Appl Intell (2014) 40:1–12
DOI 10.1007/s10489-013-0441-9

Cell assignment in hybrid CMOS/nanodevices architecture using Tabu Search

Sadiq M. Sait · Abdalrahman M. Arafeh

Published online: 31 May 2013

Abstract A recent advancement in VLSI that drastically improved the circuit density is the introduction of CMOL (CMOS/nanodevices hybrid), which consists of an overlay of a nanofabric over a CMOS stack. Combinational logic in CMOL is implemented from a netlist of NOR gates and Inverters by programming nanodevices placed between overlapping nanowires. The length of the nanowires is restricted, and therefore connectivity of the circuit elements is constrained to be within a certain radius, else additional buffers are required.

In this paper we present a Tabu Search (TS) algorithm to address the assignment problem in CMOL. The heuristic is engineered to provide sub-optimal solution by efficient exploration of search space. Empirical results for ISCAS benchmarks are compared with previous solutions using GA, MA, and LRMA heuristics. Results show that in almost all cases, TS exhibits more intelligent search of the solutions subspace, and is able to find better solutions in less time. For all tested benchmarks, over 90 % reduction in average CPU processing time when compared with best published techniques was obtained.

Keywords CMOL · Tabu Search · Combinatorial optimization · Search heuristics · Nanofabric · Assignment · VLSI

S.M. Sait (✉)
Department of Computer Engineering and Center for Communications and IT Research, Research Institute, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia
e-mail: sadiq@kfupm.edu.sa

A.M. Arafeh
Department of Computer Engineering, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia
e-mail: arafeh@kfupm.edu.sa

1 Introduction

Modern iterative non-deterministic heuristics have been increasingly applied to solve a variety of combinatorial optimization problems which are NP-hard. Assigning cells to slots is an important step in the process of electronic design automation. Over time, the objective of placement in VLSI design has changed from reducing the overall wire length to reducing the layout area, to improving timing performance, and then to reducing the overall power dissipation. With new advances in technologies come new issues. The new advancement of CMOS/nanodevice hybrid circuits, like CMOL [26], requires combinational logic to be implemented from a netlist of NOR gates and Inverters by programming nanodevices placed between overlapping nanowires. Like in the case of most programmable devices, the length of the nanowires is restricted, and therefore connectivity of the circuit elements is constrained to be within a certain radius, else additional buffers are required.

The aim of this work is to come up with an efficient and effective method for the cell assignment problem that appears in CMOL nanofabric crossbar architecture. We address the complexity associated with the confined CMOL nanowires crossbar on the logic connectivity and circuits implementation. An iterative heuristic, namely Tabu Search, will be applied to find feasible circuits implementations in CMOL technology, by minimizing the number of additional inserted buffers that may be required. The rest of the paper is arranged as follows: in Section 2, we discuss some backgrounds and previous works. Section 3 introduces CMOL FPGA-like architecture. Section 4 details the problem formulation, Section 5 outlines Tabu Search, a heuristic engineered to solve our combinatorial optimization problem. Section 6 contains the empirical results, comparison and further discussion about the problem behavior. Finally, we conclude the paper and provide final remarks.

2 Literature review

Assignment or cell placement problem has been proven to be NP-hard [22]. For relatively large instances of such problems (i.e., thousands of cells), it is not possible to use enumerative techniques; therefore we resort to heuristic techniques which, in reasonable execution time, can lead to acceptable solutions. Heuristic algorithms assigned to the assignment problem can be broadly classified into constructive and iterative algorithms. The constructive heuristic starts from a seed component; a cell to be assigned to a slot in the layout surface divided into n slots. During each step of the algorithm, one cell will be placed into one of the empty slots. At the end of each step, we have a partial placement of a subset of cells/modules. Different techniques can be applied to choose which unplaced cell must be selected and where to be added to the partial placement. For example, a cell to be placed may be selected depending on how strongly it is connected to the current partial placement. The ideal location of the selected cell can be found, for instance, using a heuristic known as force-directed placement algorithm [24], where the cell zero-force location is mathematically computed. The constructive placement algorithms are greedy and don't produce optimum solutions because at each step they make decisions in the absence of complete information. For example, when the cell is selected during the ith step, the selection is made with respect to the partial placement; the unplaced modules are ignored. Once a cell is placed, the algorithm will not go back and retrack from its already made decision. In CMOL circuits, connections between cells are done through already available nanowires and nanodevices. Constructive heuristics may reach the point where design constraints (e.g., limited connectivity radius in CMOL) are not met, which end up with not only costly (e.g., wirelength) but infeasible solutions.

On the other hand, iterative heuristics can be engineered to modify a given initial placement while respecting problem constraints and improving the given cost function. Iterative improvement procedures constitute very effective approaches to produce feasible solutions with the desired performance. Examples of iterative heuristics, which are also known as non-deterministic heuristics, have been widely used in optimization problems and not limited to only VLSI design automation. These include Genetic Algorithms [9, 15, 20], Simulated Evolution [21], Stochastic Evolution [23], Particle Swarm Optimization [14, 30], Ant Colony [16, 31], and Tabu Search [12].

Tabu Search has been applied to solve a large number of combinatorial optimization problems in various fields of sciences, engineering, and business. Results reported indicate comparable performance to other iterative heuristics. Examples of hard problems to which Tabu Search has been applied with success include routing and networking [7], scheduling [34], and space allocation [18]. Previous implementations of Tabu Search for VLSI design problems include parallel multi-objective cells placement [19], where power, area, and delay objectives are combined in an aggregated fuzzy cost function. Parallelization strategy is adapted to traverse the large search space looking for placement with minimum cost.

Feature size scaling in CMOS technology has led to difficulties in manufacturing, due to short channel effects, doping fluctuations and expensive lithography process. Meanwhile, advances in nanoelectronics are expected to achieve high density of devices that can operate at THz frequencies [4]. Many effective applications have been proposed that use molecular nanodevices, nanowires, and nanocrossbar fabrics [25, 26]. A new trend is emerging for combining the flexibility and high fabrication yield advantages of CMOS technology with nanometer-scale molecular devices. A self-assembly of two-terminal nanodevices, with nanowire crossbar fabrics, enables high functional density and sustains acceptable fabrication costs. Likharev and Strukov [26] introduced a hybrid semiconductor/nanowire/molecular integrated circuit called CMOL, which uses two levels of perpendicular nanowires as crossbar interconnection on top of inverter-based CMOS stack, and showed possible applications of CMOL in field programmable gate arrays (FPGA) [29], neuromorphic CrossNets [17], and in memories [27].

