Reversible Circuit Synthesis Using ACO and SA based Quine-McCluskey Method

Mayukh Sarkar[1], Prasun Ghosal[1,2], Saraju P. Mohanty[2]
[1] Department of Information Technology, Bengal Engineering and Science University, Shibpur, WB 711103, India
[2] Department of Computer Science and Engineering, University of North Texas, TX 76203, USA
Email: *Prasun.Ghosal@unt.edu*

ABSTRACT

Reversible circuits are becoming more and more important in terms of computing for present and future days. However, due to several factors, known synthesis approaches of classical Boolean logic like Karnaugh Map and Quine-McCluskey method cannot be applied directly to synthesize a reversible logic. In this work, a stochastic procedure to synthesize a reversible circuit has been proposed. This procedure is based on a modified version of classical Quine-McCluskey method and is being used under the wrapper of two intelligent stochastic search techniques, Simulated Annealing and Ant Colony Optimization.

INTRODUCTION

Reversible Function

An n-input, n-output Boolean function *f* is reversible if it maps each input uniquely to each output, and vice-versa, i.e., there is an one-to-one mapping between i/p and o/p.

Requirements of Reversible Function

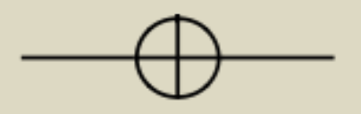
(1) Tremendous growth of VLSI technology is causing transistor size to touch the subatomic dimension in DSM, where laws of classical physics fail. It forces us to think about quantum physics based computation. Quantum gates are, by nature, reversible.

(2) Irreversible computation suffers from consistent information loss. According to Bennett, zero energy dissipation is possible only with reversible computing.

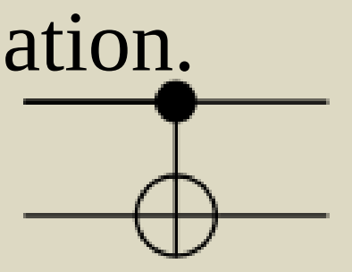
(3) Other than the quantum computation, there are several applications of reversible computing in optical computing, DNA computing, low-power CMOS design etc.

Reversible Gates

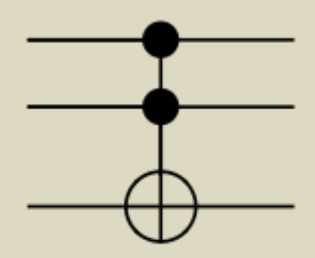
(1) There are several universal reversible gate libraries, among which NCT (NOT, CNOT, TOFFOLI) library is used by the proposed algorithm to synthesize a reversible specification.



NOT Gate



CNOT Gate



Toffoli Gate

(2) NOT gate alters the input bit unconditionally.
(3) Toffoli gate $T(x_k\ ;\ x_1\ ,\ x_2\ ,\ \dots\ ,\ x_m\ )$ with target bit at x_k (represented by larger empty circle), and m control bits at $x_1, x_2, \dots, x_m$ (represented by smaller solid circles), flips x_k, if and only if all of $x_1\ , x_2\ , \dots\ , x_m$ are set; otherwise, x_k is left unchanged. This gate leaves the control bits unaltered.
(4) The CNOT (Controlled-NOT) gate is a Toffoli gate with single control bit.

PROPOSED ALGORITHM

For each ant → Start with a blank circuit → Add a gate to the circuit based on a transition probability → Apply SA-QM to find out minimum possible circuit → Impose constraints of length and cost of the minimum circuit (returned by SA-QM to itself and other ants coming next → further add more gates to check for other possible circuits.

Simulated Annealing Based Quine-McCluskey (SA-QM) Method

(1) Begin with T_{start}, add each gate *g* to the circuit and apply the circuit to the input truth table.
(2) From resulting truth table, determine heuristic value *h(g)* using QM and remove the gate.
(3) $best\_gate \leftarrow g$ and $heuristic \leftarrow h(g)$ is set with probability 1, if $h(g) > heuristic$, otherwise with probability $e^{(h(g)-heuristic)/temperature}$.
(4) The temperature is locally updated to be used by next gate.
(5) The current best gate is added to the circuit.
(6) The operation is repeated until, either
 (i) The circuit found at some stage synthesizes the truth table, or
 (ii) The length or cost constraint is reached, in which case the ant returns as failed.
(7) The complete operation is repeated for number of *ITERATION*. Best circuit is returned.

