Journal of Circuits, Systems, and Computers
Vol. 25, No. 4 (2016) 1650023 (18 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0218126616500237

World Scientific
www.worldscientific.com

Engineering a Memetic Algorithm from Discrete Cuckoo Search and Tabu Search for Cell Assignment of Hybrid Nanoscale CMOL Circuits*

Sadiq M. Sait† and Feras Chikh Oughali‡

Center for Communications & IT Research,
Computer Engineering Department,
King Fahd University of Petroleum & Minerals,
Dhahran 31261, Saudi Arabia
†*sadiq@kfupm.edu.sa*
‡*foughali@kfupm.edu.sa*

Abdalrahman M. Arafeh§

Department of Electrical and Computer Engineering,
University of British Columbia,
Vancouver, Canada
arafeh@ece.ubc.ca

Received 17 August 2014
Accepted 14 September 2015
Published

Cuckoo search optimization (CSO) algorithm, a recently proposed metaheuristic, has shown promising results in various problem domains. Results from recent studies show that engineering and tuning discrete cuckoo search optimization' parameters is a daunting task. In this paper, an attempt to enhance the performance of the CSO algorithm in solving discrete combinatorial optimization problems is presented. Performance of the discrete modified CSO algorithm is compared with genetic algorithm (GA), particle swarm optimization (PSO), hybrid of GA/PSO, and simulated annealing. In addition, a memetic algorithm (MA) that combines discrete modified CSO and tabu search is proposed. Results show that the proposed improvements help in enhancing the performance of the original algorithm. As a test case, the NP-hard problem of buffer minimization in CMOL (CMOS + nanowire + MOLecules) circuits is addressed. The performance of the proposed implementation of CSO algorithm is compared with other heuristics.

Keywords: Discrete cuckoo search optimization; CMOL placement problem; combinatorial optimization; nature-inspired algorithm; metaheuristic; memetic algorithm.

*This paper was recommended by Regional Editor Emre Salman.
‡Corresponding author.
§Abdalrahman M. Arafeh was with King Fahd University of Petroleum & Minerals when this work was done.

1. Introduction

Cuckoo search optimization (CSO) algorithm is a recently proposed nature-inspired metaheuristic. Originally, the CSO algorithm was designed and tested to solve continuous optimization problems. For the continuous optimization case, the optimal solutions obtained by the CSO are far better than the best solutions obtained by efficient particle swarm optimization (PSO) and genetic algorithm (GA).[1] The cuckoo search optimization has shown promising results in various problem domains which include engineering optimization,[2] software implementation,[3] structural optimization,[4] scheduling,[5] and learning algorithm improvement.[6] Results from recent studies for the discrete combinatorial optimization problems show that more work is need to be done to understand and enhance the standard CSO algorithm for it to compete with other classic non-deterministic iterative heuristics[7] such as simulated annealing (SA), tabu search (TS), GA, PSO, ant-colony optimization (ACO), and the like. Literature also claims that discrete cuckoo search algorithm is inefficient in comparison with other optimization algorithms.[8,9]

The placement problem occurring in the optimization of digital logic circuits in CMOL technology is used to compare the performance of CSO algorithm against other heuristics. CMOL placement optimization is chosen as it has been exhaustively tested with many algorithms such as SA, PSO,[10] GA,[11] simulated evolution (SimE),[12] and TS,[13] which makes it a very good case study. In this paper, an attempt is made to enhance the efficiency of the cuckoo search algorithm in solving discrete optimization problems.

The rest of the paper is organized as follows. Section 2 introduces cuckoo-inspired algorithms and highlights the attempts to use them to solve combinatorial optimization problems. In Sec. 3, CMOL placement problem is discussed. In Sec. 4, the implementation of discrete modified cuckoo search (DMCS) algorithm is presented and its performance is evaluated. Furthermore, some improvements are proposed and their effects are discussed. Section 5 presents the proposed memetic algorithm (MA). Section 6 reports the experimental results. Finally, Sec. 7 concludes this work.

2. Cuckoo-Inspired Algorithms

2.1. *Cuckoo search*

Cuckoo search optimigation is a metaheuristic search algorithm which has been proposed recently by Yang and Deb.[1] It is a novel algorithm inspired by the breeding behavior of some cuckoo species. Cuckoos lay their eggs in the nests of other host birds, and may remove others' eggs to increase the hatching probability of their own eggs. They are often very specialized in mimicking the color and pattern of chosen host species.[2] This reduces the probability of their eggs being abandoned and thus increases their reproductivity. They often choose a nest where the host bird has just laid its own eggs. In general, the cuckoos' eggs hatch slightly earlier than their hosts' eggs. Once the first cuckoo chick is hatched, it will blindly propel the other eggs out of the nest thereby increasing its share of food provided by the host bird. Yang and Deb suggested three idealized rules to apply their algorithm.

- Each cuckoo lays one egg, which represents a solution, at a time and dumps it in a randomly chosen nest.
- A fraction of the nests containing high quality eggs will carry over to the next generation.
- The number of available host nests is fixed, and there is a probability that a host can discover an egg laid by a cuckoo. In this case, the host bird can either throw the egg away or abandon the nest. This can be approximated by the fraction P_a of the n nests being replaced by new nests in new locations.