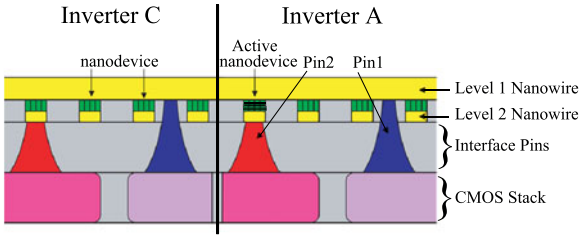
Recently, several proposals had been introduced for cell placement/assignment on FPGA-like CMOL architecture. Like in other nano-fabric crossbars and FPGA like devices, nanowires break at fixed intervals confining CMOL cell connectivity to a fixed number (M) of other cells located within its proximity square-like connectivity domain. Each CMOL cell must be connected to one of its proximity cell members, and failure to do so will require the insertion of a buffer which results in increase of congestion and delay. The problem here is to find an assignment that will result in smallest number of additional buffers.

Likharev et al. utilized existing FPGA CAD tools to perform placement and routing on 4×4 tile-based version of CMOL [28, 29]. They used reserved routing cells and recursive routing algorithm for inter-tile routing. Hossein et al. [11] proposed a recursive method for removing routing congestion by keeping and ranking placement solutions in final iterations of the placement algorithm according to cost. Subsequently, when routing of best placement configuration failed, another placement solution was considered until routing was satisfied. Instead of working at tiles level, Hung et al. [13] encoded the CMOL cell assignment as a Satisfiability problem at cells level, where a placement solutions is found when all Boolean constraints are satisfied. However, when circuits sizes increased the computation time became evident.

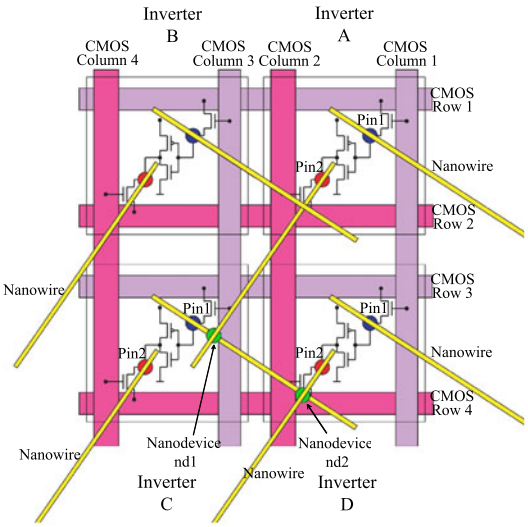
Previous attempts to use sub-optimal search heuristics are reported in [6, 32, 33]. The genetic Algorithm (GA) [32] was used with two dimensional block PMX crossover operator and mutation, where the fitness function evaluated the Manhattan distance between connected cells. The heuristic was driven to minimize the total Manhattan distance of the population. Nonetheless, memory requirements, choices of data structure for chromosomes representation, and computation time are significant disadvantages of GA. A more elaborate work was reported in [6]; where a Memetic computing approach was used by implementing a hybrid of the Genetic Algorithm and the Simulated Annealing (SA) local-based search heuristic. SA was used in each generation to enhance offsprings which resulted from PMX crossovers and pairwise interchange mutations in GA. Hung et al. [33] extended their work on Memetic approach by integrating self-learning operators using Lagrangian Multipliers (LRMA). The Lagrangian relaxation technique (LRT) was applied in population goodness function by assigning Lagrangian multipliers to penalty values corresponding to problem constraints and repeatedly updating them. Results reported using LRMA approach is promising, however, more computations are needed for the penalty updating mechanism and SA local-based search. A theoretical investigation on CMOL cell assignment is reported in [5]; the authors proved mathematically that placement of 2-input NOR/INV circuits is possible and may require adding additional buffers (i.e., pair of inverters in case of cells that require long wires to connect) to satisfy all connections.

3 CMOL FPGA architecture

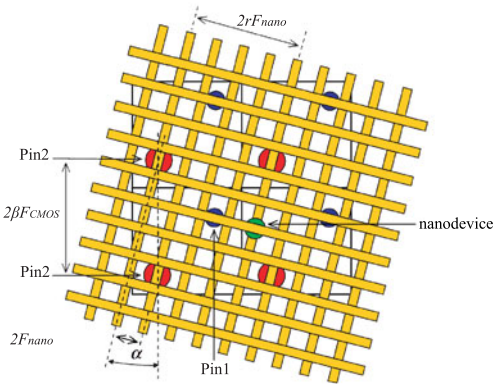
CMOL cell-based, field-programmable gate array (FPGA)-like architecture is based on integrating conventional inverter-based four-transistor MOSFET CMOS cell with uniform reconfigurable nanowire fabric. Two-terminal nanodevices "latching switches", that have two metastable internal states, are self-assembled at each crosspoint in CMOL fabric and provide diode-like I–V curves for logic circuits implementation. Likharev et al. predicted the density of nanodevices to be above 10^{12} to cm^2 for $F_{nano} = 3$ nm, where F_{nano} is the nanowires half-pitch. That results in abundant available nanodevices that can serve both inter-cells connectivity and wiring-logic. CMOS stack is connected to nanofabric by Metal pins that span to top and bottom nanowire levels as shown in Fig. 1(a). Two CMOS inverters (i.e., inverter A and inverter C) are connected by pin-nanowire-nanodevice-nanowire-pin connection. The electrical representation of four inverter-based CMOS cells and corresponding nanowire and nanodevices is shown in Fig. 1(b). Inverter A has two pins; pin1 connects the input of the CMOS inverter to one of the nanowires levels making the nanofabric, while pin2 connects the CMOS inverter's output to the second level of nanowires. The upper right cell (inverter A) is connected to the lower left cell (inverter C) by activating the appropriate nanodevice (nd1) in the crosspoint



(a) Schematic side view of two CMOL cells with two levels of nanowires. Only one nanodevice is activated to connect the output of Inverter A with input of Inverter C.



(b) Electrical representation of four CMOL cells and corresponding nanowires.



(c) Nano-fabric inclined by α on top of four CMOS cells, only one nanodevice is shown.

Fig. 1 Low-level structure of CMOL circuit: the incline angle $\alpha \ll 1$ and dimensionless parameter β satisfy two conditions, $\sin\alpha = F_{nano}/\beta F_{CMOS}$ and $\cos\alpha = rF_{nano}/\beta F_{CMOS}$ where r is an integer

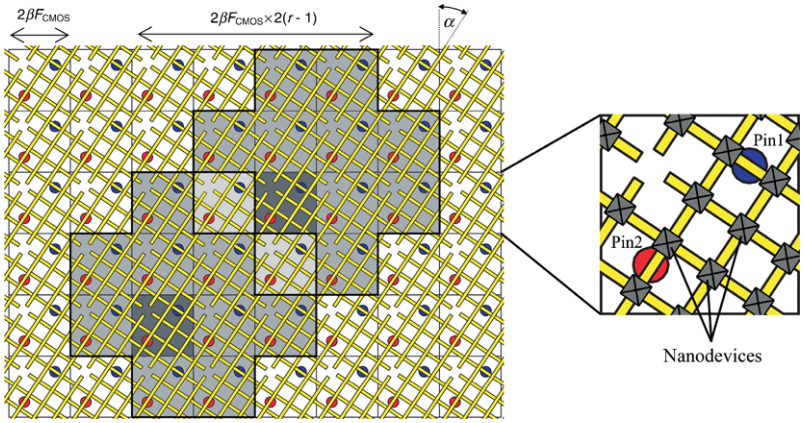


Fig. 2 CMOS FPGA topology: for $r = 3$, $M = 2r(r-1) - 1 = 11$ cells in the "Connectivity Domain" (Highlighted by *dark line*) for the input pin of cells painted in *dark-grey* [28]. The overlap between connectivity domain of two cells is shown in *light grey*

