SIMULATION OF PSO BASED APPROACH FOR CMOL CELL ASSIGNMENT PROBLEM

Prateek Shrivastava¹, Khemraj Deshmukh²

¹ Electronics & Telecommunication/SSGI/ CSVTU, Bhilai, INDIA
² Electronics & Instrumentation/ SSGI/ CSVTU, Bhilai, INDIA

Abstract:
Particle swarm optimization (PSO) approach is used over genetic algorithms (GAS) to solve many of the same kinds of problems. This optimization technique does not suffer, however, from some of GA’s difficulties; interaction in the group enhances rather than detracts from progress toward the solution. Further, a particle swarm system has memory, which the genetic algorithm does not have. In particle swarm optimization, individuals who fly past optima are tugged to return toward them; knowledge of good solutions is retained by all particles. The genetic algorithm works with the concept of chromosomes having gene where each gene act as a block of one solution. This is totally based on the solution which is followed by crossover and then mutation and finally reaches to fitness. The best fitness will be considered as a result and implemented in the practical area. Due to some drawbacks and problems exist in the genetic algorithm implemented, scientists moved to the other algorithm technique which is apparently based on the flock of birds moving to the target. This effectively overcome the shortcomings of GA and provides better fitness solutions to implement in the circuit.

Keywords:
CMOL CELL, Particle swarm optimization (PSO), genetic algorithms (GAS), algorithm technique, swarm system.


1. INTRODUCTION

In recent years, nanoelectronics has made tremendous progress, with advances in novel nanodevices, nano-circuits, nano-crossbar arrays [1] manufactured by nanoimprint lithography[2] CMOS/nano co-design architectures[3] and applications[4][5]. According to K. K. Likharev and D. V. Strukov discussed in Introduction to Molecular Electronics “Although a nanowire crossbar array (with two-terminal nanodevices) does not have the functionality of FET-based circuits, it has the potential for incredible density and low fabrication costs”.

Likharev and his colleagues have developed the concept of CMOL (Cmos / nanowire /Molecular hybrid) as a likely implementation technology for nanoelectronic devices. Examples include memory, FPGA, and neuromorphic CrossNets [6].Cell assignment is one of the important steps in CMOL based circuit design.
One of the prior works on CMOL was assigning cells by hand. Another CMOL work[7] grouped CMOL cells into $4 \times 4$ tiles, where the CMOL cell assignment can be performed easily within each tile due to its small size, and the high level placement and route were performed at the tile level (but not the CMOL cell level) which is similar to traditional FPGA tools.

In Defect Tolerant CMOL Cell Assignment via Satisfiability given by William N. N. Hung, Changjian Gao, Xiaoyu Song and Dan Hammerstrom, Instead of working with tile level abstraction, Hung[8] presented a technique that directly works on the CMOL cell level.

Consider the above figure; the nanowires are on top of CMOS circuits, with interface pins connecting CMOS metals and nanowires. The pins (blue) connecting with the upper-layer nanowires could break the lower-layer nanowires to relax the requirements for fabrication and increase interface yield.

The nanodevices are sandwiched between the two levels of perpendicular nano-imprinted nanowires. This unique structure solves the problems of addressing much denser nanodevices with sparser CMOS components. Each nanodevice is accessed by the two perpendicular nanowires which connect to the nanodevice. The nanowires are, in turn, connected by pins to the CMOS circuits. With $N$ nanowires and pins, we could address $O(N^2)$ nanodevices.
Given any boolean combinational circuit, we can easily transform the circuit into a net list of NOR gates, using any library-based technology mapping from any logic synthesis tool or the technique outlined in [8]. Mathematically, we can view the NOR gate net list as a directed acyclic graph (DAG) G:

\[ G = (V, E) \]

Where \( V \) is a set of vertices (each vertex correspond to a NOR gate in the net list), and \( E = V \times V \) is a set of directed edges. Given a collection of CMOL cells \( \Psi \). We need to find a mapping from the vertices (NOR gates) to the CMOL cells,

\[ p : V \to \Psi \]

A. GENOTYPE

The genotype is the genetic constitution of a chromosome and for our CMOL cell assignment problem, it is a solution which represent the mapping result. The genotype structure is \( (C_1, C_2, C_3, \ldots, C_N) \), where \( N = ||\Psi|| \) is the number of cells in the CMOL cell array. Each \( C_i \) can be either an index to an NOR gate or -1. The NOR gate index is in the range \([0, M-1]\), where \( M \) is the total number of NOR gates in the circuit. The value -1 indicate the cell has not been occupied by any NOR gate. Each position 1 …N in the chromosome corresponds to a CMOL cell in the array.

