Application of Genetic Algorithm for Synthesis of Large Reversible Circuits using Covered Set Partitions

Maher Hawash*, Baker Abdalhaq[§], Amjad Hawash[§], Marek Perkowski*
** Electrical & Computer Engineering, Portland State University, P.O.Box 751, Portland, OR 97027*
§ College of Information Technology, An-Najah University, P.O.Box 7, Nablus, Palestine
gmhawash@gmail.com, baker@najah.edu, amjad@najah.edu, mperkows@cecs.pdx.edu

Abstract: ***We present the results of application of Evolutionary Algorithms to the problem of synthesizing quantum circuits which belong to the class of reversible circuits, represented as an input/output mapping vectors. The paper specifically focuses on large quantum circuits where many valid solutions exist in an exponentially inflating search space. Valid solutions represent the set of all input vector permutations (arrangements) which satisfy the circuit specification. The search space for circuits with large number of variables grows exponentially making it impossible to discover the set of optimal solutions. The paper compares three methods for selecting valid solutions of input vector sequences: 1) randomly, 2) genetic algorithm, 3) Tabu search. The objective function calculates the number of elementary quantum gates needed to represent the solution such that lower number of gates represents better solutions. In addition to the choice of selection algorithm, we illustrate the impact of using different partition depths for the Covered Set Partitions algorithm used to construct valid input vector sequences.***

Keywords**:** *Genetic algorithm, Tabu, random, Covering Set Partition (CSP), reversible, quantum circuits, synthesis, Hasse, covering graphs, partially ordered sets, MMD.*

I. INTRODUCTION

In 1975, Gorden Moore, the cofounder of Intel, issued his famous prediction that the number of transistors on a microchip doubles every 18 months. Surprisingly enough, Moore's prophecy has held true for the past fourty-some years; however, as the dimensions of the transistor are reaching the low tens of nanometers, the dreadful quantum effects are exhibiting their influence on the behavior of the chip. Moore's law is nearing its end! In addition to fabrication woes, heat has been one of the greatest enemies of nono-scale miniaturization pushing the thermal conductivity of the very thin copper interconnects to their limits.

In the realm of classical technology, the irreversibility of digital logic gates results in loss of information which manifests as heat dissipation. Landauer proved that using irreversible logic gates yields a rate of energy loss proportional to kT [1]. Essentially, information equals energy. Computations which preserve information are considered reversible and gates which perform reversible computation are designated as reversible gates. Bennett [2] showed that near-zero energy dissipation is possible when a computer can operate near its thermodynamic equilibrium and displayed that such a stasis state can be achieved through reversible components. Nielsen and Chuang [3] showed that quantum logic gates are inherently reversible and demonstrated a set of universal quantum primitive capable of implementing any logic circuit - namely, NCT library (Not, Controlled-Not and Taffoli gates). The *qubit* came to represent the quantum analogy of the classical symbol of information carrier: *the bit.* Possibly years before the feasibility of mass production of quantum computers, researchers have been laying the foundation for manufacturing such a computing device by exploring automated synthesis algorithms of quantum logic circuits: this is the focus of this paper.

Mathematically, the problem of automated quantum logic synthesis can be realized through the decomposition of circuit's specification to a number of small permutations of reversible gates. Currently there are various methods and assumptions which satisfy different objectives, where each algorithm builds a cascade of quantum gate primitives such that each minterm of the input vector maps to a specified minterm of the output vector. Some approach the problem as a fully specified bijective function where exists a *one-to-one* and *onto* correspondence between the input and output vectors while other researchers focus on partially specified functions such as *n-bit* adders [4]. Another body of research considers the physical constraints of interaction between *qubits* (Ion Trap or NMR) by assuming Linear Near Neighbor Model (LNNM) [5, 4] while others assume that interaction amongst any set of *qubits* is feasible [6, 7, 8, 9, 10, 11, 12, 13]. Some algorithms avoided the addition of any ancillary (*a.k.a.* garbage) *qubits* while others required the addition of such additional bits [5, 10].