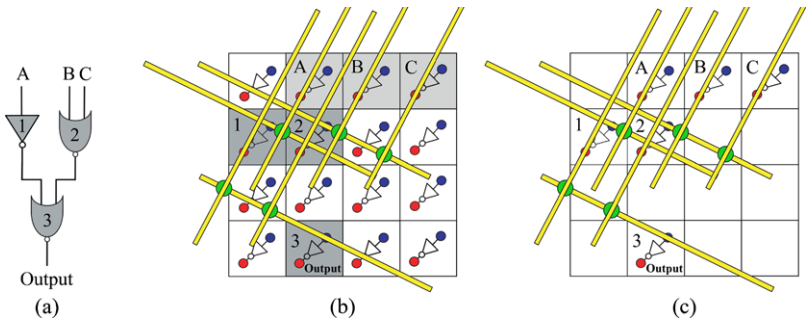


Fig. 3 Example of CMOL circuit: (**a**) NOR/INV logical circuit; (**b**) CMOL implementation of (**a**), (**c**) showing only used cells. *Shaded cells* are connected through combination of nanowires, nanodevices and CMOS pins

between the nanowire connected to output of inverter A and nanowire connected to input of inverter C. When two or more nanodevice on the same nanowire are activated as shown in Fig. 1(b) (nd1 and nd2) the output of inverter C will be equivalent to NOR gate whose inputs are cell A and cell B. Wired-OR logic is implemented through nanowires and nanodevice.

CMOL nanowire crossbar is rotated by angle $\alpha = \arcsin(F_{nano}/\beta F_{CMOS})$ related to the CMOS pins that are arranged into a square array with side of $2\beta F_{CMOS}$ as shown in Fig. 1(c), where F_{CMOS} is CMOS half-pitch, and β is a factor larger than 1. This approach allows a unique access to any nanodevice via the appropriate pin pair. Each CMOS cell has an area of $A = (2\beta F_{CMOS})^2$. Like other nano-fabric crossbars, CMOL's nanowires break at repeated intervals of $L = 2\beta^2 F_{CMOS}^2$ confining CMOL cells connectivity to only $M = 2r(r-1) - 1$ other cells located within its proximity square-like "Connectivity Domain" as shown in Fig. 2, where r is an integer value that indicates the connectivity domain diameter and represents the constraint of CMOL placement. CMOL nanodevices can be configured by setting appropriate voltages. When configuration is done the nanodevices are set to the ON (low-resistance) state or OFF (high-resistance). If the nanowires and nanodevices shown in Fig. 3(b) are activated, the CMOL circuit will be equivalent to circuit shown in Fig. 3(a). The first NOR gate of the circuit can be implemented by connecting inputs 'A' and 'B' with inverter '1' to satisfy both connectivity and logic wiring for the desired gate. The abundance of available nanodevices and nanowires provides a variety of different possible configurations for the implementation of one circuitry. Among those their could be only certain configurations that satisfy connectivity domain constraint and do not require additional routing resources.

Different variations of CMOL cells architecture were developed in the literature; initially, Likharev [28] extended cell types to include latches. Later, Dong et al. [8] proposed two new CMOL cells for efficient sequential logic implementation, the T-Cell, basically a transmission gate and the D-Cell, a transmission gate and an inverter. Those new cells can be combined with original inverter-based cells to form Tri-State buffers for MUX implementation and D Flip-Flops for sequential design. Abid et al. [1] utilized two types of nano-junction devices with CMOL-based cells to implement cryptographic algorithms. They developed XOR gates with resistive junctions and XOR/AND gates with diode-like junctions. The proposed design combines CMOS inverters with transmission gates, and results in sufficiently larger cells than the conventional inverter cell of CMOL. Abid et al. [2] also introduced a 3D CMOL FPGA implementation where a nano-wire crossbar is placed between two CMOS layers, each layer reaches to nanowires spanning in one direction. This arrangement provides improved density, but with complex inter-cell connectivity as each cell is only restricted to access one level of the nanowire crossbar.

4 Problem formulation

The placement or assignment of cells in order to minimize a cost function is a NP-hard problem [23]. Even one dimensional placement, the simplest possible, is hard to solve. In 2-D array of n locations there are as many as

$$S = n(n-1)(n-2)\cdots(n-m) \tag{1}$$

arrangements for placing m cells, where m could be in the thousands. Overtime, heuristic techniques have been developed for solving the placement problem, and finding a good solution in polynomial function of m.

Given a collection of NOR/INV gates, and the collection of nets (the set of ports to be connected together), the CMOL placement problem consists of finding suitable locations for each gate under the constraint of connectivity domain and are given a cost function. Formally the problem can be restated as: for a set of gates $G = g_1, g_2, g_3, \ldots, g_m$ and a set of netlists $\Gamma = \gamma_1, \gamma_2, \gamma_3, \ldots, \gamma_m$ where $\gamma_i = \{fan\text{-}in_i \ \& \ fan\text{-}out_i\}$ of g_i and given a set of slots or locations $L = L_1, L_2, L_3, \ldots, L_n$ where $m \leq n$, the placement problem is to assign each $g_i \in G$ to a unique location L_j such that the objective is optimized. Positions are defined by the coordinate values (x_j, y_j) and the subset of G that represent inputs/outputs may be pre-assigned fixed locations or constrained to certain positions.

Each CMOL cell can implement one inverter or one NOR gate with multiple fan-in, however, complying to the connectivity constraint can be substantially harder if gates of high fan-in are allowed. Unlike conventional CMOS-based cell assignment, CMOL cell placement is constrained to "*Connectivity Domain*" of radius r. Each CMOL cell is connectable to one of its proximity cell members, any violation of this constraint would impose further processing (i.e., buffer insertion) to satisfy connectivity. However, such a process would cause more congestion to the already congested CMOL circuit and could result in a substantial increase of timing delay. Mathematically, the "Connectivity Domain" can be defined as follow. Given a gate and its netlist $(g_i, \gamma i)$ placed in location L_i, for any gate $g_k \subseteq G$ and g_k in the netlist γi the following inequality should be satisfied.

$$dist(L_j, L_k) \leq r \tag{2}$$

where L_k is the location of g_k, *dist* is Manhattan distance, and r is CMOL connectivity diameter. The objective of CMOL cell assignment is to satisfy the constraint in Inequality (2), and to minimize distance between connected gates in circuit G. Failing to comply with the CMOL constraint will result in an implementation that has more delay and area requirements. The complexity of CMOL placement arises from the overlap in connectivity domain of adjacent cells as shown in Fig. 2, that results in fewer connectivity choices.