New solutions are generated by performing Lévy flight from the current ones. Lévy flight is essentially a random walk where the random step length is drawn from a Lévy distribution. This random walk via Lévy flight is very efficient in exploring the search space. To generate new solutions $x^{(t+1)}$ for a cuckoo i, Lévy flight is performed as in Eq. (1), where in general $x_i^{(t)}$ is the current solution, $\alpha > 0$ is the step size, and product $\oplus$ means entry-wise multiplications.

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \text{L'evy}(\lambda)\,. \tag{1}$$

To map the behavior of cuckoo search to optimization problems, the quality of a nest can be considered proportional to the value of the objective function. The aim is to use the new and potentially better solutions (cuckoo's egg) to replace a relatively bad solution in the nests.

According to Yang and Deb, there are some significant differences between CSO and other algorithms. CSO is a population-based algorithm, but it uses some sort of elitism or selection; the randomization in CSO is more efficient as the step length is heavy-tailed, and any large step is possible; the number of parameters to be tuned is low, and thus it has more potential to adapt to a wider class of optimization problems.

2.2. *Modified cuckoo search*

As stated by Yang and Deb,[2] given enough computation, the CSO will always find the optimum solution but, as the search relies entirely on random walks, a fast convergence cannot be guaranteed. Walton *et al.* presented two modifications to the original CSO method with the aim of increasing the convergence rate.[14] This should improve the performance of CSO and make it more practical for a wider range of applications, but without losing the attractive features of the original method.

The first modification is made to the size of the Lévy flight step size α; the value of α will decrease as the number of generations increase. This is done to encourage more localized search as the eggs get closer to the optimal solution.

In its essence, CSO has no information exchange between individuals, and the searches are performed independently; the second modification is done to add information exchange between the eggs in an attempt to speed up convergence to a minimum.

2.3. *Cuckoo search and discrete combinatorial optimization*

Cuckoo search algorithm has previously shown promising results in solving continuous optimization problems. This work evaluates the performance of CSO in solving discrete optimization problems. Jati *et al.* constructed a cuckoo search algorithm to solve the traveling salesman problem (TSP) to explore the potential of CSO in solving discrete problems.[8] Experimental results showed that the proposed algorithm performs very well to solve some simple TSP instances, but it can be trapped into local optimum solutions for other complex instances.

Syberfeldt and Lidberg[9] did a case study of real-world optimization of an engine manufacturing line using CSO. They used an extension of the original CSO algorithm to handle multiple objectives. Results showed that the extended CSO algorithm is inefficient in comparison with the multi-objective benchmark algorithm NSGA-II. It has been suggested that CSO algorithm might not suit combinatorial optimization problems as the Lévy flight pattern is not suited to be used as a basis for swap mutation. The authors also recommended to study other possible ways to adapt CSO to combinatorial optimization problems.

3. CMOL Placement Problem

Recently, a new trend is emerging for combining the advantages of CMOS technology, mainly its flexibility and high fabrication yield, with nanometer-scale molecular devices. In this regard, Likharev and Strukov[15] 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),[16] neuromorphic Cross-Nets,[17] and in memories.[18]

Complementary metal-oxide semiconductor (CMOS) stack is connected to nanofabric by metal pins that span to top and bottom of the nanowire levels as shown in Fig. 1. Two CMOS inverters are connected by pin–nanowire–nanodevice–nanowire–pin connection.

Like other nanofabric crossbars, CMOLs nanowires break at repeated intervals confining CMOL cells connectivity to only $M = 2r(r-1) - 1$ other cells located within its proximity "Connectivity Domain", where r is an integer value that indicates connectivity diameter and represents the constraint of CMOL placement.

The abundance of available nanodevices and nanowires provides a variety of different possible configurations for the implementation of one circuitry. Among

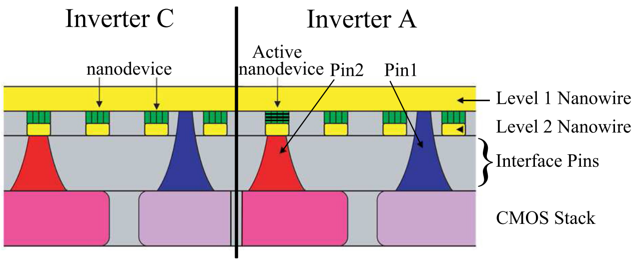

Fig. 1. Schematic side view of two CMOL cells with two levels of nanowires. Only one nanodevice is activated to connect the output of Inverter A to the input of Inverter C (color online).

those, their could be only certain configurations that satisfy connectivity domain constraint and do not require additional routing resources.

3.1. *Problem formulation*

Given a pool of NOR/INV gates, and a pool of nets (the set of inputs/outputs to be connected together), the CMOL placement problem can be formulated as a mapping of a given NOR/INV gate-based circuit G to a CMOL generic inverter-based cell array Ψ. Each CMOL cell can implement one inverter or one NOR gate with multiple fan-in. Each gate in G has a number of fan-in and fan-out gates, those comprise γ_i the netlist of gate i, as described in Eq. (2).

$$P : G \to \Psi\,, \tag{2a}$$

$$\gamma_i = \{\text{fan-in}(i)|\text{fan-out}(i)\}\,. \tag{2b}$$

Unlike conventional CMOS-based cell assignment, CMOL cell placement is constrained to a "*Connectivity Domain*" of radius r as shown in Fig. 2. Each CMOL cell can be connected to one of its proximity cell members, any violation to this constraint would impose introducing a buffer to satisfy connectivity. However, such a process could result in substantial increase in timing delay. The "Connectivity Domain" can be defined as follows. 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 constraint should be satisfied:

$$\forall i, k \in G : \text{dist}\ (L_i, L_k) \leq r\,, \tag{3}$$

where L_k is the location of g_k, dist is Manhattan distance, and r is CMOL connectivity diameter. The objective of CMOL cell mapping problem is to satisfy the constraint in Eq. (3) for all gates of circuit G.

