J. of Mult.-Valued Logic & Soft Computing, Vol. 25, pp. 643–671
Reprints available directly from the publisher
Photocopying permitted by license only

A Variable-Length Chromosome Evolutionary Algorithm for Reversible Circuit Synthesis

XIAOXIAO WANG[1,2,*], LICHENG JIAO[1], YANGYANG LI[1], YUTAO QI[1] AND JIANSHE WU[1]

[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
[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_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].

* 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. Groeße *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. Szyprowski *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

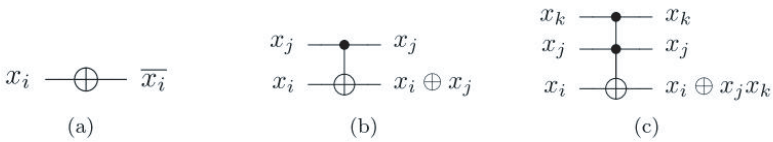

FIGURE 1
(a) NOT (b) CNOT (C) TOFFOLI

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 $C \subset X$ 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).

$$\begin{array}{l} a_o = 1 \oplus b \oplus ab \oplus c \oplus bc \\ b_o = 1 \oplus a \oplus b \oplus c \\ c_o = 1 \oplus a \oplus c \oplus ac \oplus bc \end{array} \tag{1}$$

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.

$$\mathrm{qc}\,(rc) = \sum_{i=1}^{\mathrm{len}(rc)} \mathrm{gqc}\,(g_i) \tag{2}$$

Where 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).

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

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).

$$\mathrm{cv}(rc) = \mathrm{diffTerm}(reduced\ PPRM) \tag{4}$$

The function diffTerm() returns the number of different terms between the $reducedPPRM$ and the PPRM of the identity function with the same size. The $reducedPPRM$ is obtained through substituting the sequence of reversible gates of 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 $reducedPPRM$ in (5).

$$\begin{aligned} a_o &= 1 \oplus ab \oplus c \oplus bc \\ b_o &= a \oplus b \oplus c \\ c_o &= a \oplus ac \oplus bc. \end{aligned} \tag{5}$$

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.

3 MOTIVATION AND RELATED WORK

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

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], ε-constraint [35], Stochastic Ranking (SR) [27], and etc. Giving absolute priority to CV decreasing, just as SF and ε-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 ρ, 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 VCL_GA [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

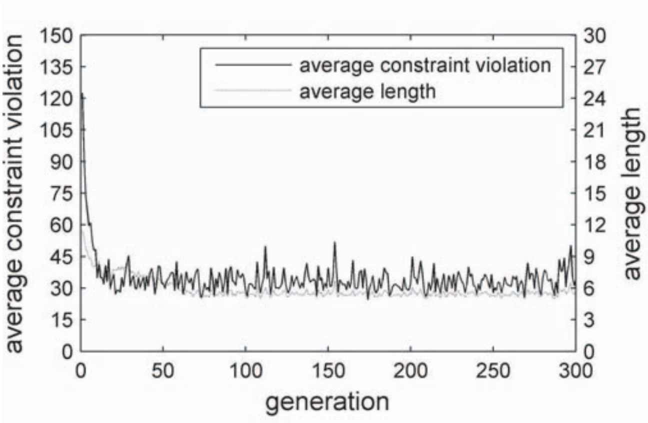

FIGURE 2
Variation of average CV and average length during the first 300 generations for reversible function *ham7*.

opportunity to search for solutions in a more correlated landscape, i.e. all chromosomes in nearby generations will bo 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, *ham7*. 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.

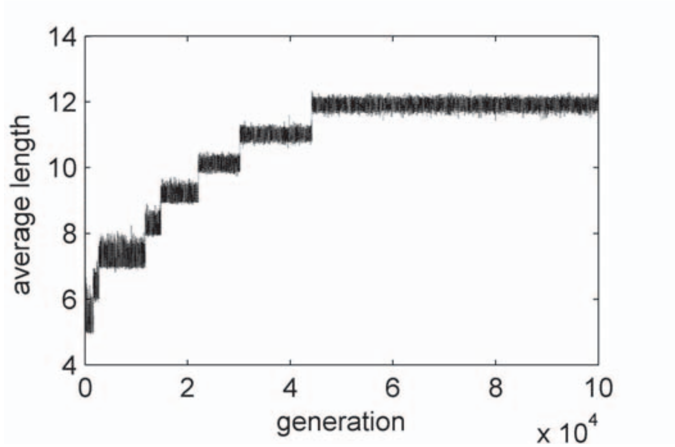

FIGURE 3
Variation of average length with generations for reversible function *ham7*.