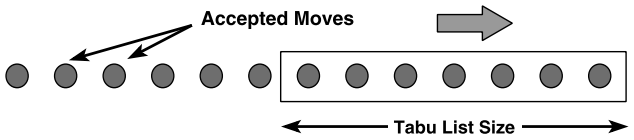


Fig. 4 Tabu list visualized as window over accepted moves

5 Tabu Search and its implementation

Tabu Search is a general iterative metaheuristic for solving combinatorial optimization problems. TS proceeds by making iterative perturbations while preventing cycling to certain number of recently visited points in search space. The TS procedure starts from an initial feasible solution S (current solution) in the search space Ω. A neighborhood $\aleph(S)$ is defined for each S. A sample of neighbor solutions $\mathbf{V}^* \subset \aleph(S)$ is generated called *trial* solutions ($n = |\mathbf{V}^*| \ll |\aleph(S)|$), and comprises what is known as the candidate list. From this generated set of trial solutions, the best solution, say $S^* \in \mathbf{V}^*$ is chosen for consideration as the next solution. A solution $S^* \in \aleph(S)$ can be reached from S by an operation called a move to S^*. The move to S^* is considered even if S^* is worse than S, that is, $Cost(S^*) > Cost(S)$. Selecting the best move in $\mathbf{V}^*$ is based on the supposition that good moves are more likely to reach the optimal or near-optimal solutions. The best candidate solution $S^* \in \mathbf{V}^*$ *may* or *may not* improve the current solution, but is still considered. It is this feature that enables *escaping* from local optima. However, with this strategy, it is possible to reach the local optimum, since moves with $Cost(S^*) > Cost(S)$ are accepted, and then in a later iteration return back to local optimum.

In order to prevent returning to previously visited solutions a memory or list $\mathbf{T}$, known as *tabu list*, is maintained. This list contains information that to some extent forbids the search from returning to a previously visited solution. Whenever a move is accepted, its attributes are introduced into the tabu list $\mathbf{T}$. Move reversal is prevented for the next $k = |\mathbf{T}|$ iterations because they *might* lead back to a previously visited solution. The tabu list can be visualized as a window on accepted moves as shown in Fig. 4. The moves which tend to undo previous moves within this window are forbidden.

In some cases, it is necessary to overrule the tabu status since only move attributes (not complete solutions) are stored in tabu lists. These tabu moves may also prevent the consideration of some solutions which were not visited earlier. This is done with the help of the notion of *aspiration criterion*. Aspiration criterion is a device used to override the tabu status of moves whenever appropriate. It temporarily overrides the tabu status if the move is sufficiently good. Aspiration criterion must make sure that the reverse of a recently made move leads the search to an unvisited solution, generally a better one. A flow chart illustrating the basic

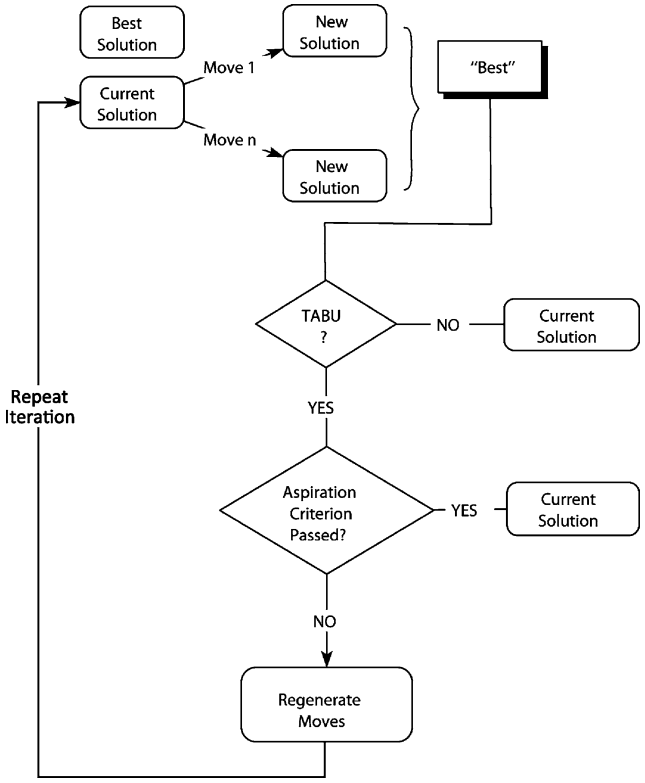


Fig. 5 Flow-chart of Tabu Search algorithm

Ω: Set of feasible solutions (i.e., placements).
S: Current solution.
S^*: Best admissible solution.
Cost: Objective function (Reduce # of buffers).
$\aleph(S)$: Neighborhood of $S \in \Omega$.
$\mathbf{V}^*$: Sample of neighborhood solutions.
T: Tabu list.
AL: Aspiration Level.

Begin
1. Start with an initial feasible solution (placement) $S \in \Omega$.
2. Initialize tabu lists and aspiration level.
3. **For** fixed number of iterations **Do**
4. Generate neighbor solutions $\mathbf{V}^* \subset \aleph(S)$.
(Each solution results from the swap of two cells).
5. Find best $S^* \in \mathbf{V}^*$.
6. **If** move S to S^* is not in **T Then**
7. Accept move & update best solution.
8. Update tabu list (Store swap reversal).
9. Update aspiration level.
(**AL** = Cost of best solution seen so far).
10. Increment iteration number.
11. **Else**
12. **If** $Cost(S^*) <$ **AL Then**
13. Accept move & update best solution.
14. Update tabu list & aspiration level.
15. Increment iteration number.
16. **EndIf**
17. **EndIf**
18. **EndFor**
End.

Fig. 6 Algorithmic description of short-term Tabu Search (TS)

short-term memory Tabu Search algorithm is given in Fig. 5. Intermediate-term and long term memory processes are used to intensify and diversify the search respectively [10, 23].

One of the Tabu Search algorithm parameters is the size of the tabu list. A small tabu list size is preferred for exploring the solution near a local optimum, and a larger tabu list size is preferable for breaking free of the vicinity of local minimum. The list size varying between 5 and 12 have been used in many applications. Any aspect (feature or component of a solution) that changes as a result of a move from S to S^{trial} can be an attribute of that move, where a single move can have several attributes. The duration for which a move containing the particular tabu attribute is forbidden (the size of tabu list) is called Tabu tenure. An algorithmic description of a simple implementation of the tabu search is given in Fig. 6.

5.1 Solution representation and initialization

A placement solution is an arrangement of logic cells in a two dimensional layout surface. The representation used in this work is in the form of a 2-D grid. The layout is constructed by computing the number of required CMOL cells to fit each benchmark circuit. The outer cells of the grid are reserved for I/O pins, where I/O pins moves are restricted to these reserved locations. In the initialization phase each logic gate is assigned a positive integer value that distinguishes it from the rest. Then, the encoded logic gates are randomly assigned in the 2-D layout as shown in Fig. 7.