3.2. *Literature review*

Previous attempts to use sub-optimal search heuristics are reported in Refs. 10–13, 19 and 20. Genetic algorithm[11] was used with two-dimensional block PMX crossover

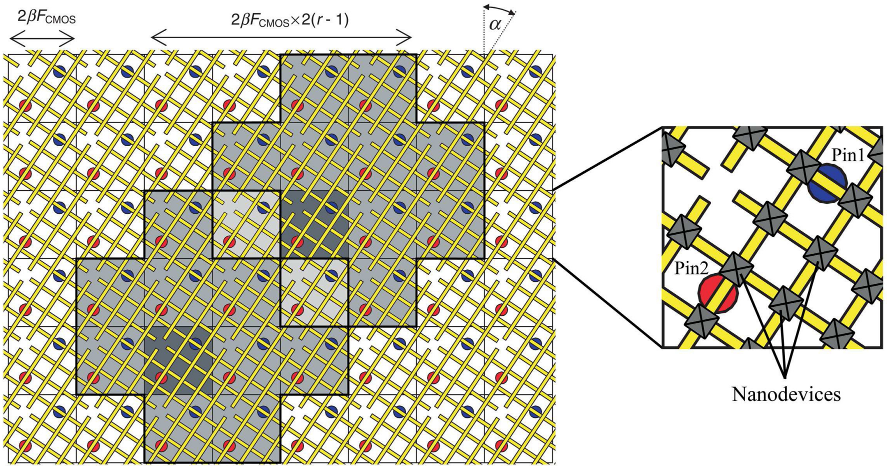
$2\beta F_{\mathrm{CMOS}}$
$2\beta F_{\mathrm{CMOS}}\times 2(r-1)$
α
Pin1
Pin2
Nanodevices

Fig. 2. CMOL FPGA topology for $r = 3$, $M = 11$ cells in the "Connectivity Domain" (highlighted by dark line) for the input pin of cells painted in dark grey. The overlap between connectivity domains of two cells is shown in light grey (color online).

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 Ref. 19 where a memetic computing approach was used by implementing a hybrid of the GA and the SA (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.*[20] 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 LRMA to penalty values corresponding to problem constraints and repeatedly updating them. Results reported using LRMA approach are promising, however, more computation is needed for the penalty updating mechanism and SA local-based search.

Particle swarm optimization[10] was used for solving the cell assignment in CMOL. In the proposed algorithm, the PSO operators such as velocity and position update were defined in the context of assignment problem. Due to the inherent greedy nature of PSO algorithm, SA was used to aid in escaping local minima. The proposed method takes advantage of the exploration and exploitation factors of PSO and the intrinsic hill climbing feature of SA to reduce the number of buffers to be inserted.

Simulated evolution algorithm was also used to solve this problem.[12] A novel goodness and allocation functions were devized that exploit better understanding of the limitations imposed by CMOL connectivity radius.

Tabu search algorithm was engineered to provide sub-optimal solution by efficient exploration of search space.[13] A probabilistic method was proposed to make tailored swaps and reduce candidate list size. TS exhibited more intelligent search of the solutions subspace, and was able to find better solutions in less time compared to previous attempts.

4. Discrete Modified Cuckoo Search

In this section, solution representation, DMCS implementation, and the suggested improvements are presented. A nest in DMCS population is represented as a 2D grid with dimensions $X \times Y$. The outer cells of the grid are reserved for I/O pins, where I/O pins moves are restricted to those reserved locations. In the initialization phase, gates are randomly assigned to locations in the 2D layout as shown in Fig. 3.

The main objective of the placement is to find a feasible assignment of cells in which all connections are satisfied. However, in CMOL placement, the problem is to place connected cells within each other's connectivity domain to avoid insertions of additional buffers. 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 violations of the connectivity domain constraint. In other words, the overall cost is the number of additional buffers that are needed to satisfy all connections. The details of the MCS algorithm are shown in Algorithm 1.

In the MCS, a fraction of the eggs with the best fitness are put into a group of top eggs. For each of the top eggs, a second egg in that group is randomly picked and a new egg is generated on the line connecting those two top eggs. The distance along that line at which the new egg is located is calculated using the inverse of the golden ratio $\psi = (1 + \sqrt{5})/2$. In case the same egg is picked twice, a local Lévy flight search is performed from the randomly picked nest with step size $\alpha = A/(2 * \sqrt{G})$, where G is the generation number. There is one parameter that controls the fraction of nests to be abandoned and the fraction of nests to make up the top nests, which needs to be

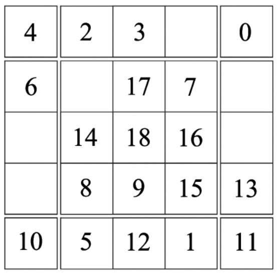

Fig. 3. Nest representation of CMOL initial placement of *s27.blif*. 19 cells, 8 gates, 7 inputs and 4 outputs.