Calculation of heuristic *h(g)* for a gate *g*

(1) Apply gate *g* to the input truth table t_{in} to get the output truth table t_{out}.
(2) The minterm table m_b for each I/O bit *b* is created by adding the input entries in m_b, for which the bit *b* gets changed in corresponding output.
(3) Minimize minterm tables according to QM.
(4) From final minterm tables of all I/O bits, the heuristic is calculated as,

$$h(g) = 100 \times (don't\_care\_ratio + one\_bit\_ratio) - total\_length - Hamming\_distance$$

where,

$total\_length\ = total\ length\ of\ all\ tables$
$don't\_care\_ratio = (total\ number\ of\ don't\ care\ terms\ /\ total\_length\ )$
$one\_bit\_ratio = (total\ number\ of\ one\ bit\ /\ total\_length\ )$
$Hamming\ distance = Hamming\ distance\ of\ t_{out}$

Ant Colony Optimization

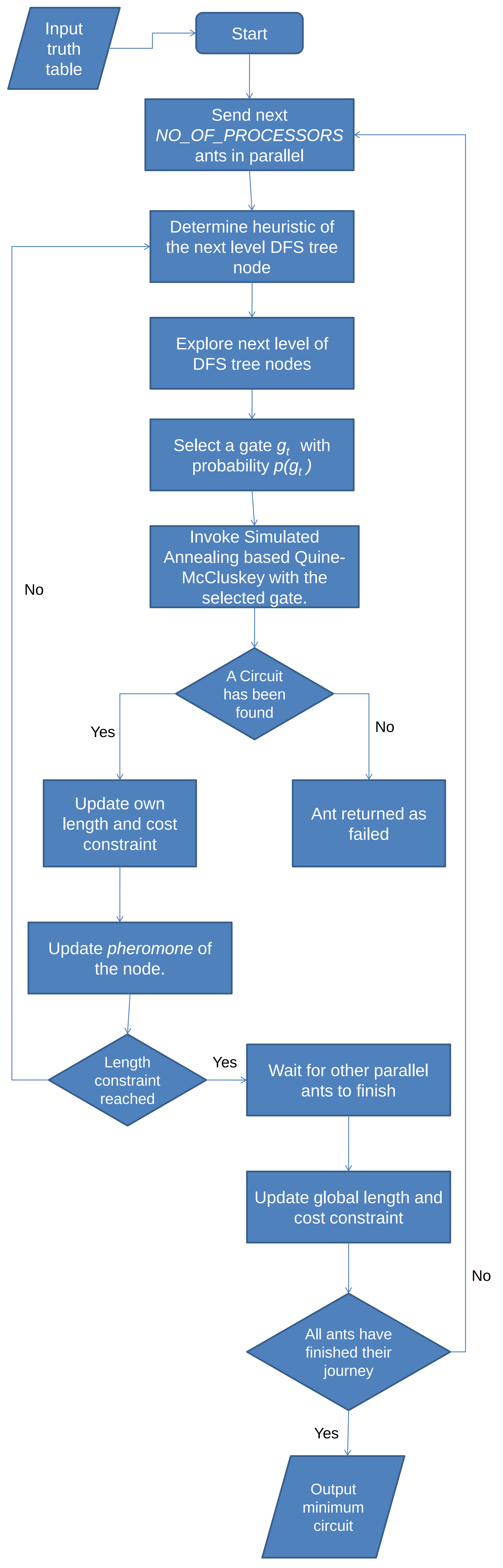
Exploring DFS Tree

(1) A *pheromone* and *heuristic* is associated with each node of the DFS tree.
(2) Initially, initialize pheromones with a fixed value *INITIAL_PHEROMONE* and calculate heuristics.
(3) Sent ants in parallel from root of the DFS tree. They explore the first level of the tree, selects a gate g_t with probability $p(g_t)$,

$$p(g_t) = \frac{pheromone_t^{\alpha} \times heuristic_t^{\beta}}{\sum_{i=1}^{b2^{(b-1)}} pheromone_i^{\alpha} \times heuristic_i^{\beta}}$$

where, $pheromone_k$ and $heuristic_k$ are pheromone and heuristic of kth node.
(4) Each ant invokes SA-QM, updates it’s own cost and length constraint.
(5) Ant updates the *pheromone* of the node, and explores the next level nodes, until the length constraint is reached. The pheromone of a node is updated as, $pheromone_t = EVAPORATION \times pheromone_t + new\_pheromone$

where, *new_pheromone* is *(1 / min_cost)*, if invoked SA-QM succeeds and returns with circuit of cost *min_cost*, otherwise *0*.
(6) Update global length and cost constraint by the minimum by n ants.
(7) Procedure continues for next n ants and so on till ants finish their journey.