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

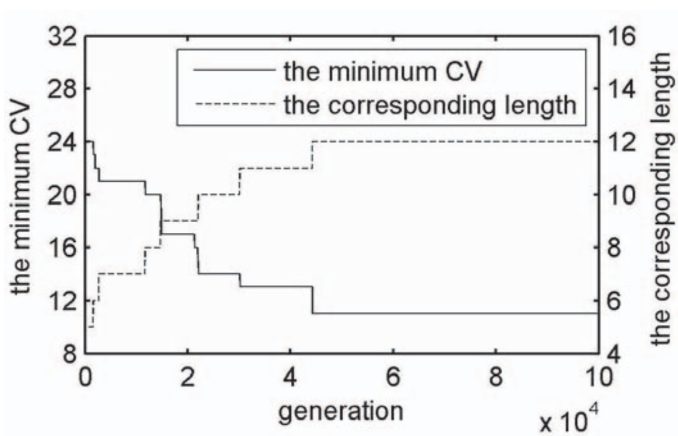

FIGURE 4
Variation of constraint violation and length of the best individual with generations for reversible function *ham7*.

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). $last\_mincv$ and $last\_mincost$ 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 $prefLib$ 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 $preferLib$ 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;

construct initial $preferLib$
detect $maxConNum$, $factorNum$ and $diffTerm$;
estimate the maximum length $maxLen$ and the initial length $initialLen$ of individuals;
initialize population $P(t)$;
compact and evaluate individuals of $P(t)$;
sort individuals using MSR;
min_cv equals to the CV of the best individual;
min_cost equals to the cost of the best individual;
$last\_mincv = min\_cv$;
$last\_mincost = min\_cost$;
$u = 0$;
while stop criterion is not met **do**
    if $u == step$ **then**
        update P(t) to P(t+1) using jumping growth;
    else
        selection,crossover and mutation according to p_c and p_m;
        generate $P(t+1)$;
    end if
    compact and evaluate the individuals in $P(t+1)$;
    sort individuals in $P(t+1)$ **using MSR**;
    elitism reservation;
    update min_cv and min_cost;
    if $(last\_mincv > min\_cv)$ or $(last\_mincv == min\_cv$ and $last\_mincost > min\_cost)$ **then**
        $last\_mincv = min\_cv$;
        $last\_mincos = min\_cost$;
    else
        $u = u + 1$;
    end if
    $t = t + 1$;
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).

$$initialLen = \begin{cases} maxLen/2 + \mathrm{rand}(0,5) & maxLen \le 30 \\ maxLen/3 + \mathrm{rand}(0,5) & 30 < maxLen \le 50 \\ 15 + \mathrm{rand}(0,5) & maxLen > 50 \end{cases} \tag{6}$$

$diffTerm$ 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 $\mathrm{T}(C, t)$ 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 $\mathrm{T}(C, t)$; 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 ρ (line 7), it means that an improvement by one unit in CV is worth an increase of QC by at most ρ 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 ρ. The larger the ρ 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, ρ;

Ensure:

Ranked population $P(t)'$;

I_j is the jth individual;

```
for i := 1 to n do
    for j := 1 to n do
        sample u ∈ U(0, 1)
        if cv(I_j) ≥ cv(I_{j+1}) and qc(I_j) ≥ qc(I_{j+1}) then
            swap(I_j, I_{j+1});
        else if cv(I_j) > cv(I_{j+1}) and qc(I_j) < qc(I_{j+1}) then
            if (qc(I_{j+1}) − qc(I_j))/(cv(I_j) − cv(I_{j+1})) < ρ) then
                swap(I_j, I_{j+1});
            else if u < p_f then
                swap(I_j, I_{j+1});
            end if
        else if cv(I_j) < cv(I_{j+1}) and qc(I_j) > qc(I_{j+1}) then
            if (qc(I_{j+1}) − qc(I_j))/(cv(I_j) − cv(I_{j+1})) > ρ) then
                if u > p_f then
                    swap(I_j, I_{j+1});
                end if
            end if
        end if
    end for
    if no swap done then
        break;
    end if
end for
```

infeasible and its real length, $realLen$, is less than $maxLen$, 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, $addLen$, is determined by $unit$ and $diffLen$ (lines 4-8). $unit$ is the default extending length. $diffLen$ is the difference between $maxLen$ and $realLen$ of an individual. The gates appended to the tail of I_j can be selected from $prefLib^{I_j}$ (lines 11-13) or generated randomly (lines 15-17). $prefLib^{I_j}$ is generated by subtracting the preferential gates from the $reducedPPRM$ 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;
update the preferential gate library of I_j, $prefLib^{I_j}$;
compute the real length of I_j, $realLen$;
$diffLen = maxLen - realLen$;
if $diffLen > unit$ **then**
  $addLen = unit$;
else
  $addLen = diffLen$;
end if
generate I'_j by duplicating I_j
if $rand()\%5 == 1$ **then**
  for $i := realLen$ to $realLen + addLen$ **do**
    append a random gate from $prefLib^{I_j}$ to I'_j;
  end for
else
  for $i := realLen$ to $realLen + addLen$ **do**
    append a randomly generated gate to I'_j;
  end for
end if
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

popSize	TCH $\boldsymbol{p_c}$	TCH $\boldsymbol{p_m}$	TCH $\boldsymbol{p_f}$	***maxLen***	TCH $\boldsymbol{\rho}$	***step***	***unit***
100, 300, 500	0.6	0.2	0.2	$factorNum+$	related to	100	10
				20, 50	$maxConNum$	100	10

TABLE 1
Considered parameter values for VLEA_RC.