The algorithm presented herein avoids the addition of extraneous output bits and does not give consideration to the LNNM model. The paper reports our latest milestone in the chain of algorithms based on Miller, Maslov and Dueck (*MMD)* [6] approach to quantum logic synthesis. Stedman and Perkowski [13] presented an algorithm capable of producing circuits with lower number of gates by exploring permutations of input vector ordering other than the natural ordering used by MMD. Stedman's method however stalls at large number of variables as it requires an exorbitant amount of time to compute. Alhagi, Hawash and Perkwoski [11] followed up with a synthesis method which explores a subset

of Stedman's orderings that produce near optimal circuits within a reasonable amount of time. Hawash, et. al. [12] explored alternative convergent sets of Stedman's orderings, dubbed *Covering Set Partitions*, which were able to discover solutions of lower quantum gate cost. This paper explores the impact of partition depth on quantum cost.

Agrawal and Jha's algorithm [7, 10] uses the number of terms in the Positive Polarity Reed-Muller (PPRM) expansion of synthesized functions as its cost function. As PPRM can be stored by an expression that is shorter than 2^n, their algorithm could, in theory, minimize larger functions. On the other hand this algorithm has to store many PPRM equations as it represents a tree-search algorithm. Non-factorized PPRMs may be, in many cases, of similar complexity to truth tables which quickly consumes resources and makes its application limited to few number of bits. Additionally, some variants of the algorithm [7, 8, 10] have trouble with convergence where a trade-off is stipulated between provable convergence and size of circuits that can be minimized.

Our main contributions of this paper are:

- The impact on quantum gate cost of using Genetic Algorithm and Tabu search compared to random selection of valid CSP sequences,
- Comparison of the performance (with respect to quantum gate cost) of various variants of the genetic algorithm (single and double cross over) and Tabu search,
- The Impact on quantum cost of varying the depth of the CSP partition used to generate valid sequences.

I. MMD Style Algorithms

In their paper, *A Transformation Based Algorithm for Reversible Logic Synthesis*, Miller, et al.[6] outlined a simple, yet powerful, synthesis method of reversible circuits. This algorithm observes a simple, yet essential, guiding principle stating that: *A completely mapped pair can never be altered by succeeding mapping calculations*. This important rule, along with inherent attribute of natural binary ordering of the input vector, allows MMD to *always* converge which is an essential principle for synthesizing arbitrary reversible circuits. The issue of convergence has been treated fully by [11, 12, 13] and, for the sake of setting context for convergence as it relates to CSP, the reader is encouraged to review [12]. Some definitions are in order before we illustrate the algorithm with an example.

Definition 1: *An n-variable* ***mapping specification*** *is a set of n variable input/output pairs (minterms), typically represented as a table, indicating the required functionality of a logic circuit (a function).*

Definition 2: *An n-variable* ***input/output pair*** *describes the expected output bit pattern for its corresponding input pattern.*

Definition 3: *A* ***completely mapped pair*** *is a pair of input/output minterms, where, at some point in the logic synthesis process, a set of quantum logic gates have been specified to map its n-variable input pattern to its corresponding n-variable output pattern.*

II. A Synthesis Example[1]

Figure 1 shows a mapping specification of a two-variable function where the inputs are designated with (***ab***) and the outputs with (***AB***). The algorithm synthesizes the function as follows:

ab	AB	⊕|	⊕|	|⊕
00	10	**0**0	00	00
01	01	**1**1	**0**1	01
10	11	**0**1	**1**1	1**0**
11	00	**1**0	10	1**1**

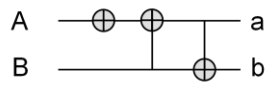

Figure 1 Synthesis of two-variable function

1. Considering the inherent reverisbility of the function, the algorithm starts synthesis from the output column (AB) towards the input column (ab).
2. Starting with the first pair (00 → 10), the algorithm realizes that an inverter on line (**a**) would correctly map the *00 input* to the *10* output. Essentially, any value presented on the (**A**) line will be inverted, as shown in the bolded text of the third column. At this stage, the first input/output pair is *completely mapped*, and according to the guiding principle mentioned above, such a pair should *never be modified* by later transformations.
3. In order to observe such a rule, the algorithm uses control lines for all subsequent synthesis as shown in the last two columns. Row two of the third column shows the pair (01 → 11) which requires an inverter on the (**A**) line with line (**B**) as a control line – shaded. As a result, only the bolded digits of second and third rows are affected.
4. Similarly, the third pair (10 → 11) is sythesized with an inverter on line (**B**) which is controlled by a value of *one (1)* on line (**A**).
5. At this stage, the algorithm realizes that the mapping circuit is complete as the first and last columns are both identical.