Fig. 7 2-D grid layout of CMOL initial placement of *s27.blif*. 19 cells; 8 gates, 7 inputs and 4 outputs

4	2	3		0
6		17	7	
	14	18	16	
	8	9	15	13
10	5	12	1	11

5.2 Cost evaluation

The main objective of placement is to find a feasible assignment of cells in which all connections are satisfied. One way to accomplish this is to place strongly connected cells close to each other. A commonly used objective function is the total weighted wirelength over all signal nets and is expressed as:

$$L(P) = \sum_{n \in N} w_n \cdot d_n \tag{3}$$

where, d_n is the estimated wirelength of net n and w_n is weight of net n. Since, in CMOL all cells are connected via

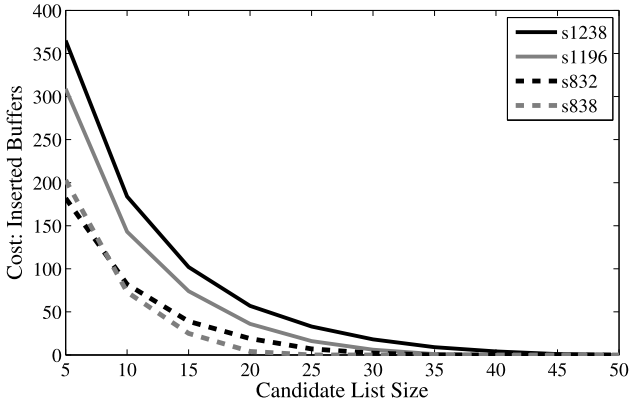


Fig. 8 Final cost yielded by TS in four circuits vs. candidate list size ($r = 12$)

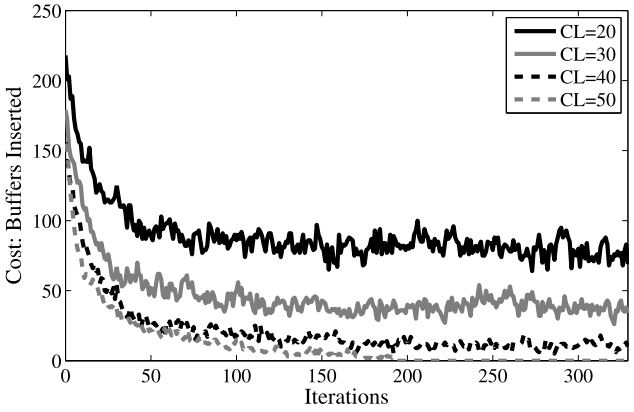


Fig. 9 Change in cost per iteration of s1238.blif for different candidate list sizes ($r = 12$)

pre-assembled nanowires, the problem we are trying to optimize is to place connected cells within each others connectivity domain as to avoid insertions of additional buffers. Therefore, we should have a measure which can quantify the overall quality of the solution. A conventional approach is to calculate the number of nets that violate the connectivity domain constraint. The overall cost of a solution is the total number of connectivity domain violating nets (the number of additional buffers that are needed to satisfy all connections). The cost of each gate $g \in G$ is expressed in Eqs. (4a) and (4b), where the overall circuit's cost is the sum of the individual cost of the gates.

$$C_i = \sum_{j \in \gamma(i)} u_{i,j} \tag{4a}$$

$$u_{i,j} = \begin{cases} 1 & \text{if } dist_{i,j} > r \\ 0 & \text{otherwise} \end{cases} \tag{4b}$$

5.3 Neighborhood solutions generation

In each iteration we generate a number of neighbor solutions (i.e., candidate list) by making perturbations as follows: two cells (two I/O pins or two logic cells) are selected randomly, then their locations are interchanged. Each solution in the candidate list is evaluated based on the change in number of buffers before and after the swap. If two or more neighborhood solutions have equal swap cost, which also happens to be the best cost in the candidate list, the solution with lesser Manhattan distance is chosen. We have experimented with different sizes of candidate lists; Fig. 8 shows the final cost yielded by TS in four benchmark circuits when candidate list size is changed, given that all other parameters are constant. It is clearly seen that for this problem TS had better results when more neighbor solutions are considered. Figure 9 shows the per iteration cost of one circuitry given different sizes of candidate list. Candidate list sizes in the range of 50 reach the optimal solution of zero buffers when $r = 12$ in less iterations, thus this size has been used throughout our implementation.

5.4 Tabu list and aspiration level

Different tabu attributes were tested, when two cells i and j are swapped. One attribute was to forbid moves related to cell i, that means that any move which included i, even swapping i with j, was tabued. Another experiment considered both i and j, forbidding any perturbations that include either of them. The Tabu attribute of a move that is used in all results reported in this paper is swap reversal. If two cells are involved in interchange, the reversal of this move is forbidden. A short-term memory element is used throughout the implementation where experiments of tabu list size ranging from 5 to 12 were conducted. We conclude that change in the tabu list size in this range has little impact on the quality of the solutions, thus the size of tabu list is taken as a fixed value equal to 5. The aspiration criterion is based on the following: if the current solution is the best seen so far (i.e., better than the global best solution), then tabu restriction is overridden and the current solution is accepted as new best solution and tabu list is updated.

5.5 Tailored neighborhood solutions

Since the number of cells that violate the given constraint are far less than the total number of cells, it is rational for Tabu swaps to always include cell(s) that violate the connectivity radius constraint. Therefore, we experimented by modifying the neighborhood solutions generation by constructing a list of top 20 cells which have many long connections and exceed the connectivity radius. In each iteration, this list is updated to keep track of the worst placed cells. Each Tabu move (i.e., swap) involves a cell from this list, and another cell that may or may not be in the list. This modification results in always including violating cells in Tabu moves, such

moves may take the search to unvisited solutions thereby enhancing search space exploration.

To avoid being greedy, and not curtail exploration, the following approach was adopted. The list of all of the circuit's cells are sorted according to the cost given in Eq. (4a). When selecting cells for swaps, one cell is selected randomly while the other is selected probabilistically. Those on the top of the list which have more violations have a higher chance of being selected, while those pairs that do not have violating cells also have a non-zero probability of being selected. To perform the probabilistic selection, we used the positive values of a Gaussian random variable which has mean $m_0 = 0$ and standard deviation $3\sigma = circuit - size$. Given the Gaussian distribution, cells in the top of the list will be more frequently selected than those in the bottom of the list.

