Krzysztof PODLASKI
FACULTY OF PHYSICS AND APPLIED INFORMATICS UNIVERSITY OF LODZ,
Pomorska 149/153, 90-236 Łódź

A hybrid PSO-GA algorithm for Reversible Circuits Synthesis

Dr Krzysztof PODLASKI

Ukończył studia magisterskie i doktoranckie na Wydziale Fizyki i Chemii Uniwersytetu Łódzkiego. Obecnie pracownik Zakładu Informatyki Stosowanej Wydziale Fizyki i Informatyki Stosowanej UŁ. Zainteresowania naukowe: układy odwracalne, modelowanie i optymalizacja procesów, inżynieria oprogramowania.

e-mail: podlaski@uni.lodz.pl

Abstract

In the domain of Reversible Circuits there is still lack of good synthesis algorithms. There are many heuristic propositions, unfortunately, their results for a given reversible function usually are circuits far from optimal implementations. There are some propositions of using Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) for this purpose. In this paper a new hybrid PSO-GA algorithm is proposed. Comparison of the proposed algorithm with the existing ones gives promising results.

Keywords: reversible circuits, reversible logic synthesis, particle swarm optimization, genetic algorithms.

Hybrydowy algorytm PSO-GA dla syntezy układów odwracalnych

Streszczenie

W dobie poszukiwania układów cyfrowych o niskim zużyciu energii układy odwracalne stanowią ciekawą alternatywę dla aktualnie stosowanych układów cyfrowych. Jednym z najistotniejszych zagadnień w dziedzinie budowy układów cyfrowych jest synteza układu reprezentującego zadaną funkcję. Niestety do dzisiaj nie ma dobrych rozwiązań w dziedzinie syntezy układów odwracalnych, istniejące rozwiązania są bardzo czasochłonne bądź generują układy o dużej redundancji. Ciekawą alternatywą dla obecnie stosowanych metod heurystycznych jest wykorzystanie algorytmów ewolucyjnych np. Particle Swarm Optimization (PSO) lub algorytmów genetycznych (GA). W niniejszym artykule zaproponowano nowy hybrydowy algorytm PSO-GA dostosowany do syntezy odwracalnych układów cyfrowych. Stworzony algorytm zastosowano do syntezy układów dla wybranych funkcji testowych (tzw. benchmarków) a wyniki porównano z wynikami otrzymywanymi za pomocą algorytmów heurystycznych. Wygenerowane układy okazały się mniej redundantne niż układy otrzymane w syntezie metodami heurystycznymi.

Słowa kluczowe: układy odwracalne, synteza układów odwracalnych, particle swarm optimization, algorytmy genetyczne.

1. Introduction

In modern circuit design, the problem of energy loss is very important. The search for low energy circuits is one of the most essential challenges in the area of digital design. It is known that every loss of information during computations leads to energy dissipation (Landauer's principle [1]). Moreover, Bennet proved that one could design a circuit with zero energy dissipation only with use of reversible circuits [2]. We cannot neglect the fact that quantum computation allows solving many hard problems not in exponential but polynomial time [3]. Quantum computers are reversible by definition. This explains why scientists are interested in reversible computing.

Even though in many papers we can find proposals for reversible circuit design, none of the proposed method gives results near to optimal even for small circuits (up to 5 lines). It is worth mentioning that complexity of the task grows rapidly with the number of lines and existing methodologies are far from efficiency [4]. This problem is an important argument why different synthesis methods are needed. Some authors introduced evolutionary algorithms into this field for example: particle swarm optimization (PSO) and ant colony optimization (ACO) [5, 6]. In this paper a new approach is proposed. It is based on the PSO algorithm and uses some elements from genetic algorithms.

The paper is organized as follows. The first section is an introduction to reversible circuits design. Then there is a short description of evolutional algorithms like particle swarm optimization and genetic algorithms. In the third section a new hybrid PSO-GA algorithm is presented. Later one can find experimental tests of the presented algorithm and the conclusions at the end of the paper.

2. Reversible circuits design

Reversible circuits synthesis (even digital ones) is a very different task than the synthesis of traditional circuits using Boolean logic. It is impossible to translate the known solutions from one area to the other. For example, the reversibility requirement disallows the use of fan-outs or feedbacks in a circuit. There are many existing methods of reversible circuits synthesis, the most promising according to [7] are: cycle-based methods, transformation-based methods, exclusive or sum of product method (ESOP), binary decision diagrams methods (BDD).

2.1. Reversible Logic

Definition 1: Reversible n*n function is a bi-jection mapping with n binary inputs into n binary outputs. This implies that any reversible function can be described as permutation operation.