Input truth table
Start
Send next *NO_OF_PROCESSORS* ants in parallel
Determine heuristic of the next level DFS tree node
Explore next level of DFS tree nodes
Select a gate g_t with probability $p(g_t)$
Invoke Simulated Annealing based Quine-McCluskey with the selected gate.
A Circuit has been found
Update own length and cost constraint
Ant returned as failed
Update *pheromone* of the node.
Length constraint reached
Wait for other parallel ants to finish
Update global length and cost constraint
All ants have finished their journey
Output minimum circuit

Heuristic Determination of DFS Tree Node

(1) The heuristic probability of some *node* is represented by the probability of adding the next gate in the circuit *c* after *node*.
(2) Assuming the result of application of *c* to the input truth table is t_{out}, the probability that a particular bit, say tth bit , will be changed first in t_{out} , is calculated as,
 For any initial permutation $(i_1, i_2, i_3, \dots, i_{2^n})$ *and final permutation* $(z_1, z_2, z_3, \dots, z_{2^n})$*, if* $j_1, j_2, \dots, j_m$ *be the indices, for which* $i_{jk} \neq z_{jk}$ *, and at these indices the permutations differ in* $b_1, b_2, \dots, b_m$ *bit positions, then for any bit t, the probability that the immediate next gate in the circuit controls the* t*th bit is,*

$$p(t) = \frac{\alpha_1(b_1-1)! + \alpha_2(b_2-1)! + \cdots + \alpha_m(b_m-1)!}{b_1! + b_2! + \cdots + b_m!}$$

where, $\alpha_k = 1$*, if* t*th bit is changed from* i_{jk} *to* z_{jk} *, and, 0, otherwise*
(3) The number of CNT gates having control at any particular bit *t* is 2^{n-1} . So the probability of next immediate gate *g* is $(p(t) / 2^{n-1})$, where *g* controls bit *t*.

Experimental Results

Function Name	Functions	Gate Count		
		MOSAIC	PPRM	Proposed Method
rand_3_1	[7,0,1,2,3,4,5,6]	3	3	3
rand_3_2	[0,1,2,3,4,6,5,7]	3	3	5
rand_3_3	[0,1,2,4,3,5,6,7]	7	5	6
rand_3_4	[1,2,3,4,5,6,7,0]	3	3	3
rand_3_5	[3,6,2,5,7,1,0,4]	8	7	8
rand_3_6	[1,2,7,5,6,3,0,4]	8	6	7
rand_3_7	[4,3,0,2,7,5,6,1]	6	7	7
rand_3_8	[7,5,2,4,6,1,0,3]	6	7	7
rand_3_9	[1,0,3,2,5,7,4,6]	4	4	5
rand_4_1	[13,1,14,0,9,2,15,6,12,8,11,3,4,5,7,10]	29	16	14
rand_4_2	[0,1,2,3,4,5,6,8,7,9,10,11,12,13,14,15]	9	7	10
rand_4_3	[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0]	4	4	4
rand_4_4	[0,7,6,9,4,11,10,13,8,15,14,1,12,3,2,5]	4	4	4
rand_4_5	[6,2,14,13,3,11,10,7,0,5,8,1,15,12,4,9]	19	15	14

Table 1 : Gate Count comparison with MOSAIC [1] and PPRM [2]

Functions	Cost		Cost Increased
	RevLib Minimum Cost	Proposed Method	
ham_3_28	9	9	0%
3_17_6	14	14	0%
4_49_7	32	36	12.5%
hwb4_12	23	26	13%

Table 2 : Cost comparison with Revlib [3]

REFERENCES

[1] M. Saeedi, M. S. Zamani, and M. Sedighi, “*Moving forward: A non-search based synthesis method toward efficient cnot-based quantum circuit synthesis algorithms.*” ASPDAC, Jan. 2008, pp. 83–88.
[2] P. Gupta, A. Agrawal, and N. Jha, “*An algorithm for synthesis of reversible logic circuits*,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, pp. 2317–2330, Nov. 2006.
[3] An online resource for reversible benchmarks. [Online]. Available: *http://www.revlib.org/*.