5.6 Buffers insertion

Buffers insertion is a post-processing procedure which is performed when Tabu Search terminates. Violating connections are resolved by inserting buffers as intermediary cells. Buffers insertion may fail if CMOL grid is highly congested and/or the circuit still has many violating connections. Initially, the procedure starts with one buffer (pair of inverters) for each violating connection. Buffers are inserted using the same iterative methods used for cell placement. Tabu Search is recalled given a number of modifications. All blank cells in the grid are considered for buffers insertion. The inverters are randomly assigned. Swaps are allowed only between added inverters or an inverter and blank cell. The algorithm continuously improves the locations of the inverters. If the algorithm terminates and still some connections violate the connectivity domain constraint, additional buffers are added for those connections and the procedure is repeated. This process repeats until all connected gates are within each others connectivity domain, or when no blank cells are left in CMOL grid (i.e., the circuit is infeasible given the current cells arrangement). In our implementation, we limited the number of inverters that can be inserted in a given connection to 6 to avoid major deterioration of the circuit's timing delay.

6 Experimental results

Evaluation of search heuristic efficiency and behavior is conducted using ISCAS'89 benchmarks [3]. Further consideration should be given to ISCAS'89 by replacing sequential elements' inputs and outputs with POs and PIs respectively [13]. ISCAS'89 benchmarks used in this work are mapped to NOR-based gates with maximum of five inputs. Tabu Search has been implemented using Java programming language and executed on a machine comparable to

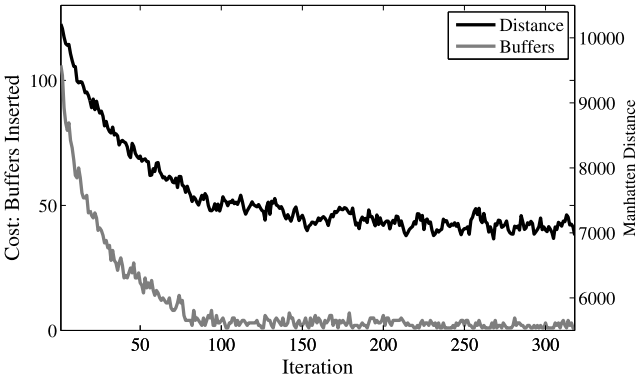


Fig. 10 Change of problem cost and Manhattan distance in TS iterations ($r = 12$—s1238.blif)

the one used by other simulations published in literature, it has 1.5 GHz Intel Pentium M processor with 512 MB memory. Three options were available to find a placement solution that satisfy the CMOL connectivity constraint; one was to minimize Manhattan distance, another was to minimize number of inserted buffers by using cost function discussed in Sect. 5.2, and the third was to minimize both distance and buffers. Results obtained for the first option showed that the number of inserted buffers are more than those of the second option. Moreover, when minimizing buffers, Manhattan distance was reduced to similar levels as if distance was being optimized. Minimizing both distance and buffers required more processing and didn't have significant advantages over minimizing buffers only. Figure 10 shows the correlation between the number of inserted buffers and Manhattan distance. It can be seen that TS is accepting bad moves to reach better solutions in terms of inserted buffers (which is the main objective), and in this process, the Manhattan distance (wirelength) also improves.

Table 1 shows the number of cells (i.e., NOR/INV logic gates), inputs and outputs of benchmark circuits used; Area (*Tiles*) is the area used by CMOL FPGA CAD 1.0 tool [29], while Area (*Row* × *Column*) is the area used in GA [32], MA [6], LRMA [33] and TS. The heuristic stops when all violations are removed or when reaching a predefined number of iterations. The median value of results obtained from 20 runs for each circuit is reported where each run uses different seeds for random numbers.

6.1 Literature comparison

Comparison is performed with CMOL FPGA CAD 1.0, we set the connectivity radius to $r = 12$ and $r = 9$. GA, MA and LRMA use population size equal to 24 and stopping criterion when fitness score is not updated for 50 times. The crossover rate in MA and LRMA is $RC = 0.33$ and mutation rate $RM = 0.01$. Simulated Annealing used in each of GA iterations has initial temperature $T = 0.2$ and terminating

Table 1 ISCAS'89 Benchmarks: showing the number of *Cells* to be placed including *Gates*, *Inputs* and *Outputs*. *Area* is the size of CMOL 2-D grid. *AU %* is the fraction of utilized cells in CMOL grid

Circuits	Cells	Gates	Inputs	Outputs	Area (*Tiles*)	Area (*Row* × *Column*)	*AU %* (*Tiles*)	*AU %*
s27	19	8	7	4	64 (2 × 2)	25 (5 × 5)	18.75	32.00
s208	136	109	18	9	256 (4 × 4)	169 (13 × 13)	48.05	64.50
s298	122	85	17	20	256 (4 × 4)	144 (12 × 12)	48.83	59.03
s344	180	130	24	26	400 (5 × 5)	196 (14 × 14)	43.50	66.33
s349	184	134	24	26	400(5 × 5)	196(14 × 14)	26.50	68.37
s382	175	124	24	27	400 (5 × 5)	196 (14 × 14)	43.25	63.27
s386	164	138	13	13	400 (5 × 5)	196 (14 × 14)	54.75	70.41
s400	188	137	24	27	400 (5 × 5)	196 (14 × 14)	47.25	69.90
s420	299	248	34	17	400 (5 × 5)	361 (19 × 19)	75.00	68.70
s444	187	136	24	27	400 (5 × 5)	196 (14 × 14)	52.50	69.39
s510	304	266	25	13	–	361 (19 × 19)	–	73.68
s526	273	222	24	27	576 (6 × 6)	324 (18 × 18)	57.12	68.52
s641	302	206	54	42	576 (6 × 6)	676 (26 × 26)	50.17	30.47
s713	321	225	54	42	–	676 (26 × 26)	–	33.28
s820	447	400	23	24	–	529 (23 × 23)	–	75.61
s832	454	407	23	24	–	529 (23 × 23)	–	76.94
s838	606	507	66	33	–	676 (26 × 26)	–	75.00
s1196	675	613	31	31	–	729 (27 × 27)	–	84.09
s1238	724	662	31	31	–	784 (28 × 28)	–	84.44

temperature 0.01. Tables 2 and 3 show the final results obtained for ISCAS'89 benchmarks when $r = 12$ and $r = 9$ respectively; (*Delay*) is the circuit's logical levels reported by SIS tool after inserting the buffers, computation time (*Time*) in seconds, (*Buf*) shows the number of inserted buffers to satisfy CMOL connectivity domain.

Tabu Search solutions are more effective than those of CMOL CAD 1.0 in terms of computation time, delay and area utilization. The last two columns of Table 1 show that cell-based CMOL architecture has better area utilization *AU %* than that of tile-based architecture. Tables 2 and 3 indicate that the tile-based approach is the most time consuming and the least effective in timing delay. It also fails to place big circuits.

Results obtained from implementation of TS for $r = 12$ are better than those obtained in GA, MA and LRMA in both computation time and Buffers count. TS required shorter CPU processing time due to its simplified operations compared to genetic crossover, mutation and Lagrangian multipliers calculation in LRMA. Table 2 shows that Tabu Search found the optimal solutions with zero buffers for all benchmarks, with 92 % average computation time saving. For example, s1238 benchmark needed only 12.87 seconds in TS, comprising only a 3.6 % of time needed by LRMA.