Definition 2: Reversible gate is an operation with n-input and n-output binary lines that realizes reversible n*n function.

Definition 3: Reversible n*n circuit is a sequence of reversible gates with no more than n inputs that realizes reversible n*n function.

It is obvious that there are many reversible circuits that realize a given reversible function, however we are always interested in optimal realization of a reversible function. In order to decide what optimal realization means, we have to define some measure to distinguish better and worse representations. There are two most acknowledged cost measures for reversible circuits: gate count and quantum cost. The first one simply counts how many gates were used in the measured circuit, the second is based on the cost of realization of reversible gates with elementary 2*2 quantum gates.

There are many basic reversible gates that can be used for a reversible circuit. In the presented paper we use only the library built from gates:

- NOT(s) – not gate that negates a signal on the selected s-th line,
- CNOT(k;s) – controlled not gate, negates a signal on s line if the signal on the k-th line is equal to 1,
- TOFFOLI(k,l;s) – toffoli gate, negates a signal on s line if the signal on lines k and l are equal to 1,
- TOFFOLI(k1,…,km; s) – generalized toffoli gate, negates a signal on s line if the signal on lines k1,…,km are equal to 1.

The selected library can be described as a library of all generalized toffoli gates (NOT gate can be described as TOFFOLI(;s) – empty set of control lines) denoted later as NCT library. Each of the presented gates has well defined quantum cost (qc): NOT and CNOT qc=1, TOFFOLI gate with two control line has qc=5 and qc=13 for TOFFOLI gate with tree control lines [8].

In some cases it is possible to reduce the gates count or the quantum cost using some additional lines (called garbage lines). However, it is not obvious how to compare the presented costs

measures with the hardware cost of an additional line in a reversible circuit. In the algorithm presented in this paper no additional lines are used.

3. Particle swarm optimization

Particle swarm optimization is an algorithm proposed by Kennedy and Eberhart in [9]. Basically PSO is an evolutionary algorithm that uses a population of test particles (swarm) in order to search space of all possible solutions for a given problem. For every analyzed case we have to define the fitness function to be optimized. In every iteration, the new position of a particle is based on the actual position, the best known position in history of this particle and the best known position in the swarm history. Historically, PSO was first introduced to minimize functions over n dimensional real space $\mathbb{R}^n$. In this case we can intuitively describe the change of particle position p_i using velocity v_i:

$$\vec{p}_{k,i+1} = \vec{p}_{k,i} + \vec{v}_{k,i}, \quad (1)$$

$$\vec{v}_{k,i+1} = c_1 * \vec{v}_{k,i} + c_2 * (\vec{p}_k^{\,best} - \vec{p}_{k,i}) + c_3 * (\vec{p}^{\,Gbest} - \vec{p}_{k,i}), \quad (2)$$

where: $\vec{p}_k^{\,best}$ - is the best known position of the k-th particle, $\vec{p}^{\,Gbest}$ - is the best known position in the swarm, c_1, c_2, c_3 - are the algorithm parameters, k – index of the particle in the swarm, i – number of the current iteration.

The PSO algorithm can be defined:

Algorithm 1 (PSO Algorithm)
begin
  foreach $\vec{p}_k$ in Swarm **do**
    initialize position;
  do
    foreach $\vec{p}_k$ in Swarm **do**
      begin
        calculate new position according to (1) and (2)
        update particle position
        calculate fitness function $ft(\vec{p}_k)$
        if $[ft(\vec{p}_k) > ft(p_k^{best})]$ **then** $p_k^{best} = \vec{p}_k$
        if $[ft(\vec{p}_k) > ft(p^{Gbest})]$ **then** $p^{Gbest} = \vec{p}_k$
      end
  until(maximum iterations reached)
end

The PSO algorithm can be redefined in the cases where the space of a possible solution is not continuous. There are PSO implementations for discrete spaces i.e. travelling salesman problem [10]. We have to remember that in discrete spaces, like in the space of all permutations, it is very difficult to introduce velocity with as intuitive interpretation as in $\mathbb{R}^n$. Remembering that the most important task in the PSO is to describe the change of the particle position that depends on $\vec{p}_k^{\,best}$ and $\vec{p}^{\,Gbest}$ there exists some approaches using, for example, the genetic crossover operator [11].

In genetic algorithms, the evolution of population is defined based on crossover/recombination operators that mix the state of particles in the population. The crossover operator usually mixes only two parent states. In the PSO one could create a recombination to mix more parents at once. This can be solved using the multi-parent crossover operator like in [12].

