. It is inspired
by Gupta’s RMRLS algorithm [14], but different from it. RMRLS extracts factors from PPRM, calculates their priority and then decides next gate with greedy choice. The population update can conquer the greediness of heuristic rules and avoid mistaken pruning by evolution process.

4 VLEA_RC ON REVERSIBLE CIRCUIT SYNTHESIS

This section describes the proposed synthesis algorithm and gives the details of MSR and population update. The basics evolved in VLEA_RC are also involved.

4.1 Overview of VLEA_RC

The framework of VLEA_RC is similar to that of GA except the execution of the population update to the selected individuals whenever evolution falls into stagnation. See Algorithm 1.

The initialization phase is described in lines 1 to 3. See Section 4.2 for more details. The evaluation phase involves compaction, evaluation and sorting, referring to lines 4-5. In line 4, each individual is compacted to guarantee no occurrence of same adjacent gates, and to obtain correct QC. In line 5, MSR sorts the individuals. The tournament selection in line 15 is conducted based on the results. min_cv and min_cost are used to record the CV and the QC of the best solution from the current generation respectively (lines 6,7 and 21). last_min_cv and last_min_cost are the previous best min_cv and min_cost (lines 8-9 and lines 22-24). If they have not been improved for step generations, we conduct a population update operation (line 13). Otherwise, the normal evolutionary operation is conducted (lines 15-16).

4.2 Initialization Phase

Preferential Gate Library

Preferential gate library pref Lib is constructed by extracting eligible factors from the PPRM of a reversible specification. Eligible factors are those factors in the PPRM expansion of each variable \( v_i \) that do not contain the variable \( v_i \). For example, we can extract four factors for variable \( a \) from (1) which are \( a = a \oplus 1 \), \( a = a \oplus b \), \( a = a \oplus c \) and \( a = a \oplus bc \), namely, Not(\( a \)), CNOT(\( b, a \)), CNOT(\( c, a \)) and T(\( b, c, a \)). The pref Lib for reversible function 3_17 contains a total of nine gates.

Attribute Information

The following three values can reflect the complexity of a reversible function to some extent and are used during the population initialization.
Algorithm 1 VLEA_RC for reversible circuit synthesis

Require:
- PPRM expansion of function \( f(x_1, x_2, x_n) \);
- Population size \( p_n \), crossover probability \( p_c \), mutation probability \( p_m \);
- Population update interval \( step \);

Ensure:
- The best individual (synthesized reversible circuit) \( I_0 \);

1: construct initial preferLib
   detect maxConNum, factorNum and diffTerm;
2: estimate the maximum length \( maxLen \) and the initial length \( initialLen \)
   of individuals;
3: initialize population \( P(t) \);
4: compact and evaluate individuals of \( P(t) \);
5: sort individuals using MSR;
6: \( min_cv \) equals to the CV of the best individual;
7: \( min_cost \) equals to the cost of the best individual;
8: \( last_min cv = min_cv \);
9: \( last_min cost = min_cost \);
10: \( u = 0 \);
11: while stop criterion is not met do
12:    if \( u == step \) then
13:       update \( P(t) \) to \( P(t+1) \) using jumping growth;
14:    else
15:       selection, crossover and mutation according to \( p_c \) and \( p_m \);
16:       generate \( P(t + 1) \);
17:    end if
18:    compact and evaluate the individuals in \( P(t + 1) \);
19:    sort individuals in \( P(t + 1) \) using MSR;
20:    elitism reservation;
21:    update \( min_cv \) and \( min_cost \);
22:    if \( (last_min cv > min_cv) \) or \( (last_min cv == min_cv \) and \( last_min cost > min_cost) \) then
23:       \( last_min cv = min_cv \);
24:       \( last_min cost = min_cost \);
25:    else
26:       \( u = u + 1 \);
27:    end if
28:    \( t = t + 1 \);
29: end while
factorNum represents the number of eligible factors subtracted from the PPRM of a given function, namely, the size of prefLib. We set maxLen, the maximum length of chromosomes, according to factorNum. Then, the initial length of a chromosome, initialLen, can be calculated according to (6).

\[
\text{initialLen} = \begin{cases} 
\frac{\text{maxLen}}{2} + \text{rand}(0, 5) & \text{maxLen} \leq 30 \\
\frac{\text{maxLen}}{3} + \text{rand}(0, 5) & 30 < \text{maxLen} \leq 50 \\
15 + \text{rand}(0, 5) & \text{maxLen} > 50 
\end{cases}
\]  

(6)

diffTerm is the number of different terms between the PPRM of a reversible function and the PPRM of the identity function of the same size.

maxConNum is the maximum number of control bits of the factors in prefLib and can be used as the upper limit of the number of control bits of a randomly generated gates constituting an individual.

**Encoding of Reversible Circuits**

An n-bit circuit consists of the GT gates from prefLib or randomly generated. Any GT gate $T(C, t)$ can be encoded as a two-tuple $(C_{n-1}C_{n-2}...C_0, t)$. $C_{n-1}C_{n-2}...C_0$ is a bit string of length n. If $C_i$ is set to 1, $i$ will be a control bit of $T(C, t)$; $t$, an integer between 0 and $n-1$, denotes the position of the target bit. A 3-bit circuit including NOT(0), CNOT(2,1) and $T(1,2,0)$ can be encoded as \{((000,0),(100,1),(110,0))\}.