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 ρ 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 verity the effectiveness of VLEA_RC on small scale problems, which carries out synthesis and optimization simultaneously.

Experiments setup

We set ρ 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.

Func.	[41] (g, c)	Our (g, c)	Time [s]	Example Circuit for QCmin
App2.1	(10, 30)	(11, 31)	26.85	C(1,3)C(0,1)T(3,0,2)C(2,3)C(1,2)N(2) T(3,1,0,2)T(3,2,1)C(0,3)C(0,2)N(3)
App2.2	(18, 102)	(12, 40)	49.07	N(2)T(3,0,1)T(3,2,1,0)N(3)C(3,0)T(1,0,2) C(3,1)C(3,0)T(2,1,3)**T(3,0,1)C(0,3)**N(0)
App2.3	(13, 43)	(12, 32)	21.82	C(0,2)N(1)T(1,0,3)C(3,1)T(2,1,0)**C(3,0) T(3,0,2)**C(1,3)**T(3,0,1)C(3,0)**T(3,1,0)C(2,1)
App2.4	(9, 36)	(10, 34)	27.17	N(3)N(1)T(2,1,3)**T(3,0,2)C(3,0)**T4(3,2,0,1) **T(3,1,0)C(3,1)**C(1,2)N(2)
App2.5	(10, 50)	(9, 29)	32.56	C(1,3)C(1,0)C(3,1)T(2,0,3)T(3,1,0)T(2,0,3) C(1,2)T(3,0,2)T(3,2,1)
App2.6	(6, 14)	(4, 12)	0.36	T(2,0,1)T(1,0,2)C(2,0)C(0,1)
App2.7	(15, 59)	(9, 29)	7.29	T(2,0,1)**C(1,0)T(1,0,2)**T(3,2,0)T(3,1,2)C(0,2) NOT(1)T(1,0,3)N(2)
App2.8	(15, 53)	(11, 43)	28.82	N(2)T(3,0,2)T(1,0,3)**C(2,1)T(2,1,3)** T(3,2,1,0)T(1,0,2)C(2,1)T(3,0,2)N(1)N(0)
App2.9	(11, 47)	(13, 33)	30.78	C(2,0)**T(3,1,2)C(1,3)**N(0)C(0,1)**C(1,2)T(2,1,3)** C(2,1)C(0,1)T(3,2,0,1)C(3,2)C(0,3)N(0)
App2.10	(13, 57)	(10, 38)	50.62	T(1,0,3)C(0,3)C(2,1)T(3,1,2)T(2,1,3)C(0,2) C(3,1)T(3,2,1,0)C(3,2)T(2,1,3)
App2.11	(12, 80)	(11, 31)	22.44	N(2)**T(2,0,1)C(0,2)**T(3,1,0)C(0,1)**C(0,2) T(2,0,3)**T(3,1,0)C(2,3)T(2,0,3)N(2)
App2.12	(17, 53)	(8, 32)	15.98	**C(2,0)T(2,0,3)**T(3,2,0,1)N(1)**C(3,2)T(3,2,0)** C(1,2)T(2,0,3)
App2.13	(12, 52)	(13, 45)	89.86	**C(1,3)T(3,1,2)**N(3)C(0,3)C(1,0)T(2,0,3)C(3,0) T(3,1,2)C(2,0)T(2,1,0)C(0,1)T(3,0,1)T(2,1,0,3)

TABLE 2
Comparison between the best results from VLEA_RC and those from [41].

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.

Func.	[13] (g, c, c')	Our (g, c, c')	Times [s]	Example Circuit for QCmin
G1	(4, 12, 12) (6, 14, 10)	(4, 12, 12)	4.86	C(2,0)T(3,1,2)CNOT(2,0)T(3,2,1)
G2	(6, 14, 14) (8, 16, 12)	(6, 14, 14)	3.94	C(0,2)CNOT(1,2)T(3,2,1)T(3,1,0)C(0,2) C(1,2)
G3	(7, 15, 15) (8, 16, 12)	(7, 15, 15)	4.94	C(1,2)T(3,2,1)C(1,2)C(1,0)C(3,1)T(3,0,1) C(1,0)
G4	(9, 21, 17) (10, 22, 16)	(9, 21, 19)	12.07	C(0,1)C(3,0)T(3,1,0)C(0,1)C(2,0)T(3,0,2) **T(2,0,3)C(2,0)**C(3,0)
G5	(8, 24, 20) (10, 26, 18)	(7, **23**, 23)	13.52	NOT(1)T(3,1,2)T(3,2,1)C(0,1)T(3,1,0) T(2,0,3) C(2,3)
G6	(9, 25, 19) (10, 26, 18)	(9, 25, 23)	11.90	N(0)**C(3,1)T(3,1,2)**C(0,1)T(3,1,0)C(0,1) T(2,0,3)T(3,2,0)N(0)
G7	(7, 23, 21) (9, 26, 19)	(7, 23, 23)	29.36	C(0,1) T(3,1,0)C(3,0)CNOT(0,1)T(3,0,2) T(3,2,0)T(2,0,3)
G8	(9, 33, 31) (11, 27, 21)	(11, 27, 25)	32.49	T(3,1,2)C(0,1)T(3,1,0)C(0,1)C(3,2)C(2,3) C(1,3)T(2,0,1)**C(1,3)T(3,1,0)**C(2,1)
G9	(9, 25, 21) (11, 27, 21)	(9, 25, 23)	23.44	C(0,1)T(3,1,0)C(1,2)**C(3,0)T(3,0,2)**C(0,1) T(3,2,0)N(0) T(2,0,3)
G10	(9, 29, 27) (12, 32, 21)	(12, **28**, 28)	37.50	C(3,0)C(0,1)T(3,1,0)T(2,0,3)T(3,0,2)C(2,0) C(0,1)C(3,1)T(2,1,0)C(3,2)C(1,3)C(2,1)
G11	(10, 34, 30) (11, 27, 25)	(11, 27, 25)	57.18	C(2,1)**T(3,1,2)C(3,1)**C(2,3)C(1,0)T(3,0,2) T(2,1,0)T(3,0,1)C(1,0)C(3,2)C(2,0)
G12	(12, 36, 28) (12, 36, 28)	(15, **35**, 33)	49.16	T(3,0,2)C(2,3)C(1,0)T(3,1,2)C(2,3)C(2,1) C(0,3)T(1,0,2)T(3,2,0)C(0,3)C(0,2)**T(3,1,0) C(1,3)** C(0,2)C(2,3)