Algorithm 1 Modified Cuckoo Search Algorithm

```
1: Set $MAX$ {$MAX$ is the maximum number of iterations}
2: Set $P_a$ {$P_a$ is the fraction of the entire population to abandon}
3: GoldenRatio $\psi \leftarrow 1.62$
4: Initialize a population of $n$ nests $x_i (i = 1, 2, \ldots, n)$
5: for all $x_i$ do
6:    Calculate fitness $F_i = f(x_i)$
7: end for
8: Generation number $G \leftarrow 1$
9: while $NumberObjectiveEvaluations < MAX$ do
10:    $G \leftarrow G + 1$
11:    Sort nests by order of fitness
12:    for all nests to be abandoned do
13:       Current position $x_i$
14:       Calculate Lévy flight step size $\alpha \leftarrow A/\sqrt{G}$
15:       Perform Lévy flight from $x_i$ to get new egg $x_k$
16:       $x_i \leftarrow x_k$, $F_i \leftarrow f(x_i)$
17:    end for
18:    for all of the top nests do
19:       Current position $x_i$
20:       Pick another nest from the top nests at random $x_j$
21:       if $x_i = x_j$ then
22:          Calculate Lévy flight step size $\alpha \leftarrow A/(2 * \sqrt{G})$
23:          Perform Lévy flight from $x_i$ to get new egg $x_k$
24:          $F_k = f(x_k)$
25:          Choose a random nest $l$ from all nests
26:          if $F_k > F_l$ then
27:             $x_l \leftarrow x_k$, $F_l \leftarrow F_k$
28:          end if
29:       else
30:          $dx = |x_i - x_j|/\psi$
31:          Move distance $dx$ from nest $i$ to $j$ to find $x_k$
32:          $F_k = f(x_k)$
33:          Choose a random nest $l$ from all nests
34:          if $F_k > F_l$ then
35:             $x_l \leftarrow x_k$, $F_l \leftarrow F_k$
36:          end if
37:       end if
38:    end for
39: end while
```

adjusted in the MCS. Through testing on the benchmarks, it was found that setting the fraction of nests to be abandoned to 0.5 yielded the best results.

4.1. *DMCS implementation*

Modified cuckoo search algorithm described in Algorithm 1 has been implemented to solve the CMOL placement problem. As this problem is a discrete one, three major parts in the algorithm have to be redefined to cope with the nature of the problem. These are, *step size* of the Lévy flight, performing *Lévy flight* from one nest to another, and *moving a distance* from one nest toward another one.

In the suggested implementation, a Lévy flight from one nest to another is performed by randomly, pairwise, swapping the locations of cells. The number of pairs of cells to swap their locations is controlled by the *step size* of the Lévy flight which is an integer number in this case. For example, if the *step size* equals to 3, then three pairs of cells will swap their respective locations in the grid. The lower bound of the *step size* is set to 1, while the upper bound is set to $(X \times Y)/2$. Figure 4 depicts the step size versus iterations. It is clearly visible that the step size is higher at the beginning. The likelihood of making large steps is also higher at the beginning.

Moving from one nest toward the other is implemented by assigning portion of the cells in one nest to the same locations in the other one. This way, the similarity between different nests is increased assuming that cells are better located in nests with high fitness. Initially, the difference between the two nests, i.e., the number of cells that are not in the same location is computed. Then, the number of cells in the first nest are assigned, based on the golden ratio, to same locations of the second nest. Figure 5 shows an example. There are two nests: nest1 and nest2. These two nests have six cells in common, which means that the difference between them equals to $25 - 6 = 19$ cells.

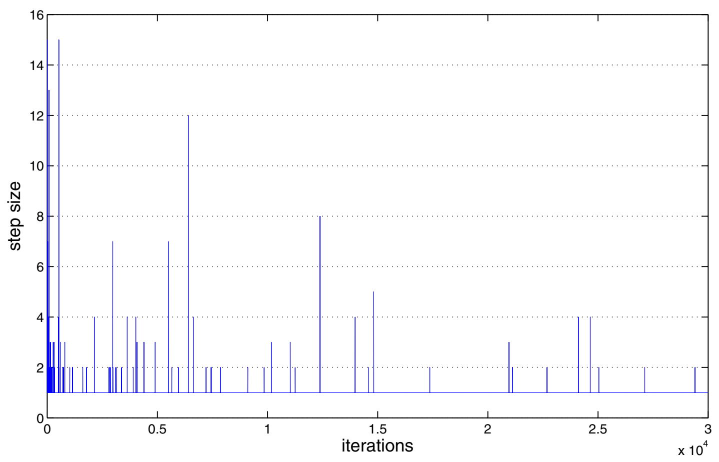

Fig. 4. Lévy flight step size (color online).

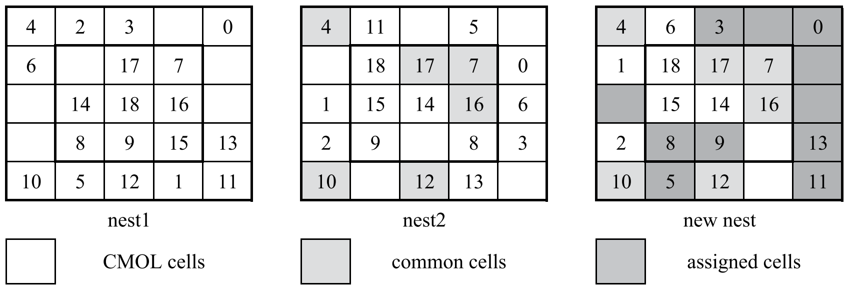

Fig. 5. Generating a new nest from nest1 and nest2.