The more the number of control bits of a gate, the lower the generating probability of the gate.

### 4.3 Modified Stochastic Ranking

See Algorithm 2. If two adjacent individuals $I_j$ and $I_{j+1}$ have a dominate relationship in light of their objective values and CVs, the dominating one will rank first (lines 4 and 5). If two individuals are non-dominated each other (lines 6 and 12), we compute a ratio of the difference between the two CVs to the difference between the QCs. If the result is less than a predefined value $\rho$ (line 7), it means that an improvement by one unit in CV is worth an increase of QC by at most $\rho$ unit and the individual with lower CV should rank first (line 8); otherwise, the individual with lower CV and overlarge cost will rank first with small probability (lines 9-10 and lines 14-15). The parsimony pressure can be controlled by $\rho$. The larger the $\rho$ is, the lower the parsimony pressure becomes.

### 4.4 Population Update using Jump Growth

Whenever the best solution has not been improved for step generations, we conduct a population update. If the individual $I_j$ selected by a tournament is
Algorithm 2 Modified Stochastic Ranking

Require:

\( P(t), p_f, \rho; \)

Ensure:

Ranked population \( P(t)'; \)

\( I_j \) is the \( j \)th individual;

1: for \( i := 1 \) to \( n \) do
2:    for \( j := 1 \) to \( n \) do
3:        sample \( u \in U(0, 1) \)
4:        if \( cv(I_j) \geq cv(I_{j+1}) \) and \( qc(I_j) \geq qc(I_{j+1}) \) then
5:            swap\( (I_j, I_{j+1}); \)
6:        else if \( cv(I_j) > cv(I_{j+1}) \) and \( qc(I_j) < qc(I_{j+1}) \) then
7:            if \( (qc(I_{j+1}) - qc(I_j))/(cv(I_j) - cv(I_{j+1})) < \rho \) then
8:                swap\( (I_j, I_{j+1}); \)
9:            else if \( u < p_f \) then
10:               swap\( (I_j, I_{j+1}); \)
11:           end if
12:        else if \( cv(I_j) < cv(I_{j+1}) \) and \( qc(I_j) > qc(I_{j+1}) \) then
13:            if \( (qc(I_{j+1}) - qc(I_j))/(cv(I_j) - cv(I_{j+1})) > \rho \) then
14:                if \( u > p_f \) then
15:                    swap\( (I_j, I_{j+1}); \)
16:                end if
17:            end if
18:        end if
19:    end for
20:    if no swap done then
21:        break;
22:    end if
23: end for

Infeasible and its real length, \( real\_Len \), is less than \( max\_Len \), we update \( I_j \) by a jump growth (JG) operation; otherwise \( I_j \) goes into the next generation directly. This is repeated until the updated population is full. See Algorithm 3 for the details of jump growth.

The extending length of an individual, \( add\_Len \), is determined by \( unit \) and \( diff\_Len \) (lines 4-8). \( unit \) is the default extending length. \( diff\_Len \) is the difference between \( max\_Len \) and \( real\_Len \) of an individual. The gates appended to the tail of \( I_j \) can be selected from \( pref\_Lib^{I_j} \) (lines 11-13) or generated randomly (lines 15-17). \( pref\_Lib^{I_j} \) is generated by subtracting the preferential gates from the \( reduced\_PPRM \) of \( I_j \).
Algorithm 3 chromosome jump growth

Require:
A selected individual $I_j$ from $P(t)$;
The default extending length $unit$;

Ensure:
An extended individual $I'_j$ based on $I_j$;

1: update the preferential gate library of $I_j$, $pref\ Lib^I_j$;
2: compute the real length of $I_j$, $real\ Len$;
3: $diff\ Len = max\ Len - real\ Len$;
4: if $diff\ Len > unit$ then
5: $add\ Len = unit$;
6: else
7: $add\ Len = diff\ Len$;
8: end if
9: generate $I'_j$ by duplicating $I_j$
10: if $rand() \% 5 == 1$ then
11: for $i := real\ Len$ to $real\ Len + add\ Len$ do
12: append a random gate from $pref\ Lib^I_j$ to $I'_j$;
13: end for
14: else
15: for $i := real\ Len$ to $real\ Len + add\ Len$ do
16: append a randomly generated gate to $I'_j$;
17: end for
18: end if
19: compact and evaluate $I'_j$;

4.5 Recombination
If two parents have no common substrings, the standard one-point crossover or two-point crossover with truncation is applied otherwise SVLC [26] is applied. The children are subjected to one of several mutations. The first type of mutation is called gate mutation inside of a gene, such as tuning control bits or target bit; the second type of mutation is applied at the chromosome level, including adding a gate, deleting a gate, exchanging two gates, and replacing a gate with a random gate. One can reference [21] for more details.