III. Anatomy of Covered Set Partition Algorithm

A. Structure

We hinted earlier that MMD [6] uses the natural binary order to arrange the minterms of the input vector and that such an arrangement ensures convergence. Stedman [13], Alhagi [11] and the current authors [12] documented the advantage of exploring alternative sequencing of input vector. Stedman outlined an algorithm for detecting

[1] Refer to [14] for description of quantum gates.

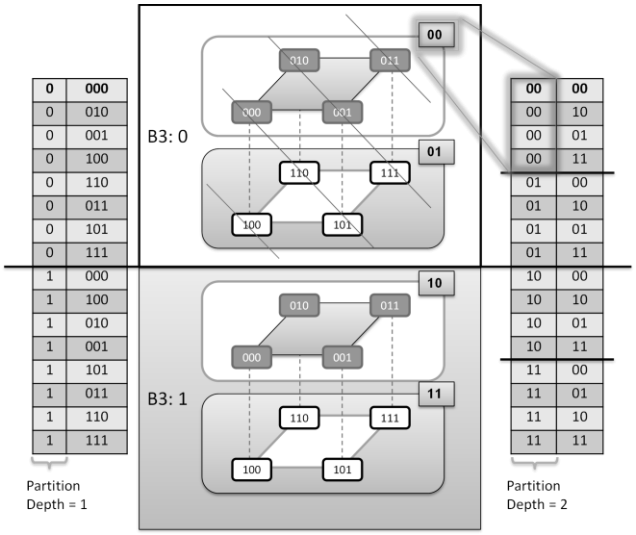

Figure 2 Covering Set Partitions using bit 3 to create two partitions of 3 bits each (upper and lower), or using bits 3-2 to create 4 partitions of 2 bits each

convergent input orderings and Alhagi devised a systematic algorithm, based on the *Hasse* diagram, for constructing valid input orderings for any number of bits and demonstrated the ability to produce circuits at lower quantum cost within a reasonable period of time. In our attempt to improve on Alhagi's work, we construct a different set of sequences based on the mathematical concept of *partially orderd sets* described below. The reader is encouraged to refer to [11, 12] for the process of constructing a valid sequence using the Hasse diagram.

Definition 4: *a* ***Hasse diagram*** *is a type of mathematical diagram used to represent a finite partially ordered set, in the form of a graph where, for the relation {(x,y) | x ≤ y | x,y ∈ S}, each element of S is a vertex in the plane and draws a line segment or curve that goes upward from x to y whenever y covers x (that is, whenever x < y and there is no z such that x < z < y).*

Figure 2 displays graphical illustrations of two variants of the *covering set partitions* method for a function of four variables. The table to the left of the graph sets a partition depth of 1 bit which is depicted graphically by the upper and lower regions labeled *(b3:0 and b3:1).* The lower half of the graph represents the partition where the highest *bit 3 = 1,* and the upper half is for the partition where *bit3 = 0.* For the remaining three bits *(b2-b0),* the algorithm uses the Hasse structure to create the sequence for each of the two halves. Notice the Hasse diagram levels are represented by the diagonal lines of the top half – see [11, 12] for more information about creating the Hasse sequence. The following ordered set represents the order of the minterms in the sequence for a partition depth of *one* (underlined).