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 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.

Func.	***size***	***factorNum***	***diffTerm***	***maxconNum***
3_17	3	9	13	2
hwb4	4	12	32	2
hwb5	5	35	90	3
hwb6	6	102	192	4
5one013	5	29	68	4
5one245	5	51	112	4
majority5	5	27	54	4
mod5adder	6	48	76	5
mod15adder	8	15	15	4
mod32adder	10	31	31	5
mod64adder	12	63	63	6
ham7	7	22	33	3
shift10_fixed	12	50	80	3
shift15_fixed	17	75	120	3
shift28_fixed	30	140	224	3
nth_prime4_inc	4	16	26	3
nth_prime5_inc	5	37	65	4
nth_prime6_inc	6	82	150	5
Plus63mod4096_79	12	441	441	11
Plus63mod8192_80	13	504	504	12
Plus127mod8192_78	13	889	889	12

TABLE 4
Considered benchmark functions and the attribute values.

5.3 Comparison of the best results from VLEA_RC and the best known as of late 2013

In this experiment, the benchmark functions with 3 to 30 variables are tested to verify the performance of VLEA_RC. The best known results in terms of GC minimization and QC minimization are published at http://www.revlib.org, which are coming from different reversible circuit synthesis algorithms, such as [2], [3, 7, 8] and [14].

Pre-Experimental Planning

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

Task

Before we go into analysis of VLEA_RC, we want to verify the effectiveness of VLEA_RC on benchmarks more than 4 bits.

Results and Visualization

We can see the results in Table 5. VLEA_RC reaches the best known solution on *3_17*, improves QC on the eight benchmarks with only

Func.	Best GC_{min}			Best QC_{min}			VLEA_RC		
	g	**c**	**c'**	**g**	**c**	**c'**	**g**	**c**	**c'**
3_17	6	14	12	6	14	12	6	14	12
hwb4	11	23	21	13	25	**19**	11	23	21
hwb5	24	104	104	38	90	80	33	**77**	**73**
hwb6	42	140	140	47	107	107	42	**102**	**100**
5one013	19	95	95		-		19	**59**	**57**
5one245	20	104	104		-		**17**	**61**	**61**
majority5	16	104	104		-		16	**68**	**64**
mod5adder	15	83	83	17	77	77	18	**70**	**68**
mod15adder	10	71	71	10	71	71	13	**57**	**53**
mod32adder	15	154	154		-		21	**129**	**123**
mod64adder	26	333	333		-		35	**203**	**197**
ham7	21	65	65	25	49	49	**21**	49	49
nth_prime4_inc	11	53	53	14	34	26	14	34	26
nth_prime5_inc	25	103	103	36	**98**	**80**	25	103	103
nth_prime6_inc	55	667	667	55	667	667	57	**549**	**545**
Shift10_fixed	19	1198	-	-	-	-	51	**223**	**223**
Shift15_fixed	30	3500	-	-	-	-	93	**481**	**481**
Shift28_fixed	56	14310	-	-	-	-	183	**1027**	**1027**
Plus63mod4096_79	429	32539	-	429	32539	-	**21**	**6785**	**6781**
Plus63mod8192_80	492	45025	-	492	45025	-	**20**	**16580**	**16576**
Plus127mod8192_78	910	73357	-	910	73357	-	**19**	**16455**	**16451**

TABLE 5
Comparison between the best reported results as of late 2013 and the best results from VLEA_RC.

GC-minimization solutions published and improves both QC and GC on *5one245*. 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.