5 EXPERIMENTAL STUDIES
In order to verify the effectiveness of VLEA_RC, to explore the contributions of MSR and JG, and to analyze the parameter settings, benchmarks
with 3 to 30 variables are tested. The details and the original sources of these benchmarks are published at http://webhome.cs.uvic.ca/dmaslov/ or http://www.revlib.org, or reported at the specified references.

All experiments are carried out on a PC with Intel Core 2 Quad CPU and 2 GB memory. Table 1 lists the parameters of VLEA_RC. popSize, the size of a population, is determined by the scale and the complexity of a benchmark. The maximum evolutionary generation maxGen and the value of $\rho$ are given in different experiments.

### 5.1 Comparison of VLEA_RC with a local optimization algorithm

#### Task and Pre-Experimental Planning

In [41], two local optimization methods are applied in sequence to reduce the costs of the circuits obtained from a deterministic algorithm. We test the randomly generated 4-bit functions from [41] in order to verify the effectiveness of VLEA_RC on small scale problems, which carries out synthesis and optimization simultaneously.

#### Experiments setup

We set $\rho$ to 9, maxGen to 1500 and step to 100. Thirty runs are conducted.

#### Results and visualization

Table 2 lists the best results in terms of QC from both the algorithms, the average convergence time over 30 runs and an example of the best circuits from VLEA_RC. $g$ in $(g, c)$ means the GC of a best solution and $c$ the QC. VLEA_RC obtains feasible solutions in all runs, smaller QCs on 12 functions and smaller GCs on 9 functions. APP2.2, APP2.10 and APP2.12 have higher complexity in terms of large diffTerm but low factorNum, so their average convergence time is relatively high. App2.6 has the lowest complexity in terms of diffTerm and maxConNum, and we can obtain the solution in about 0.36 seconds.

The preliminary results show that VLEA_RC can obtain superior solutions to the method [41] which solves the problem by two steps: synthesis and optimization.

<table>
<thead>
<tr>
<th>popSize</th>
<th>TCH$p_c$</th>
<th>TCH$p_m$</th>
<th>TCH$p_f$</th>
<th>maxLen</th>
<th>TCH$\rho$</th>
<th>step</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>100, 300, 500</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
<td>$factor, Num+$ related to 20, 50</td>
<td>maxConNum</td>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>

TABLE 1
Considered parameter values for VLEA_RC.
TABLE 2
Comparison between the best results from VLEA\_RC and those from [41].

<table>
<thead>
<tr>
<th>Func.</th>
<th>[41] ((g, c))</th>
<th>Our ((g, c))</th>
<th>Time [s]</th>
<th>Example Circuit for QCmin</th>
</tr>
</thead>
<tbody>
<tr>
<td>App2.1</td>
<td>(10, 30)</td>
<td>(11, 31)</td>
<td>26.85</td>
<td>C(1,3)C(0,1)T(3,0,2)C(2,3)C(1,2)N(2)</td>
</tr>
<tr>
<td>App2.2</td>
<td>(18, 102)</td>
<td>(12, 40)</td>
<td>49.07</td>
<td>T(3,1,0,2)T(3,2,1)C(0,3)C(0,2)N(3)</td>
</tr>
<tr>
<td>App2.3</td>
<td>(13, 43)</td>
<td>(12, 32)</td>
<td>21.82</td>
<td>C(0,2)N(1)T(1,0,3)C(3,1)T(2,1,0)C(3,0)</td>
</tr>
<tr>
<td>App2.4</td>
<td>(9, 36)</td>
<td>(10, 34)</td>
<td>27.17</td>
<td>N(3)N(1)T(2,1,3)T(3,0,2)C(3,0)</td>
</tr>
<tr>
<td>App2.5</td>
<td>(10, 50)</td>
<td>(9, 29)</td>
<td>32.56</td>
<td>C(1,3)C(1,0)C(3,1)T(2,0,3)T(3,1,0)T(2,0,3)</td>
</tr>
<tr>
<td>App2.6</td>
<td>(6, 14)</td>
<td>(4, 12)</td>
<td>0.36</td>
<td>T(2,0,1)T(1,0,2)C(2,0)C(0,1)</td>
</tr>
<tr>
<td>App2.7</td>
<td>(15, 59)</td>
<td>(9, 29)</td>
<td>7.29</td>
<td>T(2,0,1)C(1,0)T(1,0,2)T(3,2,0)T(3,1,2)C(0,2)</td>
</tr>
<tr>
<td>App2.8</td>
<td>(15, 53)</td>
<td>(11, 43)</td>
<td>28.82</td>
<td>N(2)T(3,0,2)T(1,0,3)C(2,1)T(2,1,3)</td>
</tr>
<tr>
<td>App2.9</td>
<td>(11, 47)</td>
<td>(13, 33)</td>
<td>30.78</td>
<td>T(3,2,1,0)T(1,0,2)C(3,1)T(1,0,2)N(1)N(0)</td>
</tr>
<tr>
<td>App2.10</td>
<td>(13, 57)</td>
<td>(10, 38)</td>
<td>50.62</td>
<td>C(2,0)N(1)C(1,3)T(3,1,2)C(0,1)C(1,2)T(2,1,3)</td>
</tr>
<tr>
<td>App2.11</td>
<td>(12, 80)</td>
<td>(11, 31)</td>
<td>22.44</td>
<td>T(1,0,3)C(0,3)C(2,1)T(3,1,2)T(2,1,3)C(0,2)</td>
</tr>
<tr>
<td>App2.12</td>
<td>(17, 53)</td>
<td>(8, 32)</td>
<td>15.98</td>
<td>N(2)T(2,0,1)C(0,2)T(3,1,0)C(0,1)C(0,2)</td>
</tr>
<tr>
<td>App2.13</td>
<td>(12, 52)</td>
<td>(13, 45)</td>
<td>89.86</td>
<td>C(1,3)T(3,1,2)N(3)C(0,3)C(1,0)T(2,0,3)C(3,0)</td>
</tr>
</tbody>
</table>