{{0000}, {0001, 0010, 0100}, {0011, 0101, 0110},{0111},
{1000}, {1001, 1010, 1100}, {1011, 1101, 1110},{1111}}

Alternatively, a valid sequence could be constructed using a partition depth of 2 which is represented graphically by the four planes of the upper and lower surfaces of the cube and shown in the table on the right. In this case, terms with *b3b2=00* are placed at the beginning of the sequence followed by *b3b2=01, b3b2=10* and finally *b3b2=11.* The remaining two bits could still be taken according to the Hasse sequence. The following ordered set is a valid sequence for a partition depth of *two*:

{{0000}, {0001, 0010}, {0011}, {0100}, {0110, 0101}, {0111},
{1000}, {1001, 1010}, {1011}, {1100}, {1110, 1101}, {1111} }

B. Steps for creating valid sequences

Definition 5: *For a binary function of n variables, a* ***band*** *within a Hasse diagram is the set of minterms ($b_{n-1}....b_1b_0$) which have the same number of ones; i.e.,* $\{\boldsymbol{x} = \boldsymbol{b_{n-1}} \ldots \boldsymbol{b_1 b_0} \mid \forall \boldsymbol{x} \sum_{i=0}^{n-1} \boldsymbol{b_i}\ \boldsymbol{is\ the\ same}\}$.

Corollary 1: *An n-variable binary function has a total of* ***n+1*** *bands.*

The following process outlines the steps for creating CSP sequences for an *n*- variable function using the *p* upper bits for partition:

6. Create $k=2^p$ partitions where *p* is the partition depth represented by the number of upper bits resulting in the number of partitions *N=0..k-1.*
7. To construct an input sequence, place all the terms sequenctially according to their partition number *N=0..k-1.*
8. Within each partition, use the Hasse diagram to arrange the minterms within a partition as follows:
9. Start from the base level of the Hasse diagram consisting of all *zeros,*
10. Randomy permute, i.e., shuffle, terms of the next band consisting of *single ones* and place them at the end of the ordering,
11. Repeat step (b) for each band that follows in consecutive order, where each band has an *additional one* compared to the band before it, *(two ones, three ones, …),*
12. Place the last term consisting of all *ones (k-1 ones)* at the end of the sequence.

IV. Algorithmic Contest

In section III.B above, we outlined the steps for creating a single valid sequence using the CSP algorithm. We stipulated then that there exists a number of solutions in an exponentially expanding search space. In [12] we employed a random process in constructing the sequences and maintained the ones with the best cost up to that point. It was shown then, that, for a small number of variables, the CSP performed well compared to earlier attempts by [6, 11]; yet as the number of variables increased, the ability to find better solutions became dismal at best. We concluded then that the

vastness of search space hindered our ability to discover the *proverbial needle in the haystack*. In this effort, we present the results of exploring two additional alternative selection methods of the input vector sequence and compare the performance of the three methods: *Random, Genetic Algorithm and Tabu search*. It is noteworthy that we were careful to provide a fare comparison by stipulating that each method selects and synthesizes the same number of sequences and only varied the method of constructing valid sequences.

A. Objective function using Quantum Cost

The quality of a solution is measured by *quantum cost* which represents the number of elementary quantum gates used to implement the specification. For an arbitrary quantum circuit *C* with *k* quantum NCT gates, the quantum cost Q*c* is calculated as follows:

$$Q_c = \sum_{i=1}^{k} G_{qc}(i)$$

where: G_{qc} is the quantum cost of each gate in the cascade and can be calculated according to Table 1 below.

TABLE 1
QUANTUM COST OF ELEMENTARY GATES

Gate Type	*Quantum Cost*
NOT, C^1NOT	1 [3]
C^2NOT (Toffoli)	5 [14]
C^mNOT ($3 \le m \le \left\lfloor \frac{n}{2} \right\rfloor$)	12m – 22 [15]
C^mNOT ($\left\lfloor \frac{n}{2} \right\rfloor + 1 \le m \le n-2$)	24m – 64 [15]
C^{n-1}NOT	2^n – 3 [14]

B. Method 1: A Random Skip and Hop

In order to discover a solution with the lowest quantum cost, a set of *k* valid solutions are created *randomly* according to the steps outlined above. Notice that because each band is shuffled in a random manner, step 10 above, potential solutions are selected for examination in a blind manner; i.e., past solutions has no influence on the structure of new solutions. A new solution is saved only if its quantum cost is lower than the *current lowest cost;* otherwise, the solution is purged, and a new solution is randomly constructed.

```
01: cost ← MAX_INTEGER
02: k ← number of solutions to examine
03: for i := 1 to k
04: solution ← initialize();          // randomly
05: if (evaluate(solution) < cost)    // save best solution
06: best ← solution;
07: end if
08: end for
```

Although the search space grows exponentially, $(2^n)!$, there exists a very small possibility that a solution would be visited more than once. More dramatically, however, is that the odds of finding solutions with low quantum cost are equivalent to the odds of hitting the jackpot of the grand lottery.