$hwb5$
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)C(4,0)C(3,2)
$hwb6$
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(5,3,1)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)
$5one013$
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)T4(3,2,1,4)T(4,0,1) C(4,3)C(0,4)N(0) T(3,0,4)T(2,1,3)C(2,1) T4(4,1,0, 2)N(2)
$5one245$
C(2,1)C(3,2)N(1)C(1,2)C(0,3)C(2,0)C(4,0)C(3,4)T4(2,1,0,3)T5(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)
$majority5$
T(4,3,0)T(4,3,1)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)T4(3,1,0,4) T(4,2,3)
$mod5adder$
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)T(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)
$mod15adder$
T(6,2,3)C(6,2)C(7,3)C(5,1)T(4,0,5)C(5,1)T4(5,2,1,3)T4(4,2,0,3)T(5,1,2)T(4, 0,5)T(4,0,2) C(4,0)C(5,1)
$mod32adder$
T(8,3,4)C(8,3)C(9,4)T(5,0,1)T4(7,3,2,4)T(7,2,3)C(7,2)T(5,0,6)T5(6,3,2,1,4) T5(5,3,2,0,4) T4(5,2,0,3)T4(6,2,1,3)T(6,1,2)T(5,0,2)T(5,0,6)C(6,1)C(5,0)
$mod64adder$
T(10,4,5)T(6,0,7)C(10,4)T4(9,4,3,5)T(6,0,1)T(9,3,4)C(9,3)C(11,5)T5(8,4,3,2, 5)T4(8,3,2,4) T(8,2,3)C(8,2)T5(7,4,3,2,1,5)T6(6,4,3,2,0,5)T5(6,3,2,0,4)T5(7, 3,2,1,4)T4(7,2,1,3)T4(6,2,0,3) T(6,0,2)T(7,1,2)T(6,0,7)C(6,0)C(7,1)
$plus63mod4096\_79$
NOT(0)C(0,1)T(1,0,2)T4(2,1,0,3)T6(10,9,8,7,6,11)T5(9,8,7,6,10)T5(3,2,1,0, 4)T6(4,3,2,1,0,5) T8(6,5,4,3,2,1,0,11)T5(11,9,8,7,10)T8(6,5,4,3,2,1,0,11)T5(11,9,8,7,10)T4(8,7,6,9) T9(7,6,5,4,3,2,1,0,8)T(7,6,8)T8(6,5,4,3,2,1,0,7)C(6,7) T10(8,7,6,5,4,3,2,1,0,9)N(6) T7(5,4,3,2,1,0,6)T12(10,9,8,7,6,5,4,3,2,1,0,11)

TABLE 6
Some examples of the circuits obtained from VLEA_RC.

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*, p_c, p_m, p_f 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 *maxGen* 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*mod**). Among which, VLEA_RC finds feasible solutions in all runs on 5 problems (*mod32adder*, *mod64adder*, *plus*mod**), 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 ρ are same as those in the previous experiment. The value of ρ varies with each benchmark. The results are compared according to the five criteria (G_m, Fr, $best$, $mean$, and $st.\ dev.$). The effect of different value of ρ on VLEA_RC with MSR is studied in the next experiment.

	G_m		F_r		*best*		*mean*		*st.dev.*	
Benchmark	**No JG**	**JG**	**No JG**	**JG**	**No JG**	**JG**	**No JG**	**JG**	**No JG**	**JG**
hwb6	-	2834	0.0000	0.3333	-	161	-	240.4000	-	90.4699
5one245	1646	836	0.6000	1.0000	63	64	65.3333	69.0667	4.1312	4.9126
majority5	1821	875	0.4333	1.0000	104	84	127.0000	125.5000	12.7475	21.1667
mod5adder	6803	369	0.4667	1.0000	83	83	102.4286	99.1667	20.4628	18.6457
mod15adder	446	213	0.1000	1.0000	74	58	74.0000	73.8667	0.0000	3.2135
mod32adder	-	461	0.0000	1.0000	-	134	-	181.9000	-	23.5860
mod64adder	-	815	0.0000	1.0000	-	322	-	440.4333	-	52.6876
ham7	-	738	0.0000	0.9333	-	55	-	71.4643	-	6.1853
nth_prime5_inc	-	1973	0.0000	0.8333	-	160	-	210.2400	-	28.5545
nth_prime6_inc	-	4804	0.0000	0.2333	-	789	-	823.8571	-	29.1171
Shift10_fixed	-	8540	0.0000	0.4667	-	273	-	317.8571	-	24.6852
Shift15_fixed	-	20868	0.0000	0.2333	-	517	-	557	-	27.6677
Shift28_fixed	-	33005	0.0000	0.1000	-	1101	-	1127.3333	-	28.7460
Plus63mod4096_79	-	688	0.0000	1.0000	-	8266	-	82 77.6000	-	14.0703
Plus63mod8192_80	-	536	0.0000	1.0000	-	16580	-	16925.0000	-	554.6798
Plus127mod8192_78	-	1465	0.0000	1.0000	-	16455	-	16991.0000	-	1009.3000

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 ρ on MSR

Task

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

Experimental Setup

Testing all benchmarks under all possible ρ is time consuming and of slight significance, so we test one randomly selected function, $5one013$, with ρ increasing from 5 to 30 to get an intuitive understanding of the impact of ρ on Fr, $mean$ and $best$. In theory, ρ 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 ρ less than 5 or larger than 30, the parsimony pressure of MSR is too high or too low, so we set ρ between 5 and 30. Thirty runs were performed for each value of ρ.