5.2 Comparison of VLEA\_RC with a QC-minimization algorithm

**Pre-Experimental Planning and Task**

Golubitsky et al. [10] found the GC-minimization circuits for any 4-bit reversible functions. For a given function \(f\), the algorithm conducts a breadth-first-search to find if there exist reversible circuits \(h\) and \(g\) of length \(k\) and at most \(k\), respectively, such that \(f = h \circ g\). \(k\) starts at 1 and is increasing until the condition is satisfied.

Based on [10], Szyprowski et al. [13] found the QC-minimum circuits under several different GCs. The above two algorithms are space-consuming because they must maintain a circuit library which contains all of the circuits with length from 1 to \(k\). By far, they can not be generalized to functions with more than 4 variables.

In order to test the solution quality of VLEA\_RC on small scale problems, we compare the best results from VLEA\_RC through 30 runs with that from [13]. The parameter settings are same as the previous experiment.
TABLE 3
Comparison between the best results from VLEA_RC and those from [13].

Results and visualization

Our algorithm is based on GT library, while the approach from [13] is based on CNTP. It treats every “T(a, b, c)T(a, b)” pair (or its inverse) or “T(a, b, c)T(b, a)” pair (or its inverse) in a circuit as a Peres gate and assigns the cost of 4 to it as opposed to 6, and thus obtains a smaller QC of a circuit. In Table 3, \(g\) in triple \((g, c, c')\) means GC, \(c\) means QC without considering Peres gate, \(c'\) means QC considering Peres gate. That is to say, our algorithm is aimed at minimizing \(c\), while [13] is aimed at minimizing \(c'\). Although [13] gives the minimum \(c'\) under a few different lengths, Table 3 only lists the best two solutions in terms of \(c\) and \(c'\) respectively due to space limitation and for comparison.

Among the twelve functions, VLEA_RC finds the circuits with smaller \(c\) on functions \(GT5\), \(GT10\) and \(GT12\) and the circuits with the same \(c\) compared with those found in [13] on 9 functions. The feasible ratio over thirty runs is 1 for each function. It shows that VLEA_RC is competitive in QC minimization on 4-bit reversible functions.
Considered benchmark functions and the attribute values.

### Pre-Experimental Planning

The names and the key attribute values of the benchmarks are given in Table 4.\(\text{size}\) is the number of variables in a reversible benchmark and represents the scale of the problem. \(\text{factorNum}\), \(\text{diffTerm}\) and \(\text{maxConNum}\) are introduced in section 4.

### Task

Before we go into analysis of VLEA\textsubscript{RC}, we want to verify the effectiveness of VLEA\textsubscript{RC} on benchmarks more than 4 bits.

### Results and Visualization

We can see the results in Table 5. VLEA\textsubscript{RC} reaches the best known solution on 3\_17, improves QC on the eight benchmarks with only
### TABLE 5
Comparison between the best reported results as of late 2013 and the best results from VLEA_RC.

<table>
<thead>
<tr>
<th>Func.</th>
<th>Best (G_{min}^{c})</th>
<th>Best (Q_{min}^{c})</th>
<th>VLEA_RC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(g)</td>
<td>(c)</td>
<td>(c')</td>
</tr>
<tr>
<td>3.17</td>
<td>6</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>hwb4</td>
<td>11</td>
<td>23</td>
<td>21</td>
</tr>
<tr>
<td>hwb5</td>
<td>24</td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>hwb6</td>
<td>42</td>
<td>140</td>
<td>140</td>
</tr>
<tr>
<td>S0ne013</td>
<td>19</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>S0ne245</td>
<td>20</td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>majority5</td>
<td>16</td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>mod5adder</td>
<td>15</td>
<td>83</td>
<td>83</td>
</tr>
<tr>
<td>mod15adder</td>
<td>10</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>mod32adder</td>
<td>15</td>
<td>154</td>
<td>154</td>
</tr>
<tr>
<td>mod64adder</td>
<td>26</td>
<td>333</td>
<td>333</td>
</tr>
<tr>
<td>ham7</td>
<td>21</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>nth_prime4_inc</td>
<td>11</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>nth_prime5_inc</td>
<td>25</td>
<td>103</td>
<td>103</td>
</tr>
<tr>
<td>nth_prime6_inc</td>
<td>55</td>
<td>667</td>
<td>667</td>
</tr>
<tr>
<td>Shift10_fixed</td>
<td>19</td>
<td>1198</td>
<td>-</td>
</tr>
<tr>
<td>Shift15_fixed</td>
<td>30</td>
<td>3500</td>
<td>-</td>
</tr>
<tr>
<td>Shift28_fixed</td>
<td>56</td>
<td>14310</td>
<td>-</td>
</tr>
<tr>
<td>Plus063mod4096_79</td>
<td>429</td>
<td>32539</td>
<td>-</td>
</tr>
<tr>
<td>Plus063mod8192_80</td>
<td>492</td>
<td>45025</td>
<td>-</td>
</tr>
<tr>
<td>Plus127mod8192_78</td>
<td>910</td>
<td>73357</td>
<td>-</td>
</tr>
</tbody>
</table>