B. FITNESS FUNCTION

In genetic algorithm, fitness function is used to evaluate the adaptive ability. This function provides a measurement standard for choosing the best population to the next generation. The CMOL cell assignment problem requires the wire lengths of connected NOR gates to be less than a constant \( R \), the radius of the connectivity domain.

C. GENETIC OPERATORS

1) Crossover: Crossover is one of the main genetic operators in the GA. It takes two parent solutions to reproduce their children. After the selection process, the population is enriched with better individuals. In the process of selection, we adopted Roulette Algorithm to choose the top best parents to crossover based on their fitness and survival probability.

2) Mutation: Mutation occurs after crossover. It is used to prevent the algorithm from being trapped in local minima. Mutation is viewed as an operator which recovers lost genetic pattern as well as randomly disturbs genetic information.
HOW GENETIC ALGORITHMS WORK

Genetic algorithm maintains a population of individuals, say $P(t)$, for generation $t$. Each individual represents a potential solution to the problem at hand. Each individual is evaluated to give some measure of its fitness. Some individuals undergo stochastic transformations by means of genetic operations to form new individuals. There are two type of transformation:-

1) Mutation, which creates new individuals by making changes in a single individual.
2) Crossover, which creates new individuals by combining parts from two individuals.

The new individuals, called offspring $C(t)$, are then evaluated. A new population is formed by selecting the more fit individuals from the parent population and offspring population.
After several generations, genetic algorithm converges to the best individual, which hopefully represents an optimal or suboptimal solution to the problem. The general structure of the Genetic algorithms is as follow:

Begin
{
    t=0;
    Initialize P(t);
    Evaluate P(t);
    While (not termination condition) do
        Begin
            {  
                Apply crossover and mutation to P(t) to yield C(t);
                Evaluate C(t);
                Select P(t+1) from P(t) and C(t);
                t=t+1;
            }
        End
    }
End

Figure (1) Flowchart explains the general structure of the genetic algorithms.
OUR APPROACH

PARTICLE SWARM OPTIMIZATION

A concept for the optimization of nonlinear functions using particle swarm methodology is introduced. Particle swarm optimization has roots in two main component methodologies. Perhaps more obvious are its ties to artificial life (A-life) in general, and to bird flocking, fish schooling, and swarming theory in particular. It is also related, however, to evolutionary computation, and has ties to both genetic algorithms and evolutionary programming.

Particle swarm optimization as developed by the authors comprises a very simple concept, and paradigms can be implemented in a few lines of computer code. It requires only primitive mathematical operators, and is computationally inexpensive in terms of both memory requirements and speed.

Particle swarm optimization is an extremely wimple algorithm that seems to be effective for optimizing a wide range of functions. We view it as a mid-level form of A-life or biologically derived algorithm, occupying the space in nature between evolutionary search, which requires eons, and neural processing, which occurs on the order of milliseconds. Conceptually, it seems to lie somewhere between genetic algorithms and evolutionary programming.

2. MATERIALS AND METHODS

2.1. PROBLEM IDENTIFICATION

1) Local minima problem
Genetic algorithms (GA) are an optimization technique for searching very large spaces that models the role of the genetic material in living organisms. A small population of individual exemplars can effectively search a large space because they contain schemata, useful substructures that can be potentially combined to make fitter individuals. Formal studies of competing schemata show that the best policy for replicating them is to increase them exponentially according to their relative fitness. This turns out to be the policy used by genetic algorithms. Fitness is determined by examining a large number of individual fitness cases. But local minima problem exists in using genetic algorithm. Local minima problem is a point where the function value is smaller than at nearby points, but possibly greater than at a distant point in the search space. In genetic algorithm, the fitness value of one chromosome can attain minimum value with respect to the nearest chromosome but greater than at a distant chromosome. Therefore it will not be the efficient way to achieve the best fitness of the function.