C. Method 2: Genetic Algorithm

Rather than bouncing randomly around the search space, a genetic algorithm follows a set of directed *probabilistic* steps where *new* solutions are the *offspring* of existing *good* solutions. The following block exhibits the standard structure of a genetic algorithm:

```
01: g ← number of generations;
02: initialize(P(g));
03: do
04: evaluate(P(g));
05: P1(g), P2(g) ← select(P(g));    // Set of parent pair
06: g ← g - 1;
07: P(g) ← recombine(P1(g), P2(g)); // crossover ➔ children
08: P(g) ← mutate(P(g));            // Mutate children
09: while (g > 0);
```

The initialization and evaluation steps (02: , 04:) are exactly the same steps used in the random algorithm of step IV.B above. *Roulette wheel* selection process was used to randomly select two parents of the current generation for recombination (step 07:). *Single and double crossover operators* were used to create the offspring with certain limitations on the position of the crossover, discussed below. Finally, mutation is probabilistically applied to each offspring in order to continuously maintain population diversity and avoid premature convergence to local minima.

C-1. Genotype and Valid Operators

As discussed earlier and shown in [11, 12], the band structure of definition 6, above, must be maintained to ensure algorithmic convergence. Consequently, recombination operators are limited to the band boundary and mutation operators are limited to intra-band alterations.

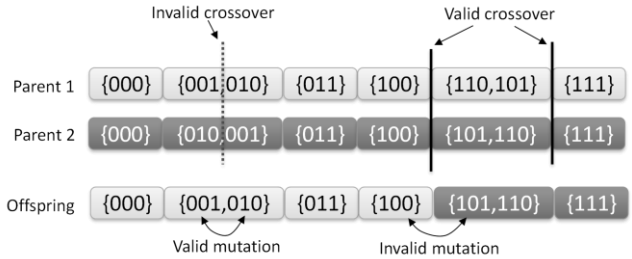

Figure 3 Genetic Algorithm genotype and valid operators

Figure 3 illustrates the structure of a chromosome for a three variable binary function with CSP partition depth of one (1). In order to ensure that a child is a valid CSP sequence, the crossover point(s) must happen at the band boundary position. Had the *invalid crossover* point been taken in Figure 3, the resultant child would have been invalid as it would have included the term *001* twice and lacked the

term *010*. Of course a repair process could have detected and corrected such a defect which would surely yield different result and could be the subject of future exploration. The reader might correctly surmise that the choice of limiting crossover to band boundary could potentially result in stale members within each band, leading to premature convergence to local minima. Such an anomaly is treated with random mutation within a band at a level higher than mutation probability suggested by standard genetic algorithms. A high level of mutation probability, we theorized, would inject diversity within children allowing them to escape such hasty local minima.

D.Method 3: Tabu Search

We also implemented the Tabu[2] search [16, 17, 18] to examine whether it would discover solutions with lower quantum cost than either the genetic algorithm or random methods. Tabu is a meta-heuristic search algorithm which, during the selection process, *forbids* search moves to solutions already visited in the past *m* steps. As a result, the algorithm is amenable to accept, temporarily, solutions with inferior quantum cost, in order to *skip*, possibly better, solutions which were *just* investigated. Such an approach, we assumed, should provide protection against the trap of falling into local minima early. The following list describes the Tabu search:

```
01: C ← {15, 20, 25, 30};      // constant for different runs
02: θ ← initialize();
03: τ ← bands /2 + C(runs);
04: do
05: evaluate(θ);
06: N(θ) ← sort( neighborhood(θ) );   // neighborhood set
07: λ ← select(N(θ)) { λ ∉ T(θ) OR T(θ) > τ };
08: T ← {θ} ;                         // add to top of taboo set
09: while (not termination-condition);
```