GC-minimization solutions published and improves both QC and GC on S0ne245. Among the remaining twelve problems, VLEA_RC obtains smaller \(c\) and \(c'\) on eight problems and dominates both the GC-minimization and QC-minimization best results on five problems. VLEA_RC obtains the inferior solution to the best known one on nth_prime5_inc.

VLEA_RC performs effectively for most test problems except for hwb series and nth_prime*_inc series. The two series have large \(diffTerm\) and \(maxConNum\) compared with other benchmarks of the same size and have strong interactions between their variables. The latter may lead the factors subtracted from a PPRM lose efficacy.

Some of the examples of the obtained circuits are published in Table 6.

### 5.4 Comparison of VLEA_RC with and without jump growth

**Pre-Experimental Planning**

For the sake of brevity, we use VLEA_RC and VLEA_RC-JG to represent VLEA_RC with and without population update respectively. Both algorithms adopt MSR as equality constraint control mechanism.
TABLE 6
Some examples of the circuits obtained from VLEA-RC.

<table>
<thead>
<tr>
<th>hub5</th>
<th>hub6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(3,1)C(0,1)T(4,2,0)T(2,1,0)N(4)C(4,3)T(3,1,0)N(4)C(4,2)N(3)T(3,2,4)T(2,1,3)C(1,2)C(4,2)T(2,0,1)C(4,3)T(3,1,0)C(3,1)T(2,0,3)C(4,2)T(3,1,4)T(2,0,3)C(1,2)C(4,1)T(4,1,0)C(3,0)C(2,0)C(3,4)C(0,3)C(3,0)C(4,2)T(4,0)C(3,2)</td>
<td></td>
</tr>
<tr>
<td>C(0,2)N(0)C(2,4)C(1,5)C(4,0)C(3,4)C(5,4)C(1,3)T(5,3,1)T(2,1,0)C(0,2)T(2,0,1)N(3)C(4,1)T(2,1,0)C(1,0)N(5)T(3,1,2)C(2,5)N(3)T(3,1,5)T(5,1,3)T(2,0,1)T(4,2,0)T(5,1,3)C(1,2)N(1)T(3,1,5)C(0,3)T(4,3,0)C(0,3)T(5,4,2)C(2,0)C(2,5)C(2,4)T(3,1,5)N(0)C(0,2)C(1,2)T(5,1,3)C(5,1)C(3,1)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5one013</th>
<th>5one245</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(4,0)N(3)C(3,2)C(3,1)C(1,3)C(2,0)N(3)T(4,0,3)C(3,4)T(4,0,1)C(4,3)C(0,4)N(0)T(3,0,4)T(2,1,3)C(2,1)T(4,1,0,2)N(2)</td>
<td></td>
</tr>
<tr>
<td>C(2,1)C(3,2)N(1)C(1,2)C(0,3)C(2,0)C(4,0)C(3,4)T(4,1,0,3)T(5,3,2,1,0,4)C(1,4)T(1,0,2)C(4,1)C(1,0)C(1,3)N(1)C(0,2)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>majority5</th>
<th>mod5adder</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(4,3,0)T(3,1,0)C(4,3)T(2,0,3)C(3,0)T(3,0,1)T(3,2,0)T(3,1,2)T(2,0,3)C(4,2)T(3,2,4)C(4,3)C(4,1)T(4,0,1)T(4,1,0,4)T(4,2,3)</td>
<td></td>
</tr>
<tr>
<td>T(3,0,1)N(2)N(1)C(1,0)T(5,1,0,2)T(3,2,0)T(3,1,0,2)C(4,2,1)T(1,0,2)T(4,2,0)T(1,0,2)C(1,0)T(5,2,0)N(2)N(1)T(5,0,1)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mod15adder</th>
<th>mod32adder</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(6,2,3)C(6,2)C(7,3)C(5,1)T(4,0,5)C(5,1)T(4,5,2,1,3)T(4,2,0,3)T(5,1,2)T(4,0,5)C(4,0)C(5,1)</td>
<td></td>
</tr>
<tr>
<td>T(8,3,4)C(8,3)C(9,4)T(5,0,1)T(7,3,2,4)T(7,2,3)C(7,2)T(5,0,6)T(5,3,2,1,4)T(5,5,3,2,0,4)T(4,5,0,3)T(4,6,2,1,3)T(6,1,2)T(5,0,2)T(5,0,6)C(6,1)C(5,0)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mod64adder</th>
<th>plus63mod4096_79</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(10,4,5)T(6,0,7)C(10,4)T(9,4,3,5)T(6,0,1)T(9,3,4)C(9,3)C(11,5)T(8,4,3,2,5)T(4,8,3,2,4)T(8,2,3)C(8,2)T(5,7,4,3,2,1,5)T(6,4,3,2,0,5)T(5,6,3,2,0,4)T(5,7,3,2,1,4)T(4,7,2,1,3)T(4,6,2,0,3)T(6,0,2)C(7,1,2)T(6,0,7)C(6,0)C(7,1)</td>
<td></td>
</tr>
<tr>
<td>NOT(0)C(0,1)T(1,0,2)T(2,1,0,3)T(6,10,9,8,7,6,11)T(5,9,8,7,6,10)T(3,2,1,0,4)T(6,4,3,2,1,0,5)T(8,6,5,4,3,2,1,0,11)T(5,11,9,8,7,10)T(8,6,5,4,3,2,1,0,11)T(5,11,9,8,7,10)T(8,6,5,4,3,2,1,0,9)N(6)T(7,5,4,3,2,1,0,6)T(12,10,9,8,7,6,5,4,3,2,1,0,11)</td>
<td></td>
</tr>
</tbody>
</table>