Using the inverse of the golden ratio, number of cells in nest2 $(19/\psi = 11)$ are assigned to same locations of nest1. The result is presented as new nest.

Using this implementation, MCS performed well for small benchmarks, and it was able to reach 0 buffers. However, its performance degraded notably for larger benchmarks, and it was not able to reach 0 buffers. It is noted that there is a high chance that the algorithm will be trapped in a local minimum.

4.2. *Suggested improvements*

It was noted that implementing the MCS algorithm, as introduced in Ref. 14, to solve the problem at hand did not deliver the same performance as observed for other continuous problems. Hence, in this section some improvements to the algorithm are suggested and their outcome is evaluated.

Due to the nature of the problem, many nests or solutions can have the same cost. Small perturbations, like swapping two neighbor cells, might not affect the quality of the solution nor change its cost. It has been observed that this particular factor is what is causing the algorithm to be trapped in a local minimum. Lines 26 and 34 in Algorithm 1 suggest that only those nests with a better cost value can be accepted. This will limit the exploration of the search space and the algorithm will be stuck with those top nests which it already has. By introducing the equality factor to these two lines as in IF $F_k \geq F_l$, the performance of the algorithm has been remarkably improved. With this small change, the algorithm was able to reach the optimal solution (zero buffers) for the "s400" benchmark circuit for 85% of the runs. However, the original algorithm failed to reach the optimal solution for the "s400" benchmark circuit for all the runs. This statistics is based on 20 runs with the same number of maximum iterations MAX.

Despite the fact that given enough computation time, CSO will always find the optimum solution; considering the size of the search space and the nature of the problem, it is not sufficient to rely merely on random moves as suggested by the algorithm. Hence, it is suggested to use some sort of smart moves in order to improve the convergence time of the algorithm.

One way of performing smart moves is by evaluating the fitness of each element in the gird. Based on that smart moves can be performed by relocating those poorly located elements. The same goodness measure introduced in Ref. 12 is adopted to evaluate the fitness of each element. The fitness of each element can be expressed as follows:

$$\text{fitness}_i = \frac{\text{inside}_i}{|\gamma_i|}\,, \tag{4}$$

where inside_i represents the number of gates in set γ_i that satisfy connectivity constraint (i.e., inside the connectivity domain of element i) and $|\gamma_i|$ is the cardinality of the netlist of gate i. Figure 6 shows an example of how fitness value is calculated, where two gates in γ_i are outside the connectivity domain of gate i and three otherwise. This fitness function results in a precise selection of those elements that violate the constraint expressed in Eq. (3), which directs the heuristic into enhancing the overall cost of the solution.

Since the number of cells that violate the given constraint are far less than the total number of cells, it is rational for swaps to always include cell(s) that violate the connectivity radius constraint, or those with low fitness values. To avoid being greedy, and not curtail exploration, the following approach was adopted from Ref. 13. The list of all of the circuit's cells are sorted according to their fitness. When selecting cells for swaps, one cell is selected randomly while the other is selected probabilistically. Those cells on the top of the list which have more violations have a higher chance of being selected, while those cells that do not violate the constraint also have a small non-zero probability of being selected. To perform the probabilistic selection, the positive values of a Gaussian random variable which has mean $m_0 = 0$ and standard deviation $3\sigma = \mathit{circuit} - \mathit{size}$ are used. Given the Gaussian distribution, cells in the top of the list will be more frequently selected than those in the bottom of the list.

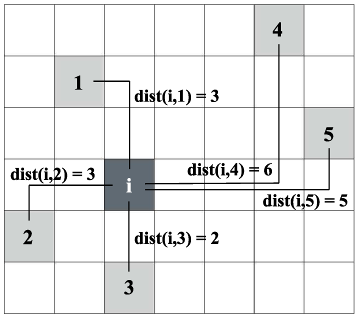

Fig. 6. Evaluation of gate i's goodness; for $r = 3$ cells 1, 2 and 3 are inside i's connectivity domain (i.e., dist $\leq r$), while cells 4 and 5 are out of it (i.e., dist $> r$). $\text{goodness}_i = 3/5 = 0.6$.

5. Memetic Algorithm

The term "Memetic Algorithms" (MAs) is used to denote a family of metaheuristics that blend together several concepts from separated families such as evolutionary algorithms (EAs) and SA for instance. The term "memetic" comes from "meme" which was coined by Dawkins[21] to denote an analogous to the gene in the context of cultural evolution. MAs are population-based metaheuristics that maintain a pool of solutions for the problem at hand. Each of these solutions is called an individual in the EA domain. These solutions are subject to processes of competition and mutual cooperation in a way that resembles the behavioral patterns of living beings from a same species. The pool of solutions is called a "generation". Each generation consists of updating a population of individuals that, hopefully, will lead to better solutions.

Essentially, a mutation operator must generate a new solution by partly modifying an existing one. This modification can be random or can be endowed with problem-dependent information so as to bias the search to probably-good regions of the search space. It is due to this possibility that one of the most distinctive components of MAs is introduced: local-improvers.[22] The local-improver algorithm can be used in different parts of the generation process. For instance, it can be inserted after the utilization of any other mutation operator; alternatively, it can be used at the end of the reproduction stage.

Despite the improvements suggested in Sec. 4, DMCS (like other population-based algorithms) still lacks the competency with algorithms like SimE. Syberfeldt and Lidberg[9] also stated that the Lévy flight pattern might not be suitable as a basis for swap mutation. We believe that hybridizing DMCS with other algorithms will lead to a better performance and open the floor for the algorithm to tackle similar problems effectively.