4. PSO-GA algorithm for reversible circuit design

At the beginning, in order to use the PSO in the reversible circuit design, we have to define a particle and its position. In the proposed approach, the particle will be a representation of the desired reversible n*n function f. The position of the particle is reversible circuit C_m that is a sequence of gates $C_m = \{g_1, g_2, \dots, g_m\}$, where each of $g_i, (i = 1, \dots, m)$ is a reversible gate from the NCT library, all gates g_i have no more than n inputs. For every circuit C_m we can count the cost (gate count or quantum cost).

Now we can define a new rule for the particle position change

$$\vec{p}_{k,i+1} = \vec{p}_{k,i} \otimes \vec{p}_k^{\,best} \otimes \vec{p}^{\,Gbest}, \quad (3)$$

where $\otimes$ is the crossover operator. As was mentioned previously, one can define one recombination operator to mix $\vec{p}_{k,i}$, $\vec{p}_k^{\,best}$ and $\vec{p}^{\,Gbest}$ at once, in this paper that possibility was omitted.

4.1. Random gate sequence generation

Before definition of our crossover operator we have to solve the problem of particle positions initialization. This process will be based on a randomly generated sequence of gates that represents the given n*n reversible function f defined via the algorithm:

Algorithm 2 (Random gate sequence generation)
begin
  do
    randomly select line number i, 1 <i<n
    randomly select number of controlled lines q, 0<q<n-1
    randomly select q control lines $k_1, k_2, \dots, k_q$
    append to sequence C gate TOFFOLI$(k_1, k_2, \dots, k_q; i)$
  until(C realizes given function f)
end

In order to create better circuits during random selection of i, we check if this line still needs to be changed (using hamming distance). This prevents addition of new gates, which changes the state of lines that already have proper output values.

4.2. Crossover operator for reversible circuit design

Especially for Reversible circuit design, we can define a new crossover operator that mixes positions of two particles. If we have two positions $\overrightarrow{p_1} = \{g_1, g_2, \dots, g_m\}$ and $\overrightarrow{p_2} = \{g'_1, g'_2, \dots, g'_r\}$ representing the same n*n reversible function f, we can define $\otimes$ operator as the result of the algorithm

Algorithm 3 ($\otimes$, Crossover operator)
begin
  randomize s where 1<s<m
  randomize p,q, where 1<p,q<Min(m,r)/2
  create new sequence of gates C
  copy gates $\{g'_s, g'_{s+1}, \dots, g'_{s+p}\}$ into sequence C
  copy gates $\{g_{s+p}, g_{s+p+1}, \dots, g_{s+p+r}\}$ into sequence C
  do
    append random gates g'' to sequence C
  until(C realizes given function f)
end

The resulting sequence has the form:

$$C_{p+q+w} = \{g'_s, g'_{s+1}, \dots, g'_{s+p}, g_{s+p}, g_{s+p+1}, \dots, g_{s+p+r}, g''_1, \dots, g''_w\}$$

where w is the number of the randomized gates at the end of the algorithm. This random filling of the sequence has a similar algorithm as the random gate sequence generation presented earlier (Algorithm 2).

4.3. Definition of the PSO-GA algorithm

Now after defining the crossover operator and the particle position initialization, we can describe the PSO algorithm for the reversible circuit design. At the beginning, we have to choose the fitness function. In our experiment tests the gate count measure was used. However, the change of measure into the quantum costs would not change the algorithm.

Algorithm 4 (PSO-GA Algorithm)
begin
  foreach $\vec{p}_k$ in Swarm **do**
    initialize position;
  do
    foreach $\vec{p}_k$ in Swarm **do**
      begin
        calculate new position according to (3) and Algorithm 3
        update particle position
        calculate fitness function $ft(\vec{p}_k)$
        if $[ft(\vec{p}_k) > ft(p_k^{best})]$ **then** $p_k^{best} = \vec{p}_k$
        if $[ft(\vec{p}_k) > ft(p^{Gbest})]$ **then** $p^{Gbest} = \vec{p}_k$
      end
  until(maximum iterations reached)
  end

Comparing the proposed algorithm with the existing PSO implementations [5] we can see a difference in the particle position update. In the cited work the authors do not require from all particles to implement the given function f. This means that during update they mix circuits that represents different functions. This algorithm using the hamming measure tends to implement function f. In the proposed (in this paper) approach all particles always implement the given function, and that means we search using only proper implementations of function f. This implies that during recombination we try to select (propagate) the best gates, which leads to implementation of the given function f.