**Task**
We want to verify that VLEA-RC can obtain a higher feasible ratio and a faster convergence speed than VLEA-RC-JG does, and meanwhile it does not decrease the quality of solutions very much.

**Experimental Setup**
For the parameters (maxGen, popSize, pc, pm, pf and ρ) common to both algorithms, we set the same values for all test benchmarks except ρ. ρ is
set to an intermediate value between $2^{\maxConNum} - 3$ and $2^{\maxConNum+1} - 3$, which provides a very small parsimony pressure to control chromosome bloat and accordingly help us focus on studying the effect of JG. We use max-Gen as the termination condition and give adequate evolutionary time for the slow convergence of VLEA_RC-JG. maxGen is set to 20000 for all problems apart from shift* inc and plus*modu*; for which, maxGen is 40000 and 3000 respectively. Thirty repeats are performed. The results are compared according to five criteria (Gm, Fr, best, mean, and st. dev.). Gm is the mean generations for finding the best solutions in all runs. Fr is given as the ratio of the number of runs during which feasible solutions are obtained to the total number of runs. best, mean and st. dev. are computed based on the objective values of the feasible solutions found out of the 30 runs.

Results and Discussion
The results are listed in Table 7. Only sixteen representative reversible benchmarks are considered.

VLEA_RC-JG does not find feasible solutions in all runs on 12 problems (hwb6, mod32adder, mod64adder, ham7, nth_prime5 inc, nth_prime6 inc, shift* fixed and plus*modu*). Among which, VLEA_RC finds feasible solutions in all runs on 5 problems (mod32adder, mod64adder, plus*modu*), in more than half of the runs on 2 problems (ham7 and nth_prime5 inc) and in less than 10 runs on 3 problems (nth_prime6 inc, shift15 fixed and shift28 fixed). For the remaining 4 problems, VLEA_RC not only obtains feasible solutions in all runs, but also converged faster than VLEA_RC-JG.

VLEA_RC improves the convergence speed and obtains more feasible solutions on all test problems. It outperforms VLEA_RC-JG on mod5adder, mod15adder in terms of best and mean and is only inferior to VLEA_RC-JG on 5one245 in terms of best and mean.

5.5 Comparison between VLEA_RC with SR and VLEA_RC with MSR
Task
The aim is to compare VLEA_RC with different constraint solving methods: SR and MSR. The hypothesis is tested: VLEA_RC with MSR can perform better than VLEA_RC with SR in terms of best and mean.