Based on the above, the memetic algorithm described in Algorithm 2 is proposed. This algorithm starts with a population of random solutions. Then, it uses the proposed DMCS to produce a new generation. After that, TS is used to improve the quality of the top nests obtained from DMCS. Finally, the MA updates the list of top nests and repeat the process again. The same TS implementation introduced

Algorithm 2 DMCS-TS Hybrid Memetic Algorithm

```
1: initialize population $P$
2: repeat
3:    $P \leftarrow DMCS(P)$
4:    for each individual $i \in$ top-nests do
5:       $i$ =Tabu-Local-Search($i$)
6:    end for
7:    update top-nests list
8: until MA-Termination-Criterion()
```

in Ref. 13 is used to perform the local search to improve the quality of the top nests. The local improvement is performed on the top nests only, as the other nests are used to diversify the search in the search space. The local improvement process is performed in each iteration on the top nests after the generation of the new population. TS runs for 50 iterations, which provides a high quality result in reasonable run time. The size of the candidate list is 50, while the size of the tabu list is 5. The population size of DMCS is reduced to 10 in order to reduce the run time.

6. Experimental Results

Evaluation of DMCS performance and behavior is conducted using ISCAS89 (Ref. 23) benchmarks. The benchmarks used in this work are mapped to NOR-based gates with maximum of five inputs. DMCS has been implemented using Java programming language. Initially, fine tuning to the parameters of the algorithm is performed. The first parameter to consider is the size of the population or the number of nests n. Figure 7 shows the change in the obtained cost as a result of changing the population size. It can be clearly seen that up to a certain point increasing the number of nests does enhance the cost of the solution. However, after a certain point the improvement in cost is negligible.

The second parameter to consider is the percentage of the top nests. Figure 8 depicts the change in cost as a result of changing the percentage of the top nests. It is evident that increasing the percentage of the top nests does enhance the cost of the solution. However, after a certain point, increasing the percentage of the top nests has a negative impact on the quality of the obtained solution. A percentage between 50% and 60% does produce the best results. These experiments are done for a medium size circuit (s510) and a large size circuit (s1196); both of them show the same trend for the two experiments.

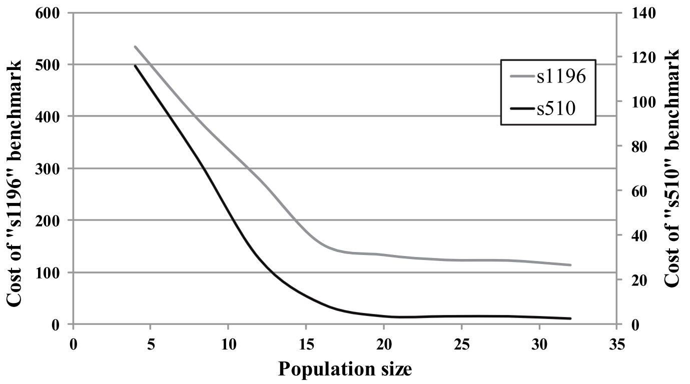

Fig. 7. Change in nest cost versus population size for “s510” and “1196” benchmarks.

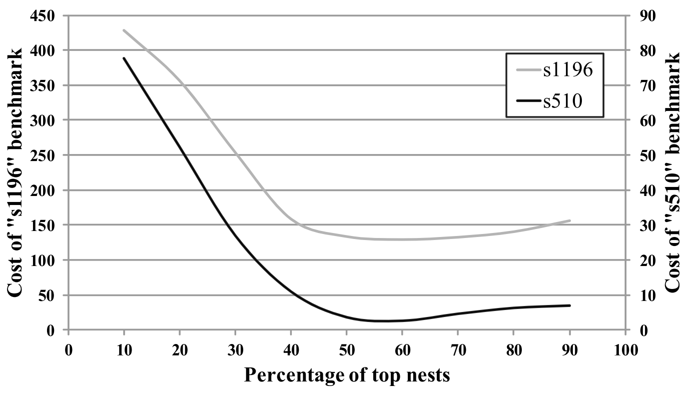

Fig. 8. Change in nest cost versus percentage of top nests for "s510" and "1196" benchmarks.

As a result, population size (number of nests) in DMCS is set to 20, P_a (the fraction of the entire population to be abandoned) is set to 50%, and MAX (the maximum number of iterations) equals to 30,000. The median value of 20 runs is reported in Tables 2 and 3.

Table 1 shows the number of cells (i.e., gates and I/Os) of benchmark circuits used; Area (Row $\times$ Column) is the area used in GA, MA, LRMA, PSO, SimE, TS and DMCS.

Figure 9 shows a comparison of change in nest cost per iteration for "s820" benchmark for DMCS and the hybrid algorithm. It is evident that the suggested

Table 1. ISCAS'89 Benchmarks: Area is the size of CMOL 2D grid. AU% is area utilization.