2) Premature convergence
In genetic algorithms, we may often hear about premature convergence, that is a population for an optimization problem converged too early. Or we can say that case of loss of individual variation (every individual in the population is identical). But, how can it happen? One possible explanation may be the race between the innovation time and the takeover time,
originally proposed in "Toward a better understanding of mixing in genetic algorithms," by Goldberg, Deb, & Thierens (1993). According to David E. Goldberg's GA design decomposition theory, premature convergence happens because the innovation time is longer than the takeover time. Briefly, takeover time refers to "how long does it take for the best individual to take over the population (the force imposed by selection)" and innovation time refers to "how long does it take to create a new best individual (the force generated by the genetic operators)." Convergence is majorly about selection. If selection is not considered, theorizing the working mechanisms/principles regarding premature convergence may not be sound nor complete.

3) Older approach
Although genetic algorithm seems to scale much better as the problem size increases, this is now got older. Genetic algorithms do not scale well with complexity. That is, where the number of elements which are exposed to mutation is large there is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or plane. In order to make such problems tractable to evolutionary search, they must be broken down into the simplest representation possible. Hence we typically see evolutionary algorithms encoding designs for fan blades instead of engines, building shapes instead of detailed construction plans, airfoils instead of whole aircraft designs. The second problem of complexity is the issue of how to protect parts that have evolved to represent good solutions from further destructive mutation, particularly when their fitness assessment requires them to combine well with other parts.

4) Lack of multi constraint
GAs cannot effectively solve problems in which the only fitness measure is a single right/wrong measure (like decision problems), as there is no way to converge on the solution (no hill to climb). In these cases, a random search may find a solution as quickly as a GA. However, if the situation allows the success/failure trial to be repeated giving (possibly) different results, then the ratio of successes to failures provides a suitable fitness measure.

For specific optimization problems and problem instances, other optimization algorithms may be more efficient than genetic algorithms in terms of speed of convergence. Alternative and complementary algorithms include evolution strategies, evolutionary programming, simulated annealing, Gaussian adaptation, climbing, and swarm intelligence (e.g.: ant colony optimization, particle swarm optimization) and methods based on integer linear programming. The suitability of genetic algorithms is dependent on the amount of knowledge of the problem; well-known problems often have better, more specialized approaches.

2.2. METHODOLOGY

Particle swarm optimization is an extremely wimple algorithm that seems to be effective for optimizing a wide range of functions. We view it as a mid-level form of A-life or biologically derived algorithm, occupying the space in nature between evolutionary search, which requires
eons, and neural processing, which occurs on the order of milliseconds. Early testing has found the implementation to be effective with several kinds of problems. This paper discusses application of the algorithm to the training of artificial neural network weights; Particle swarm optimization has also been demonstrated to perform well on genetic algorithm test functions.

2.3. THE ETIOLOGY OF PARTICLE SWARM OPTIMIZATION

The particle swarm optimizer is probably presented by explaining its conceptual development. As mentioned above, the algorithm began as a simulation of a simplified social milieu. Agents were thought of as collision-proof birds, and the original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock.

Nearest Neighbor Velocity Matching and Craziness
A satisfying simulation was rather quickly written, which relied on two props: nearest-neighbor velocity matching and “craziness.” A population of birds was randomly initialized with a position for each on a torus pixel grid and with X and Y velocities. At each iteration a loop in the program determined, for each agent (a more appropriate term than bird), which other agent was its nearest neighbor, then assigned that agent’s X and Y velocities to the agent in focus.

Eliminating Ancillary Variables
Once it was clear that the paradigm could optimize simple, two-dimensional, linear functions, it was important to identify the parts of the paradigm that are necessary for the task. For instance, the authors quickly found that the algorithm works just as well, and looks just as realistic, without craziness, so it was removed. Next it was shown that optimization actually occurs slightly faster when nearest neighbour velocity matching is removed, though the visual effect is changed. The flock is now a swarm, but it is well able to find the codielidl.