Experimental Setup
All parameter settings in addition to $\rho$ are same as those in the previous experiment. The value of $\rho$ varies with each benchmark. The results are compared according to the five criteria (Gm, Fr, best, mean, and st. dev.). The effect of different value of $\rho$ on VLEA_RC with MSR is studied in the next experiment.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>$G_m$</th>
<th>$F_r$</th>
<th>best</th>
<th>mean</th>
<th>st.dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No JG</td>
<td>JG</td>
<td>No JG</td>
<td>JG</td>
<td>No JG</td>
</tr>
<tr>
<td>hw6</td>
<td>-</td>
<td>2834</td>
<td>0.0000</td>
<td>0.3333</td>
<td>-</td>
</tr>
<tr>
<td>5one245</td>
<td>1646</td>
<td>836</td>
<td>0.6000</td>
<td>1.0000</td>
<td>63</td>
</tr>
<tr>
<td>majority5</td>
<td>1821</td>
<td>875</td>
<td>0.4333</td>
<td>1.0000</td>
<td>104</td>
</tr>
<tr>
<td>mod5adder</td>
<td>6803</td>
<td>369</td>
<td>0.4667</td>
<td>1.0000</td>
<td>83</td>
</tr>
<tr>
<td>mod15adder</td>
<td>446</td>
<td>213</td>
<td>0.1000</td>
<td>1.0000</td>
<td>74</td>
</tr>
<tr>
<td>mod32adder</td>
<td>-</td>
<td>461</td>
<td>0.0000</td>
<td>1.0000</td>
<td>-</td>
</tr>
<tr>
<td>mod64adder</td>
<td>-</td>
<td>815</td>
<td>0.0000</td>
<td>1.0000</td>
<td>-</td>
</tr>
<tr>
<td>ham7</td>
<td>-</td>
<td>738</td>
<td>0.0000</td>
<td>0.9333</td>
<td>-</td>
</tr>
<tr>
<td>nth_prime5_inc</td>
<td>-</td>
<td>1973</td>
<td>0.0000</td>
<td>0.8333</td>
<td>-</td>
</tr>
<tr>
<td>nth_prime6_inc</td>
<td>-</td>
<td>4804</td>
<td>0.0000</td>
<td>0.2333</td>
<td>-</td>
</tr>
<tr>
<td>Shift10_fixed</td>
<td>-</td>
<td>8540</td>
<td>0.0000</td>
<td>0.4667</td>
<td>-</td>
</tr>
<tr>
<td>Shift15_fixed</td>
<td>-</td>
<td>20868</td>
<td>0.0000</td>
<td>0.2333</td>
<td>-</td>
</tr>
<tr>
<td>Shift28_fixed</td>
<td>-</td>
<td>33005</td>
<td>0.0000</td>
<td>0.1000</td>
<td>-</td>
</tr>
<tr>
<td>Plus63mod4096_79</td>
<td>-</td>
<td>688</td>
<td>0.0000</td>
<td>1.0000</td>
<td>-</td>
</tr>
<tr>
<td>Plus63mod8192_80</td>
<td>-</td>
<td>536</td>
<td>0.0000</td>
<td>1.0000</td>
<td>-</td>
</tr>
<tr>
<td>Plus127mod8192_78</td>
<td>-</td>
<td>1465</td>
<td>0.0000</td>
<td>1.0000</td>
<td>-</td>
</tr>
</tbody>
</table>

**TABLE 7**  
Comparison between VLEA_RC with and without jump growth.
Results and Discussion
VLEA_RC with MSR achieves better $Fr$, $best$ and $mean$ on all benchmarks except for $nth\_prime5\_inc$ and $shift28\_fixed$, and acquires better $st.dev.$ on 11 problems, as listed in Table 8. However, VLEA_RC with MSR needs a longer average convergence time on 15 problems.

MSR can efficiently balance the decreasing of CV and the increasing of objective value. It prefers the decreasing of CV by a slight cost increasing, so we can obtain better solutions through the slow detailed search.

5.6 The Impact of $\rho$ on MSR

Task
The main objective is to observe the influence of $\rho$ value on the algorithm performance and to determine how to set a proper value for $\rho$.

Experimental Setup
Testing all benchmarks under all possible $\rho$ is time consuming and of slight significance, so we test one randomly selected function, $5one013$, with $\rho$ increasing from 5 to 30 to get an intuitive understanding of the impact of $\rho$ on $Fr$, $mean$ and $best$. In theory, $\rho$ can be set to any integer greater than zero. While, for a specific function, such as $5one013$, the $maxConNum$ is 4 and the cost of the gate used to build the circuit may be 1, 5, 13 and 29 according to (3). For the value of $\rho$ less than 5 or larger than 30, the parsimony pressure of MSR is too high or too low, so we set $\rho$ between 5 and 30. Thirty runs were performed for each value of $\rho$.

Results and Observations
Figure 5 shows the variation of feasible ratio with $\rho$ changing from 5 to 30 for $5one013$. $Fr$ rises with the increasing of $\rho$ and reaches 1 when $\rho$ is equal to and larger than 10. Figure 6 shows the variation of $mean$ with different $\rho$. $mean$ declines with $\rho$ changing from 5 to 8 and rises on the whole along with local oscillation when $\rho$ changing from 9 to 15, then keeps tiny increasing trend with turbulence when $\rho$ is larger than 15. Whether for a too small (close to 5) or a too large (larger than 15) $\rho$, $mean$ is comparatively large. The difference is that the feasible ratio for the former is much less than that for the latter. This can be interpreted through the fact that the bloat control effect by MSR is too strong when $\rho$ is over small, thus the decreasing of CV is only possible through large increasing of objective value under small probability. Figure 7 illustrates the variation of $best$ with different $\rho$. Smaller objective values can be obtained when $\rho$ is between 8 and 13. From the above, MSR can work well when $\rho$ is between 8 and 11 in terms of $Fr$, $mean$ and $best$.