Results and Observations

Figure 5 shows the variation of feasible ratio with ρ changing from 5 to 30 for $5one013$. Fr rises with the increasing of ρ and reaches 1 when ρ is equal to and larger than 10. Figure 6 shows the variation of $mean$ with different ρ. $mean$ declines with ρ changing from 5 to 8 and rises on the whole along with local oscillation when ρ changing from 9 to 15, then keeps tiny increasing trend with turbulence when ρ is larger than 15. Whether for a too small (close to 5) or a too large (larger than 15) ρ, $mean$ 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 ρ 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 ρ. Smaller objective values can be obtained when ρ is between 8 and 13. From the above, MSR can work well when ρ is between 8 and 11 in terms of Fr, $mean$ and $best$.

The impact of parameter ρ on the remaining test benchmarks obeys the same law. We suggest applying a medium or little strong parsimony pressure,

	G_m		F_r		*best*		*mean*		*st.dev.*	
Benchmark	**SR**	**MSR**	**SR**	**MSR**	**SR**	**MSR**	**SR**	**MSR**	**SR**	**MSR**
hwb6	7779	2834	0.1667	0.1667	275	161	352.4000	240.4000	39.1269	90.4699
5one245	1014	969	1.0000	1.0000	64	61	70.4000	68.5667	5.3449	5.0679
majority5	862	1034	1.0000	1.0000	84	74	132.1000	111.8667	21.3000	19.2475
mod5adder	411	396	1.0000	1.0000	83	76	136.4000	90.6667	75.0764	8.7112
mod15adder	113	187	1.0000	1.0000	74	57	75.5000	73.6667	4.1735	3.3356
mod32adder	314	461	1.0000	1.0000	185	129	205.5000	175.5667	19.3349	27.2165
mod64adder	341	766	1.0000	1.0000	309	203	447.7667	357.8000	88.0869	66.4465
ham7	846	738	0.8667	0.9333	56	55	71.8462	71.4643	8.1182	6.1853
nth_prime5_inc	1351	1579	0.9000	0.8000	153	129	246.0000	192.0417	50.9671	35.0853
nth_prime6_inc	3068	3989	0.0667	0.1667	590	549	610.5000	598.2000	28.9914	30.8010
Shift10_fixed	8157	8540	0.4000	0.4667	287	273	371.1667	317.8571	55.3236	13.4288
Shift15_fixed	19348	19324	0.3667	0.3667	551	481	610.0000	532.5455	37.5588	36.8368
Shift28_fixed	12269	25047	0.0667	0.1000	1065	1027	1071.5000	1056.6667	9.1924	26.5016
Plus63mod4096_79	631	688	1.0000	1.0000	8266	6785	8273.2000	8217.7670	8.8723	270.6268
Plus63mod8192_80	421	536	1.0000	1.0000	16582	16580	16777.0000	16925.0000	568.5109	554.6798
Plus127mod8192_78	425	465	1.0000	1.0000	16457	16455	16961.0000	16991.0000	1090.9000	1009.3000

TABLE 8
Comparison between VLEA_RC with SR and VLEA_RC with MSR.

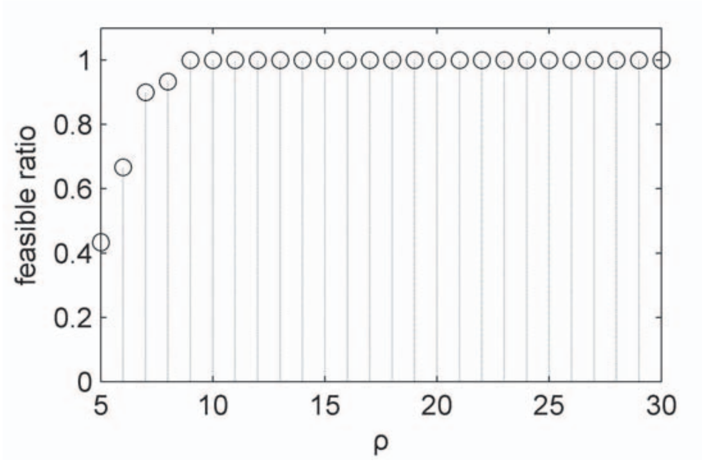

FIGURE 5
Variation of feasible ratio with parameter ρ for 5one013, thirty runs were performed for each value of ρ.

which corresponds to a value of ρ from the median of $2^{maxConNum-1} - 3$ and $2^{maxConNum} - 3$ to $2^{maxConNum} - 3$. Moreover, we have done the similar experiments about how to set $step$ and to discover the relationship between $step$ and ρ. The details of all these experimental results may be obtained from the author upon request.