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

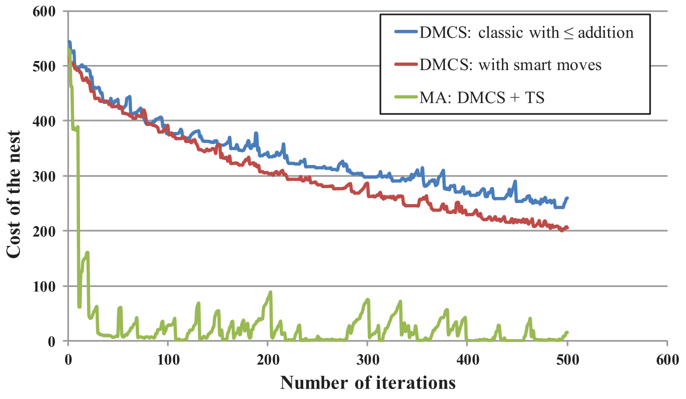

Fig. 9. Comparison of change in nest cost per iteration of s820.blif ($r = 12$) (color online).

improvements do enhance the performance of the algorithm. However, the hybrid algorithm converges much faster than the original one. This suggests, as mentioned earlier, that the Lévy flight pattern is not suitable as a basis for swap mutation.

Table 2 shows the final results obtained for the population-based algorithms GA, PSO, and DMC; Time is the computation time in seconds, Buf reports the number of inserted buffers to satisfy CMOL connectivity domain. Results shows that for all circuits, DMCS outperforms PSO in terms of computation time and the total number

Table 2. ISCAS'89 comparison of population-based algorithms GA, PSO and DMCS.

	GA		PSO		DMCS	
Benchmarks	Time (s)	# BUF	Time (s)	# BUF	Time (s)	# BUF
s27	0.01	0	0.01	0	0.01	0
s208	1.12	0	4	0	0.28	0
s298	0.17	0	4	0	0.32	0
s344	0.57	0	8	0	0.55	0
s349	0.49	0	12	0	0.58	0
s382	1.6	0	13	1	3.17	0
s386	1.05	0	11	2	1.74	0
s400	2.12	1	20	1	4.25	0
s420	8.5	1	40	4	54	1
s444	1.86	2	65	1	3.72	0
s510	16.56	2	60	8	63	2
s526	9.75	5	185	6	57	2
s641	82.66	15	220	37	63	13
s713	52.84	34	250	39	70	18
s820	77.52	41	400	126	128	41
s832	69.27	54	350	115	134	47
s838	201.37	50	600	70	160	46
s1196	234.88	84	705	188	205	99
s1238	268.92	121	1500	240	231	140
Avg	54.28	22	234.05	44	62.14	22

of inserted buffers. For almost all circuits (except the last two), the number of needed buffers obtained from DCMS is similar or even better than those obtained from GA; GA outperformed DMCS for the last two circuits, namely "s1196" and "s1238". On an average, the performance of DMCS and GA is the same for the number of buffers, however, GA requires less computation time.

With respect to the suggested improvements, the performance of DMCS is notably enhanced; DMCS is matching or outperforming the efficiency of other population-based optimization algorithms like GA and PSO. On the other hand, it seems that these population-based algorithms (i.e., GA, PSO and DMCS) are inefficient, for solving this type of problems, in comparison with other optimization algorithms like SimE [12] which works on a single solution or other hybrid algorithms.

Fortunately, as it has been shown in Refs. 10, 19 and 20, hybridizing those population-based algorithms with other algorithms has an encouraging potential. Results of these algorithms along with the proposed MA, which combines DMCS with TS, are reported in Table 3. The proposed algorithm outperforms all other attempts in terms of computation time and quality of the solution. The proposed algorithm was able to reach the optimal solution for all the benchmarks circuits. This dramatic improvement has been achieved, thanks to the well-engineered TS which is used to improve the quality of the individuals of the cuckoo search population. To be more precise, it has been found that the quality of the solution is mostly driven by TS. This observation is made due to the fact that tuning the parameters of DMCS

Table 3. ISCAS'89 comparison of hybrid algorithms MA, LRMA, PSO+SA and DMCS+TS.

	MA (GA+SA)		LRMA		PSO+SA		DMCS+TS	
Benchmarks	Time (s)	# BUF	Time (s)	# BUF	Time (s)	# BUF	Time (s)	# BUF
s27	0.01	0	0.01	0	0.01	0	0.02	0
s208	0.12	0	0.1	0	0.01	0	0.04	0
s298	0.11	0	0.09	0	0.01	0	0.10	0
s344	0.29	0	0.16	0	2.12	0	0.17	0
s349	0.28	0	0.18	0	2.67	0	0.19	0
s382	0.38	0	0.32	0	3.52	0	0.25	0
s386	0.33	0	0.34	0	3.62	0	0.33	0
s400	0.4	0	0.34	0	2.08	0	0.30	0
s420	3.41	0	1.57	0	20.11	0	0.38	0
s444	0.4	0	0.34	0	4.39	0	0.17	0
s510	7.56	0	3.42	0	40.23	0	0.55	0
s526	4.36	0	1.59	0	30.25	0	0.54	0
s641	39.4	4	22.02	0	120.77	0	5.46	0
s713	30.11	3	41.77	2	120.73	2	4.34	0
s820	61.71	10	54.09	6	250.32	4	10.99	0
s832	60.17	11	63.77	4	180.37	6	22.46	0
s838	85.62	7	100.4	4	250.12	4	17.15	0
s1196	208.15	19	179.47	9	301.47	1	32.52	0
s1238	267.34	31	353	9	450.61	27	46.42	0
Avg	40.53	4	43.31	2	93.86	2	7.49	0

did not help much in improving the quality of the obtained solutions. However, adjusting the parameters of TS had a big influence on the quality of the solution.