The impact of parameter $\rho$ on the remaining test benchmarks obeys the same law. We suggest applying a medium or little strong parsimony pressure,
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>$G_m$</th>
<th>$F_r$</th>
<th>best</th>
<th>mean</th>
<th>st.dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SR</td>
<td>MSR</td>
<td>SR</td>
<td>MSR</td>
<td>SR</td>
</tr>
<tr>
<td>hwb6</td>
<td>7779</td>
<td>2834</td>
<td>0.1667</td>
<td>0.1667</td>
<td>275</td>
</tr>
<tr>
<td>sone245</td>
<td>1014</td>
<td>969</td>
<td>1.0000</td>
<td>1.0000</td>
<td>64</td>
</tr>
<tr>
<td>majority5</td>
<td>862</td>
<td>1034</td>
<td>1.0000</td>
<td>1.0000</td>
<td>84</td>
</tr>
<tr>
<td>mod5adder</td>
<td>411</td>
<td>396</td>
<td>1.0000</td>
<td>1.0000</td>
<td>83</td>
</tr>
<tr>
<td>mod15adder</td>
<td>113</td>
<td>187</td>
<td>1.0000</td>
<td>1.0000</td>
<td>74</td>
</tr>
<tr>
<td>mod32adder</td>
<td>314</td>
<td>461</td>
<td>1.0000</td>
<td>1.0000</td>
<td>185</td>
</tr>
<tr>
<td>mod64adder</td>
<td>341</td>
<td>766</td>
<td>1.0000</td>
<td>1.0000</td>
<td>309</td>
</tr>
<tr>
<td>ham7</td>
<td>846</td>
<td>738</td>
<td>0.8667</td>
<td>0.9333</td>
<td>56</td>
</tr>
<tr>
<td>nth_prime5_inc</td>
<td>1351</td>
<td>1579</td>
<td>0.9000</td>
<td>0.8000</td>
<td>153</td>
</tr>
<tr>
<td>nth_prime6_inc</td>
<td>3068</td>
<td>3989</td>
<td>0.0667</td>
<td>0.1667</td>
<td>590</td>
</tr>
<tr>
<td>Shift10_fixed</td>
<td>8157</td>
<td>8540</td>
<td>0.4000</td>
<td>0.4667</td>
<td>287</td>
</tr>
<tr>
<td>Shift15_fixed</td>
<td>19348</td>
<td>19324</td>
<td>0.3667</td>
<td>0.3667</td>
<td>551</td>
</tr>
<tr>
<td>Shift28_fixed</td>
<td>12269</td>
<td>25047</td>
<td>0.0667</td>
<td>0.1000</td>
<td>1065</td>
</tr>
<tr>
<td>Plus63mod4096_79</td>
<td>63</td>
<td>688</td>
<td>1.0000</td>
<td>1.0000</td>
<td>8266</td>
</tr>
<tr>
<td>Plus63mod8192_80</td>
<td>421</td>
<td>536</td>
<td>1.0000</td>
<td>1.0000</td>
<td>16582</td>
</tr>
<tr>
<td>Plus127mod8192_78</td>
<td>425</td>
<td>465</td>
<td>1.0000</td>
<td>1.0000</td>
<td>16457</td>
</tr>
</tbody>
</table>

**TABLE 8**
Comparison between VLEA \_RC with SR and VLEA \_RC with MSR.
which corresponds to a value of \( \rho \) from the median of \( 2^{\text{maxConNum}} - 1 - 3 \) and \( 2^{\text{maxConNum}} - 3 \) to \( 2^{\text{maxConNum}} - 3 \). Moreover, we have done the similar experiments about how to set \( step \) and to discover the relationship between \( step \) and \( \rho \). The details of all these experimental results may be obtained from the author upon request.
6 CONCLUSIONS AND FUTURE WORKS

In this article, we presented VLEA_RC, a new variable-length chromosome evolutionary algorithm for quantum cost optimization of reversible Toffoli networks. VLEA_RC combines the heuristic information from the PPRM expressions of a reversible function with the global search capability of EAs. It does this by conducting a periodic population update operation, called jump growth, in order to explore the modified search space. It was shown to improve the convergence speed and the feasible ratio. Heuristic information is used throughout this algorithm to improve efficiency. For example, diffTerm is used to estimate the maximum length of the chromosomes; prefLib can be used during population initialization phase and during the jump growth phase; maxConNum is used to set the upper limit of the number of control bits of randomly generated gates. Further, a modified constraint solving method MSR is proposed which can strike a better balance between decreasing the constraint violation and increasing the objective value so that the average quantum cost of the circuit is decreased.

In the experimental study section, we verify our conjecture about MSR and JG. Experimental results show that VLEA_RC can solve problems not only with small scale, but also with larger scale and higher complexity when compared with other EAs for reversible circuit synthesis. It obtained solutions which are better than or at least comparable to the best known solutions as of late 2013. The software can potentially synthesize functions with more than 20 variables, but as the number of variables grows, such as for
benchmarks: \textit{shift28\_fixed}, \textit{nth\_prime6\_inc} and \textit{hw6}, the feasible ratio drops quickly.

There are a number of possible directions for future research: a diversity keeping scheme in variable length EAs for reversible circuit synthesis; a combination of an EA with other methods, such as the decomposition method [8], to enable the synthesis of large and complex reversible functions; a method that would reduce the quantum cost of the generated circuits by introducing other types of gates, such as Peres gates [13] or mixed-polarity Toffoli gates [42].

REFERENCES