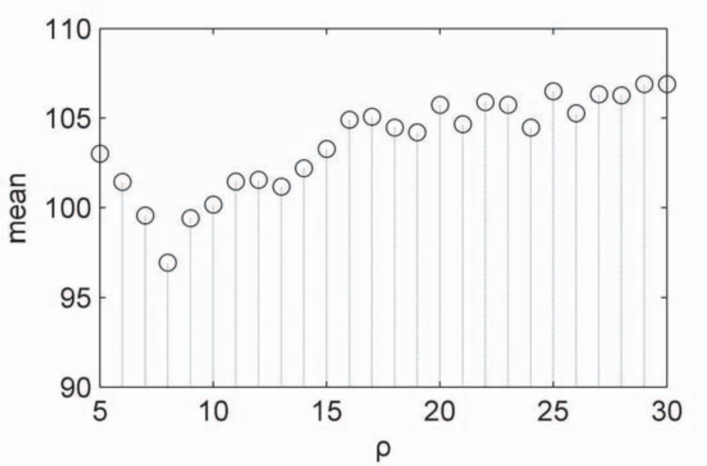

FIGURE 6
Variation of the mean of the objective values of feasible solutions with parameter ρ for 5one013, thirty runs were performed for each value of ρ.

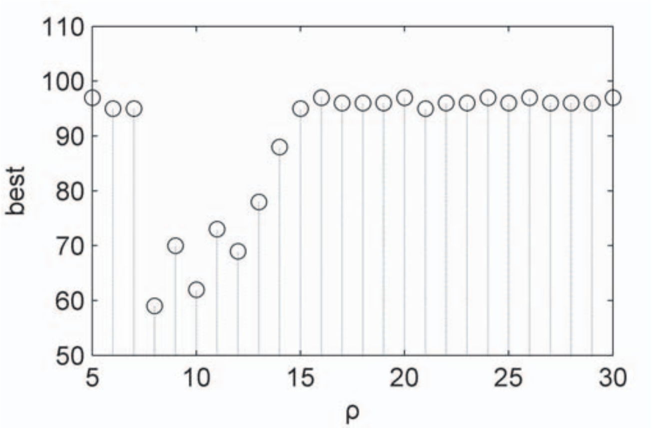

FIGURE 7
Variation of the minimum objective value of feasible solutions with parameter ρ for 5*one*013, thirty runs were performed for each value of ρ.

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: $shift28\_fixed$, nth_prime6_inc and $hwb6$, 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

[1] Charles H Bennett. (1973). Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532.

[2] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. (2003). A transformation based algorithm for reversible logic synthesis. In *Design Automation Conference, 2003. Proceedings*, pages 318–323. IEEE.

[3] Dmitri Maslov, Gerhard W Dueck, and D Michael Miller. (2007). Techniques for the synthesis of reversible Toffoli networks. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(4):42.

[4] Dmitri Maslov, Gerhard W Dueck, and D Michael Miller. (2005). Toffoli network synthesis with templates. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(6):807–817.

[5] Robert Wille and Rolf Drechsler. (2009). BDD-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference*, pages 270–275. ACM.

[6] Robert Wille and Rolf Drechsler. (2010). Effect of BDD optimization on synthesis of reversible and quantum logic. *Electronic Notes in Theoretical Computer Science*, 253(6):57–70.

[7] Mehdi Saeedi, Morteza Saheb Zamani, Mehdi Sedighi, and Zahra Sasanian. (2010). Reversible circuit synthesis using a cycle-based approach. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 6(4):13.

[8] Mehdi Saeedi, Mehdi Sedighi, and Morteza Saheb Zamani. (2010). A library-based synthesis methodology for reversible logic. *Microelectronics Journal*, 41(4):185–194.

[9] Vivek V Shende, Aditya K Prasad, Igor L Markov, and John P Hayes. (2003). Synthesis of reversible logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(6):710–722.

[10] Oleg Golubitsky and Dmitri Maslov. (2012). A study of optimal 4-bit reversible Toffoli circuits and their synthesis. *Computers, IEEE Transactions on*, 61(9):1341–1353.

[11] Daniel Große, Xiaobo Chen, and Rolf Drechsler. (2006). Exact Toffoli network synthesis of reversible logic using boolean satisfiability. In *Design, Applications, Integration and Software, 2006 IEEE Dallas/CAS Workshop on*, pages 51–54. IEEE.

[12] Daniel Große, Robert Wille, Gerhard W Dueck, and Rolf Drechsler. (2009). Exact multiple-control Toffoli network synthesis with SAT techniques. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(5):703–715.

[13] Marek Szyprowski and Pawel Kerntopf. (2011). Reducing quantum cost in reversible Toffoli circuits. *arXiv preprint arXiv:1105.5831*.

[14] Pallav Gupta, Abhinav Agrawal, and Niraj K Jha. (2006). An algorithm for synthesis of reversible logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(11):2317–2330.

[15] Colin P Williams and Alexander G Gray. (1999). *Automated design of quantum circuits*. Springer.

[16] André Leier and Wolfgang Banzhaf. (2004). Comparison of selection strategies for evolutionary quantum circuit design. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 557–568. Springer.