5. Experimental results

The proposed PSO-GA algorithm was tested on six selected functions (hwb4, hwb5, hwb6, nth_prime4, mod10_171, dmasl) from well-known reversible benchmarks [13].

For all the presented pso tests the swarm always contained 30 particles and each run of the algorithm had the maximum number of iterations equal to 3000. The obtained results was compared with the results obtained using RevKit [14] for Transformation based, ESOP based and BDD based synthesis algorithms. The best known solution is taken from [13].

Tab. 1. Wyniki testów zaproponowanego algorytmu PSO-GA
Tab. 1. Experimental results of the proposed PSO-GA algorithm (gate counts)

Benchmark	Best GC	PSO-GA	Transf. based	ESOP	BDD
hwb4	11	12	19	58	39
hwb5	23	40	58	137	87
hwb6	27	112	164	312	161
nth_prime4	11	12	15	58	38
mod10_171	9	9	9	58	27
dmasl	9	11	15	58	44

Analyzing Table 1. we can see that the proposed algorithm gives better results than the existing heuristic approaches. However, the obtained results are not the best possible existing solutions. This leads to the conclusion that the PSO-GA is a good candidate for future research in the area. The proposed algorithm has many places where it could be improved, and the presented results can be treated as the proof of the concept. Using the PSO in the reversible circuit design is a promising approach. Unfortunately, it is hard to compare these results with different PSO approach [5]. The authors used mostly 3*3 functions and only hwb4 is in their paper, the result for this function is the same (circuit with 12 gates).

6. Conclusions

The proposed PSO-GA algorithm was compared with the results obtained using well known and acknowledged reversible circuits design algorithms. The results in Table 1 show that the proposed algorithm gives better results for selected 4*4, 5*5, 6*6 reversible functions. Analyzing the proposed PSO-GA algorithm one can find many places where some innovations can be introduced in the future. For example, during the procedure of random gate sequence generation (Algorithm 2) one can introduce a better way to select on which line the next gate should operate.

7. References

[1] Landauer R.: Irreversibility and Heat Generation in the Computing Process, IBM Journal, Vol. 5 pp 183-191, 1961.
[2] Bennett C. H.: Logical reversibility of computation, IBM Journal of Research and Development, vol. 17, no. 6, pp. 525-532, 1973.
[3] Nielsen M, Chuang I.: Quantum Computation and quantum Information, Cambridge University Press, 2000.
[4] Kerntopf P., Perkowski M., Podlaski K.: Synthesis of Reversible Circuits: A View on the State-of-the-Art, in Nanotechnology (IEEE-NANO), 2012 12th IEEE Conference on. IEEE, pp. 1-6, 2012.
[5] Datta, K., Sengupta, I., Rahaman, H.: Particle Swarm Optimization Based Circuit Synthesis of Reversible Logic. In Electronic System Design (ISED), 2012 International Symposium on. IEEE. pp. 226-230, 2012.
[6] Li, M., Zheng, Y., Hsiao, M. S., & Huang, C.: Reversible logic synthesis through ant colony optimization. In Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, pp. 307-310, 2010.
[7] Saeedi M., Markov I. L.: Synthesis and optimization of reversible circuits – a survey, ACM Computing Surveys, Vol. 45, Issue 2, 2012.
[8] Maslov D. , Dueck G.W.: Improved Quantum Cost for n-bit Toffoli Gates, IEE Electronic Letters, vol. 39, 2003, no 25.
[9] Kennedy J., Eberhart R.: Particle swarm optimization. In Proceedings of IEEE international conference on neural networks, Vol. 4, no. 2, pp. 1942-1948, 1995.
[10] Poli R.: Analysis of the publications on the applications of particle swarm optimisation. Journal of Artificial Evolution and Applications, 685175, 2008.
[11] Hu X., Eberhart R.C., Shi Y.: Swarm intelligence for permutation optimization: A case study of n-queens problem, in: Proceedings of the 2003 IEEE Swarm Intelligence Symposium, (2003).
[12] Eiben, A. E., Raue, P. E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In Parallel Problem Solving from Nature-PPSN III, pp. 78-87, 1994.
[13] Maslov D.: Reversible Logic Synthesis Benchmarks Page, http://www.cs.uvic.ca/~dmaslov.
[14] Soeken M., Frehse S., Wille R., Drechsler R.: RevKit: A toolkit for reversible circuit design, Proc. Workshop on Reversible Computation, 2010, pp. 69-72, available at http://www.revkit.org.

otrzymano / received: 12.04.2014
przyjęto do druku / accepted: 02.06.2014 *artykuł recenzowany / revised paper*