As a result, this work shows that population-based algorithms are not suitable for this type of discrete combinatorial problems. However, its performance can be further improved by using problem-dependent information to bias the search toward good regions in the search space, and by utilizing algorithms like SA and TS to improve the quality of its individuals. The performance of DMCS was comparable with the performance of other algorithms in its class (population-based algorithms).

7. Conclusion

In this work, discrete CSO algorithm to solve CMOL placement problem is implemented and evaluated. It is found, as indicated by previous works, that classic version of CSO is not suited for combinatorial optimization. Some improvements are introduced to enhance the performance of the algorithm. These improvements suggest some tweaks to cope with the nature of the problem. The proposed improvements helped in enhancing the performance of the original algorithm. Results of DMCS outperformed that of PSO and match those of GA.

Acknowledgments

The authors acknowledge King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia, for all support. Special thanks to Sumaiya S. S. Mohammed and Raza Ali for their help in editing and improving the quality of the manuscript.

References

1. X.-S. Yang and S. Deb, Cuckoo search via Lévy flights, *World Congr. Nature Biologically Inspired Computing, 2009 (NaBIC 2009)*, Dec 2009, pp. 210–214.
2. X.-S. Yang and S. Deb, Engineering optimisation by cuckoo search, *Int. J. Math. Model. Numer. Optimisation* **1** (2010) 330–343.
3. N. Bacanin, An object-oriented software implementation of a novel cuckoo search algorithm, *Proc. 5th European Conf. European Computing Conf. ECC'11* (World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2011), pp. 245–250.
4. A. Gandomi, X.-S. Yang and A. Alavi, Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems, *Eng. Comput.* **29** (2013) 17–35.
5. S. Burnwal and S. Deb, Scheduling optimization of flexible manufacturing system using cuckoo search-based approach, *Int. J. Adv. Manuf. Technol.* **64** (2013) 951–959.
6. H. Salimi, D. Giveki, M. A. Soltanshahi and J. Hatami, Extended mixture of MLP experts by hybrid of conjugate gradient method and modified cuckoo search, *Int. J. Artif. Intell. Appl.* **3** (2012).
7. S. M. Sait and H. Youssef, *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems* (IEEE Computer Society Press, 1999).

8. G. Jati, H. Manurung and S. Suyanto, Discrete cuckoo search for traveling salesman problem, *7th Int. Conf. Computing and Convergence Technology (ICCCT)* (Seoul, Korea, Dec 2012), pp. 993–997.
9. A. Syberfeldt and S. Lidberg, Real-world simulation-based manufacturing optimization using cuckoo search, *Proc. 2012 Winter Simulation Conf. (WSC)*, Dec. 2012, pp. 1–12.
10. S. M. Sait, A. T. Sheikh and A. H. El-Maleh, Cell assignment in hybrid CMOS/nanodevices architecture using a PSO/SA hybrid algorithm, *J. Appl. Res. Technol.* **11** (2013).
11. Y. Xia *et al.*, CMOL cell assignment by genetic algorithm, *Proc. 8th IEEE Int. NEWCAS Conf. 2010*, 20–23 June 2010, pp. 25–28.
12. S. M. Sait and A. M. Arafeh, Efficient CMOL nanoscale hybrid circuit cell assignment using simulated evolution heuristic, *Proc. Great Lakes Symp. VLSI (GLSVLSI '12)*, (ACM, NY, USA, 2012), pp. 21–26.
13. S. M. Sait and A. Arafeh, Cell assignment in hybrid CMOS/nanodevices architecture using Tabu Search, *Appl. Intell.* **40** (2014) 1–12.
14. S. Walton, O. Hassan, K. Morgan and M. Brown, Modified cuckoo search: A new gradient free optimisation algorithm, *Chaos Solitons Fractals* **44** (2011) 710–718.
15. D. B. Strukov and K. K. Likharev, CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices, *Nanotechnology* **16** (2005), p. 888.
16. D. Strukov and K. Likharev, A reconfigurable architecture for hybrid CMOS/nanodevice circuits, *Proc. 2006 ACM/SIGDA 14th Int. Symp. Field Programmable Gate Arrays, FPGA '06* (ACM, NY, USA, 2006), pp. 131–140.
17. K. K. Likharev, CrossNets: Neuromorphic hybrid CMOS/nanoelectronic networks, *Sci. Adv. Mater.* **3** (2011) 322–331.
18. D. B. Strukov and K. K. Likharev, Prospects for terabit-scale nanoelectronic memories, *Nanotechnology* **16** (2005), p. 137.
19. Z. Chu, Y. Xia, W. N. N. Hung, L. Wang and X. Song, A memetic approach for nanoscale hybrid circuit cell mapping, *13th Euromicro Conf. Digital System Design: Architectures, Methods and Tools (DSD)* (Lille, France, Sept. 2010), pp. 681–688.
20. Y. Xia, Z. Chu, W. N. N. Hung, L. Wang and X. Song, An integrated optimization approach for nanohybrid circuit cell mapping, *IEEE Trans. Nanotechnol.* **10**(2011) 1275–1284.
21. R. G. Dawkins, *The Selfish Gene* (Clarendon Press, Oxford, 1976).
22. G. C. Onwubolu and B. V. Babu, Memetic algoriths, *New Optimization Techniques in Engineering* (Springer, Heidelberg, Germany, 2010), pp. 53–72.
23. F. Brglez, D. Bryan and K. Kozminski, Combinational profiles of sequential benchmark circuits, *IEEE Int. Symp. Circuits and Systems*, Vol. 3, May 1989, pp. 1929–1934