The variables pbest and gbest and their increments are both necessary. Conceptually pbest resembles autobiographical memory, as each individual remembers its own experience (though only one fact about it), and the velocity adjustment associated with pbest has been called “simple nostalgia” in that the individual tends to return to the place that most satisfied it in the past. On the other hand, gbest is conceptually similar to publicized knowledge, or a group norm or standard, which individuals seek to attain. In the simulations, a high value of p-increment relative to g-increment results in excessive wandering of isolated individuals through the problem space, while the reverse (relatively high g-increment) results in the flock rushing prematurely toward local minima. Approximately equal values of the two increments seem to result in the most effective search of the problem domain.

Multidimensional Search
The algorithm seems too impressively”del a flock searching for a cornfield, most interesting optimization problems are neither linear nor two-dimensional. Since one of the authors’ objectives is to model social behavior, which is multidimensional and collision-free, it seemed a
simple step to change present\textit{x} and present\textit{y} (and of course \textit{vx}[\textit{j}] and \textit{vy}[\textit{n}] from one-dimensional arrays to D x N matrices, where \textit{D} is any number of dimensions and \textit{N} is the number of agents. Multidimensional experiments were performed, using a nonlinear, multidimensional problem: adjusting weights to train a feed forward multilayer perceptron neural network (NN). One of the authors’ first experiments involved training weights for a three-layer NN solving the exclusive-or (XOR) problem.

This problem requires two inputs and one output processing elements (PES), plus some number of hidden PES. Besides connections from the previous layer, the hidden and output PE layers each has a bias PE associated with it. Thus a 2,3,1 NN requires optimization of 13 parameters. This problem was approached by flying the agents through 13-dimensional space until an average sum-squared error per PE criterion was met. The algorithm performed very well on this problem. The thirteen dimensional XOR network was trained, to am e < 0.05 criterion, in an average of 30.7 iterations with 20 agents.

\textbf{Acceleration by Distance}

Though the algorithm worked well, there is something aesthetically displeasing and hard to understand about it. Velocity adjustments were based on a crude inequality test: if \textit{present}\textit{x} > \textit{best}\textit{x}, make it smaller; if \textit{present}\textit{x} < \textit{best}\textit{x}, make it bigger. Some experimentation revealed that further revising the algorithm made it easier to understand and improved its performance. Rather than simply testing the sign of the inequality, velocities were adjusted according to their difference, per dimension, from best locations:

1. \textit{Vid} = \textit{Vid} + \textit{C1r1}(\textit{Pbest} - \textit{Gbest}) + \textit{C2r2}(\textit{Pbest} - \textit{ldest})
2. \textit{Xid} = \textit{Xid} + \textit{Vid}
3. RESULTS AND DISCUSSIONS

We perform above algorithm by setting desired parameters and this yield the randomly generated quantities. By performing the fitness evaluation repeatedly, the global best value will be stored. Further particle \( p_{best} \) value will be compared with global \( g_{best} \) value until the best possible value is achieved. This performance is compared with two different approaches, first is tile approach which is shown in Table I and the other is via Satisfiability Table II. By comparing both approaches, PSO based approach giving better results as shown in the figures below.