Table 3 shows TS results when $r = 9$; solutions found by TS are better than those of MA for all benchmark circuits. TS falls behind LRMA in only two circuits (s820 and s1238) while sustaining equal averaged results. Again, TS found solutions in lesser time with 73 % saving.

Experiments were conducted using the tailored neighborhood generation; quality of results was similar to those of Tabu search with random swaps, however the range of candidate list size reduced from 30–50 swaps to 20–40 swaps. This constituted for 20–35 % reduction in candidate list size compared to that required when random cells were selected.

For circuits s820, s832, s1196, s1238 when $r = 9$, buffers insertion heuristic was unable to resolve all of the violating connections due to limited CMOL grid size. For example, circuit s1238 requires 54 buffers (i.e., 108 inverters) where only 60 blank cells are available in CMOL grid. Therefore, a bigger CMOL grid should be used to implement the circuit. For accurate comparison with previous algorithms, we used the same grid size as reported in literature. Results given for the aforementioned circuits in Table 3 indicate the number of violating connections rather than the number of buffers inserted.

7 Conclusion

In this paper we presented the implementation of Tabu Search heuristic for CMOL nano-hybrid cells placement. We analyzed the problem behavior and engineered a Tabu Search solution that exploits better understanding of the limitations imposed by CMOL connectivity domain. Further,

Table 2 CMOL CAD, GA, MA, LRMA and TS results comparison for ISCAS'89 benchmark circuits and connectivity radius $r = 12$. *Delay* is circuit's logic levels. *Time* is computation time in seconds. *Buf* is the number of inserted buffers

Circuits	CMOL CAD 1.0		GA [32]			MA [6]			LRMA [33]			Tabu Search		
	Delay	*Time*	*Delay*	*Time*	*Buf*	*Delay*	*Time*	*Buf*	*Delay*	*Time*	*Buf*	*Delay*	*Time*	*Buf*
s27	9	1	7	0.01	0	7	0.01	0	7	0.01	0	7	0.01	0
s208	18	3	16	1.12	0	16	0.12	0	16	0.10	0	16	0.01	0
s298	13	7	11	0.17	0	11	0.11	0	11	0.09	0	11	0.01	0
s344	20	8	18	0.57	0	1	0.29	0	18	0.16	0	18	0.01	0
s349	20	7	18	0.49	0	18	0.28	0	18	0.18	0	18	0.01	0
s382	13	7	11	1.60	0	11	0.38	0	11	0.32	0	11	0.03	0
s386	16	11	10	1.05	0	10	0.33	0	10	0.34	0	10	0.03	0
s400	15	8	11	2.12	1	11	0.40	0	11	0.34	0	11	0.02	0
s420	20	8	16	8.50	1	16	3.41	0	16	1.57	0	16	0.07	0
s444	17	9	11	1.86	2	11	0.40	0	11	0.34	0	11	0.03	0
s510	–	–	18	16.56	2	18	7.56	0	18	3.42	0	18	0.18	0
s526	16	13	11	9.75	5	11	4.36	0	11	1.59	0	11	0.48	0
s641	25	8	23	82.66	15	**19**	**39.40**	**4**	16	22.02	0	**16**	**6.27**	**0**
s713	–	–	24	52.84	34	19	30.11	3	19	**41.77**	**2**	19	**8.69**	**0**
s820	–	–	15	77.52	41	12	61.71	10	12	**54.09**	**6**	12	**11.77**	**0**
s832	–	–	16	69.27	54	12	60.17	11	12	**63.77**	**4**	12	**10.55**	**0**
s838	–	–	28	201.37	50	24	85.62	7	24	**100.40**	**4**	24	**4.48**	**0**
s1196	–	–	30	234.88	84	23	208.15	19	**24**	**179.47**	**9**	**23**	**6.87**	**0**
s1238	–	–	37	268.92	121	28	267.34	31	26	**353.00**	**9**	26	**12.87**	**0**
Average	–	–	17	54.28	22	15	**40.53**	**4**	15	**43.31**	**2**	15	**3.28**	**0**

Table 3 CMOL CAD, MA, LRMA and TS results comparison for ISCAS'89 benchmark circuits and connectivity radius $r = 9$. *Time* is computation time in seconds. *Buf* is the number of inserted buffers

Circuits	CMOL CAD 1.0	MA [6]		LRMA [33]		Tabu Search	
	Time	*Time*	*Buf*	*Time*	*Buf*	*Time*	*Buf*
s27	0.07	0.01	0	0.01	0	0.01	0
s208	509.84	0.22	0	0.20	0	0.01	0
s298	370.3	0.27	0	0.37	0	0.05	0
s344	6.18	0.85	0	0.65	0	0.04	0
s349	7.6	0.57	0	0.72	0	0.04	0
s382	12.88	5.70	0	1.43	0	0.67	0
s386	10.3	1.89	0	1.62	0	0.20	0
s400	7.52	4.48	0	1.82	0	0.61	0
s420	–	13.83	0	7.73	0	1.24	0
s444	7.59	5.74	0	2.05	0	0.97	0
s510	213.27	22.71	7	**25.49**	**5**	**64.57**	**1**
s526	–	21.72	5	**23.13**	**2**	**39.44**	**0**
s641	–	48.26	11	**106.64**	**6**	**51.73**	**1**
s713	–	79.63	12	**97.38**	**3**	**51.88**	**2**
s820	–	202.60	42	**153.20**	**31**	**75.91**	**32**
s832	–	118.83	45	**164.06**	**39**	**77.75**	**37**
s838	–	22.60	15	**189.12**	**10**	**63.13**	**1**
s1196	–	502.22	49	**565.41**	**36**	**72.35**	**35**
s1238	–	404.11	55	**856.69**	**39**	**73.00**	**54**
Average	–	**76.64**	**13**	**115.67**	**9**	**30.19**	**9**

we proposed a probabilistic method to make tailored swaps and reduce candidate list size. Results obtained are better than those published in literature with savings in required computation time. We are investigating the implementation of other search heuristics for CMOL placement problem and are experimenting with placement and reconfiguration around defective nanodevices.

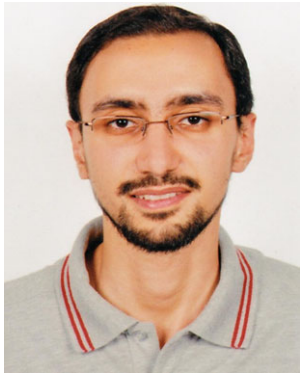
Acknowledgements The authors acknowledge King Fahd University of Petroleum & Minerals for its support, and Dr. William N.N. Hung and Mr. Zhufei Chu for providing ISCAS'89 benchmark files. The authors would like also to thank Dr. Rajat Subhra Chakraborty for his help and support. Thanks are also due to the reviewers of the manuscript whose comments helped in improving our solutions.