Unlike the genetic algorithm where an initial population of *m solutions* is created followed by generations of solutions through intra breeding and mutation, the Tabu search starts with a randomly begotten single solution, θ. At each step of the process, a *n mutations* are serially performed to create a *neighborhood of n solutions*, step 06: above using the same probabilistic intra-band swap operator of the genetic algorithm above. The selection criteria of new solutions is:

1) When a solution is selected for synthesis, it is added to the Tabu list, *T*, used to reject future encounters of the same solution (step 08:).
2) When a solution λ with a better cost than θ is found:
   a) Select λ only if it is not in the Tabu list *T* or it has not been visited for *τ* iterations (step 07:).
   b) Otherwise, select the next best solution in the neighborhood *N(θ)* according to the same criteria in I above2)a) above.
3) If all neighbor solutions are in the Tabu list, a new set of neighbors is generated.

Rather than generating a fixed number of neighbors, the algorithm determines the size of the neighborhood based on the size of the band selected for mutation:

$$Neighborhood\ Size = \beta \times length(band);\ //\ \beta \in \{\tfrac{1}{4}, \tfrac{1}{8}, \tfrac{1}{16}\}$$

The factor β is inversely proportional to the number of variables and was introduced as a trade-off to speed up computation by reducing the neighborhood size as the number of variables increases. For the sake of reducing memory usage and increasing speed of comparison, we chose to store the *checksum* of the input vector rather than saving the entire vector in the list.

I. Experimental Results

For the purposes of this paper we limited our experiment to the completely unstructured **urf4** (11 variables) benchmark function [19] and compared the performance of three methods of *selecting input vector sequences* using the Covered Set Partition algorithm for generating valid input vectors. In order to keep a balanced comparison, the following steps were observed:

1. The same synthesis algorithm was used,
2. All algorithms processed the same number of input sequences (30,000 sequences), nerveless the chance, that the same sequence could have been selected repeatedly,
3. The comparison was performed for different partition and results for each partition size are reported separately.

TABLE 2
COMPARISON BETWEEN RANDOM, GA AND TABU SEARCH FOR URF4 REVERSIBLE FUNCTION.

Partition Size	Random	Genetic Algorithm		Tabu search
		Single	Double	
4	3204824	3127213	3074025	3245133
5	3198885	3037825	3105020	3102076
6	3189991	3129114	3121246	2994857
7	3178404	3135759	3058571	3095698

Table 2 shows random selection of input vector sequences consistently resulted in higher quantum cost of the synthesized circuit. The genetic algorithm and Tabu search on the other hand were able to *discover* input vector sequences which produced lower quantum cost. Different recombination and mutation probability thresholds were used to experiment with the genetic algorithm where a mutation probability around 0.2 and recombination probability of 0.8

[2] Tabu or taboo.

produced the best results up to 9.5% improvement over random selection. Although the results do not exhibit any conclusive verdict regarding the CSP partition depth, the random selection and Tabu search both give slightly better performance for higher depth of CSP partition.

TABLE 3
RESULTS OF TABU CALIBRATION WHEN CSP PARTITION = 6

τ \ β	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$
30	3143504	3122341	3118063
40	**2994857**	3120547	3090183
50	3063227	3082519	3173990
60	3150021	3095002	3149779

Table 3 shows the results for various values of the factors β and τ with the best results underlined for different values of β. Clearly the Tabu search performed well compared to both the GA and random search. For a partitions size of 6, the Tabu search yielded the best overall results for a β factor of ¼ and τ of 40.

II. CONCLUSION AND FUTURE WORK

By utilizing heuristic based selection of future sequences based on the quality of already visited solutions, both the genetic algorithm and Tabu search methods were able to discover input vector sequences which produce superior results compared to purely random selection (~ 9.5%). In our experiments, we limited our search to 30,000 sequences for the sake of time that it takes to perform the synthesis (around 3 minutes); however, for an eleven variable function, there exists around an astounding $1.6 \times 10^{5,894}$ (2^{11}!) possible input vector sequence which is impossible to explore fully. Obviously, despite our effort to infuse a hint of intelligence in our selection criteria, we barely visited few small islands in this massive search space.