[17] Cristian Ruican, Mihai Udrescu, Lucian Prodan, and Mircea Vladutiu. (2009). Genetic algorithm based quantum circuit synthesis with adaptive parameters control. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 896–903. IEEE.

[18] Martin Lukac and Marek Perkowski. (2002). Evolving quantum circuits using genetic algorithm. In *Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on*, pages 177–185. IEEE.

[19] Martin Lukac, Marek Perkowski, Hilton Goi, Mikhail Pivtoraiko, Chung Hyo Yu, Kyusik Chung, Hyunkoo Jee, Byung-Guk Kim, and Yong-Duk Kim. (2004). Evolutionary approach to quantum and reversible circuits synthesis. In *Artificial intelligence in logic design*, pages 201–257. Springer.

[20] Taro Yabuki and Hitoshi Iba. (2000). Genetic algorithms for quantum circuit design-evolving a simpler teleportation circuit. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 421–425. Citeseer.

[21] Timothy Reid. (2005). On the evolutionary design of quantum circuits. Master's thesis, Univ. of Waterloo, Ontario.

[22] Shengchao Ding, Zhi Jin, and Qing Yang. (2008). Evolving quantum circuits at the gate level with a hybrid quantum-inspired evolutionary algorithm. *Soft Computing*, 12(11):1059–1072.

[23] Mingming Zhang, Shuguang Zhao, and Xu Wang. (2009). Automatic synthesis of reversible logic circuit based on genetic algorithm. In *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, volume 3, pages 542–546. IEEE.

[24] Kamalika Datta, Indranil Sengupta, and Hafizur Rahaman. (2012). Reversible circuit synthesis using evolutionary algorithm. In *Computers and Devices for Communication (CODEC), 2012 5th International Conference on*, pages 1–4. IEEE.

[25] Wen-Hsin Wang, Chia-Hui Chiu, Shu-Yu Kuo, Sheng-Fei Huang, and Yao-Hsin Chou. (2012). Quantum-inspired tabu search algorithm for reversible logic circuit synthesis. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 709–714. IEEE.

[26] Ben Hutt and Kevin Warwick. (2007). Synapsing variable-length crossover: Meaningful crossover for variable-length genomes. *Evolutionary Computation, IEEE Transactions on*, 11(1):118–131.

[27] Thomas P. Runarsson and Xin Yao. (2000). Stochastic ranking for constrained evolutionary optimization. *Evolutionary Computation, IEEE Transactions on*, 4(3):284–294.

[28] Rammohan Mallipeddi and Ponnuthurai N Suganthan. (2010). Ensemble of constraint handling techniques. *Evolutionary Computation, IEEE Transactions on*, 14(4):561–579.

[29] Kalyanmoy Deb and Rituparna Datta. (2010). A fast and accurate solution of constrained optimization problems using a hybrid bi-objective and penalty function approach. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

[30] Abu SSM Barkat Ullah, Ruhul Sarker, David Cornforth, and Chris Lokan. (2007). An agent-based memetic algorithm for solving constrained optimazation problems. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 999–1006. IEEE.

[31] Stephanus Daniel Handoko, Kwoh Chee Keong, Ong Yew Soon, and Jonathan Chan. (2011). Classification-assisted memetic algorithms for solving optimization problems with restricted equality constraint function mapping. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1209–1216. IEEE.

[32] Abu SSM Barkat Ullah, Ruhul Sarker, and Chris Lokan. (2012). Handling equality constraints in evolutionary optimization. *European Journal of Operational Research*, 221(3):480–490.

[33] Rituparna Datta and Kalyanmoy Deb. (2011). A bi-objective based hybrid evolutionary-classical algorithm for handling equality constraints. In *Evolutionary Multi-Criterion Optimization*, pages 313–327. Springer.

[34] Kalyanmoy Deb. (2000). An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2):311–338.

[35] Tetsuyuki Takahama and Setsuko Sakai. (2006). Constrained optimization by the ϵ constrained differential evolution with gradient-based mutation and feasible elites. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1–8. IEEE.

[36] Sean Luke and Liviu Panait. (2006). A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344.

[37] Hal Stringer and Annie S Wu. (2004). Winnowing wheat from chaff: The chunking GA. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 198–209. Springer.

[38] Il Yong Kim and Olivier L. De Weck. (2005). Variable chromosome length genetic algorithm for progressive refinement in topology optimization. *Structural and Multidisciplinary Optimization*, 29(6):445–456.

[39] Inman Harvey. (1992). The saga cross: The mechanics of recombination for species with variable length genotypes. In *PPSN*, page 271. Citeseer.

[40] Dominique Chu and Jonathan E Rowe. (2008). Crossover operators to control size growth in linear GP and variable length GAs. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 336–343. IEEE.

[41] Ahmed Younes. (2012). Detection and elimination of non-trivial reversible identities. *International Journal of Computer Science, Engineering & Applications*, 2(4).

[42] Marek Szyprowski and Paweł Kerntopf. (2013). Optimal 4-bit reversible mixed-polarity toffoli circuits. In *Reversible Computation*, pages 138–151. Springer.