References

1. Abid Z, Alma'aitah A, Barua M, Wang W (2009) Efficient CMOL gate designs for cryptography applications. IEEE Trans Nanotechnol 8(3):315–321
2. Abid Z, Liu M, Wang W (2011) 3D integration of CMOL structures for FPGA applications. IEEE Trans Comput 60(4):463–471
3. Brglez F, Bryan D, Kozminski K (1989) Combinational profiles of sequential benchmark circuits. In: IEEE international symposium on circuits and systems, 1989, vol 3, pp 1929–1934
4. Butts M, DeHon A (2002) Molecular electronics: devices, systems and tools for gigagate, gigabit chips. In: ICCAD-2002, pp 433–440

5. Chen G, Song X, Hu P (2009) A theoretical investigation on CMOL FPGA cell assignment problem. IEEE Trans Nanotechnol 8(3):322–329
6. Chu Z, Xia Y, Hung WN, Wang L, Song X (2010) A memetic approach for nanoscale hybrid circuit cell mapping. In: 2010 13th euromicro conference on digital system design: architectures, methods and tools (DSD), pp 681–688
7. Cortes P, Muñuzuri J, Onieva L, Fernández J (2011) A tabu search algorithm for dynamic routing in ATM cell-switching networks. Appl Soft Comput 11(1):449–459
8. Dong C, Wang W, Haruehanroengra S (2006) Efficient logic architectures for CMOL nanoelectronic circuits. Micro Nano Lett 1(2):74–78
9. Garbolino T, Papa G (2010) Genetic algorithm for test pattern generator design. Appl Intell 32(2, SI):193–204
10. Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Norwell
11. Hamidipour H, Keshavarzi P, Naderi A (2010) Routing congestion removing of CMOL FPGA circuits by a recursive method. In: Proceedings of the 9th WSEAS international conference on microelectronics, nanoelectronics, optoelectronics, MINO'10. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, pp 75–79
12. Hedar AR, Ali A (2012) Tabu search with multi-level neighborhood structures for high dimensional problems. Appl Intell 37:189–206. doi:10.1007/s10489-011-0321-0
13. Hung WN, Gao C, Song X, Hammerstrom D (2008) Defect-tolerant CMOL cell assignment via satisfiability. IEEE Sens J 8(6):823–830
14. Khan SA, Engelbrecht AP (2012) A fuzzy particle swarm optimization algorithm for computer communication network topology design. Appl Intell 36(1):161–177
15. Korkmaz EE (2010) Multi-objective genetic algorithms for grouping problems. Appl Intell 33(2):179–192
16. Lee CY, Lee ZJ, Lin SW, Ying KC (2010) An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem. Appl Intell 32(1):88–95
17. Likharev KK (2011) Crossnets: neuromorphic hybrid CMOS/nanoelectronic networks. Sci Adv Mater 3:322–332
18. McKendall AR Jr, Jaramillo J (2006) A tabu search heuristic for the dynamic space allocation problem. Comput Oper Res 33(3):768–789
19. Minhas MR, Sait SM (2005) A parallel tabu search algorithm for optimizing multiobjective VLSI placement. In: ICCSA (4). Lecture notes in computer science, vol 3483. Springer, Berlin, pp 587–595
20. Ozyer T, Zhang M, Alhajj R (2011) Integrating multi-objective genetic algorithm based clustering and data partitioning for skyline computation. Appl Intell 35(1):110–122
21. Sait SM, Khan JA (2003) Simulated evolution for timing and low power VLSI standard cell placement. Eng Appl Artif Intell 16(5–6):407–423
22. Sait SM, Youssef H (1995) VLSI physical design automation: theory and practice. McGraw-Hill, New York
23. Sait SM, Youssef H (1999) Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems. IEEE Comput Soc, Los Alamitos
24. Shahookar K, Mazumder P (1991) VLSI cell placement techniques. ACM Comput Surv 2(23):143–220
25. Snider GS, Williams SR (2007) Nano/CMOS architectures using a field-programmable nanowire interconnect. Nanotechnology 18(3):035204
26. Strukov DB, Likharev KK (2005) CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. Nanotechnology 16(6):888–900
27. Strukov DB, Likharev KK (2005) Prospects for terabit-scale nanoelectronic memories. Nanotechnology 16(1):137
28. Strukov DB, Likharev KK (2006) CMOL FPGA circuits. In: Proc of int conf on computer design, CDES'2006, pp 213–219
29. Strukov DB, Likharev KK (2006) A reconfigurable architecture for hybrid CMOS/nanodevice circuits. In: Proceedings of the 2006 ACM/SIGDA 14th international symposium on field programmable gate arrays, FPGA'06. ACM, New York, pp 131–140
30. Wang K, Zheng YJ (2012) A new particle swarm optimization algorithm for fuzzy optimization of armored vehicle scheme design. Appl Intell 37(4):520–526
31. Wu J, Abbas-Turki A, El Moudni A (2012) Cooperative driving: an ant colony system for autonomous intersection management. Appl Intell 37(2):207–222
32. Xia Y, Chu Z, Hung WN, Wang L, Song X (2010) CMOL cell assignment by genetic algorithm. In: 2010 8th IEEE international NEWCAS conference (NEWCAS), pp 25–28
33. Xia Y, Chu Z, Hung W, Wang L, Song X (2011) An integrated optimization approach for nano-hybrid circuit cell mapping. IEEE Trans Nanotechnol 10(6):1275–1284
34. Zhang C, Li P, Guan Z, Rao Y (2007) A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. Comput Oper Res 34(11):3229–3242

Sadiq M. Sait obtained a Bachelor's degree in Electronics from Bangalore University in 1981, and Master's and Ph.D. degrees in Electrical Engineering from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in 1983 & 1987 respectively. Since 1987 he has been working at the Department of Computer Engineering where he is now a Professor. In 1981 Sait received the best Electronic Engineer award from the Indian Institute of Electrical Engineers, Bangalore (where he was born). In 1990, 1994 & 1999 he was awarded the 'Distinguished Researcher Award' by KFUPM. In 1988, 1989, 1990, 1995 & 2000 he was nominated by the Computer Engineering Department for the 'Best Teacher Award' which he received in 1995, and 2000. Sait has authored over 200 research papers, contributed chapters to technical books, and lectured in over 25 countries. Sadiq M. Sait is the principal author of the books (1) VLSI Physical Design Automation: Theory & Practice, published by McGraw-Hill Book Co., Europe, (and also co-published by IEEE Press), January 1995, and (2) Iterative Computer Algorithms with Applications in Engineering (Solving Combinatorial Optimization Problems): published by IEEE Computer Society Press, California, USA, 1999. He was the Head of Computer Engineering Department, KFUPM from January 2001–December 2004, Director of Information Technology and CIO of KFUPM between 2005 and 2011, and now is the Director of the Center for Communications and IT Research at the Research Institute of KFUPM.



Abdalrahman M. Arafeh received his B.S. degree in Computer Engineering from Damascus University, Damascus, Syria, and MS degree from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia. He coauthored three conference papers and two journal papers. His research interests include FPGA, VLSI, Design Automation, and CAD.