In our future research, we plan on study the impact of using the genetic algorithm and Tabu search described herein on some of the structured benchmarks such as the *n-th prime* and *hidden weighted bit* functions [19]. We also plan on implementing the algorithm on expanding the number of sequences visited by utilizing GPU hardware acceleration through CUDA and studying the impact of uniform cross over, elitism, and k-parent crossover.

III. BIBLIOGRAPHY

1. Landauer, R., Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development* **5**, 183-191 (1961).
2. Bennett, C., Logical reversiblity of computation. *IBM Journal of Research and Development*, 525-532 (1973).
3. Nielsen, M. & Chuang, I., *Quantum Computation and QUantum Information* (Cambridge University Press, 2009).
4. Caccaro, S. A., Draper, T. G., Kutin, S. A. & Moution, D. P., A new quantum ripple-carry addition circuit. *Quantum Physics* (2004).
5. Perkowski, M., Lukac, M., Shah, D. & Kameyama, M., Synthesis of quantum circuits in Linear Nearest Neighbor model using Positive Davio Lattices. *FACTA UNIVERSITATIS (NIS)* **24** (1), 73-89 (2011).
6. Miller, D. M., Maslov, D. & Dueck, G. W., *A Transformation Based Algorithm for Reversible Logic Synthesis*, presented at Design Automation Conference (DAC), Anaheim, June 2003 (unpublished).
7. Agrawal, A. & Jha, N. K., *Synthesis of Reversible Logic*, presented at DATE, Paris, France, 2004 (unpublished).
8. Donald, J. & Jha, N. K., Reversible Logic Synthesis with Fredkin and Peres Gates. *ACM Journal on Emerging Tecnolgies in Computing Systems* **4** (1) (2008).
9. Dueck, G. W. & Maslov, D., *Reversible Function Synthesis with Minimum Garbage Output*, presented at 6th International Symposium on Representations and Methodology of Future Computing Technologies (RM), Trier, Germany, 2003 (unpublished).
10. Gupta, P., Agrawal, A. & Jha, N. K., *An Algorithm for Synthesis of Reversible Logic Circuits*, presented at IEEE Transaction on CAD, 2006 (unpublished).
11. Alhagi, N., Hawash, M. & Perkowski, M., *Synthesis of Reversible Circuits with No Ancilla Bits for Large Reversible Functions*, presented at International Symposium on Multiple-Valued Logic, Barcelona, Spain, 2010 (unpublished).
12. Hawash, M., Perkowski, M., Bleiler, S., Caughman, J. & Hawash, A., *Reversible Function Synthesis of Large Reversible Functions With No Ancillary Bits Using Covering Set Partitions*, presented at 8th International Conference on Information Technology- New Generation, Las Vegas, NV, 2011 (unpublished).
13. Stedman, C., Yen, B. & Perkowski, M., *Synthesis of Reversible Circuits with Small Ancilla Bits for Large Irreversible Incompletely Specified Multi-Output Boolean Functions*, presented at 14th International Workshop on Post-Binary ULSI Systems, Calgary, Canada, 2005 (unpublished).
14. A Barenco, e. a., Elementary gates for quantum computation. *Physical Review A* **52**, 52:3457–3467 (1995).
15. Maslov, D., Young, C., Miller, D. M. & Dueck, G. W., *Quantum circuit simplification using templates*, presented at Design, Automation and Test in Europe (DATE), Europe, 2005 (unpublished).
16. Glover, F., Future paths for integer programming and links to artifical intelligence. *Computers and Operations Research* **13** (5), 533-549 (1986).
17. Glover, F. & Laguna, M., *Tabu Search, in modern heuristics techniques for combinatorial problems* (John Wiley & Sons, Inc., 1993).
18. Abdalhaq, B., P.h.D. Dessertation, Universitat Autònoma de Barcelona Report No. ISBN: 8468877816, A Methodology to Enhance the Prediction of Forest Fire Propagation.
19. Saeedi, M., Unstructured Reversible Function 4 (urf4), Available at http://www.informatik.uni-bremen.de/rev_lib/function_details.php?id=89.