### Table I

**Comparison with Tile Approach [7]**

<table>
<thead>
<tr>
<th>Circuit</th>
<th>CMOL FPGA CAD tool 10 ( (r = 12) )</th>
<th>Our work ( (r = 12) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cells</td>
<td>Area (Tiles)</td>
</tr>
<tr>
<td>s27</td>
<td>12</td>
<td>64(2 x 2)</td>
</tr>
<tr>
<td>s208</td>
<td>123</td>
<td>256(4 x 4)</td>
</tr>
<tr>
<td>s298</td>
<td>125</td>
<td>256(4 x 4)</td>
</tr>
<tr>
<td>s344</td>
<td>174</td>
<td>400(5 x 5)</td>
</tr>
<tr>
<td>s349</td>
<td>186</td>
<td>400(5 x 5)</td>
</tr>
<tr>
<td>s382</td>
<td>175</td>
<td>400(5 x 5)</td>
</tr>
<tr>
<td>s386</td>
<td>219</td>
<td>400(5 x 5)</td>
</tr>
<tr>
<td>s400</td>
<td>189</td>
<td>400(5 x 5)</td>
</tr>
<tr>
<td>s420</td>
<td>200</td>
<td>400(5 x 5)</td>
</tr>
<tr>
<td>s444</td>
<td>210</td>
<td>400(5 x 5)</td>
</tr>
<tr>
<td>s510</td>
<td>290</td>
<td>576(6 x 6)</td>
</tr>
<tr>
<td>s526</td>
<td>329</td>
<td>576(6 x 6)</td>
</tr>
<tr>
<td>s613</td>
<td>329</td>
<td>576(6 x 6)</td>
</tr>
<tr>
<td>s713</td>
<td>377</td>
<td>576(6 x 6)</td>
</tr>
<tr>
<td>s820</td>
<td>290</td>
<td>576(6 x 6)</td>
</tr>
<tr>
<td>s832</td>
<td>329</td>
<td>576(6 x 6)</td>
</tr>
<tr>
<td>s838</td>
<td>377</td>
<td>576(6 x 6)</td>
</tr>
<tr>
<td>s1196</td>
<td>329</td>
<td>576(6 x 6)</td>
</tr>
<tr>
<td>s1388</td>
<td>377</td>
<td>576(6 x 6)</td>
</tr>
</tbody>
</table>

### Table II

**Comparison with SAT approach [8]**

<table>
<thead>
<tr>
<th>SAT approach ( (r = 9) )</th>
<th>Our work ( (r = 9) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells</td>
<td>Delay</td>
</tr>
<tr>
<td>s27</td>
<td>8</td>
</tr>
<tr>
<td>s208</td>
<td>109</td>
</tr>
<tr>
<td>s298</td>
<td>85</td>
</tr>
<tr>
<td>s344</td>
<td>130</td>
</tr>
<tr>
<td>s349</td>
<td>134</td>
</tr>
<tr>
<td>s382</td>
<td>124</td>
</tr>
<tr>
<td>s386</td>
<td>138</td>
</tr>
<tr>
<td>s400</td>
<td>137</td>
</tr>
<tr>
<td>s420</td>
<td>248</td>
</tr>
<tr>
<td>s444</td>
<td>136</td>
</tr>
<tr>
<td>s510</td>
<td>266</td>
</tr>
<tr>
<td>s526</td>
<td>222</td>
</tr>
<tr>
<td>s614</td>
<td>206</td>
</tr>
<tr>
<td>s713</td>
<td>225</td>
</tr>
<tr>
<td>s820</td>
<td>400</td>
</tr>
<tr>
<td>s832</td>
<td>407</td>
</tr>
<tr>
<td>s838</td>
<td>507</td>
</tr>
<tr>
<td>s1196</td>
<td>613</td>
</tr>
<tr>
<td>s1388</td>
<td>662</td>
</tr>
</tbody>
</table>
4. CONCLUSIONS & RECOMMENDATIONS

Particle swarm optimization is similar to a genetic algorithm in that the system is initialized with a population of random solutions. It is unlike a genetic algorithm, however, in that each potential solution is also assigned a randomized velocity, and the potential solutions, called particles, are then “flown” through hyperspace.

Each particle keeps track of its coordinates in hyperspace which are associated with the best solution (fitness) it has achieved so far. (The value of that fitness is also stored.) This value is called pbest. Another “best” value is also tracked. The “global” version of the particle swarm optimizer keeps track of the overall best value, and its location, obtained thus far by any particle in the population; this is called gbest.

The particle swarm optimization concept consists of, at each time step, changing the velocity (accelerating) each particle toward its pbest and gbest (global version). Acceleration is weighted by a random term, with separate random numbers being generated for acceleration toward pbest and gbest.

The whole analysis and implementation of approach is depending on the following outcome:

1. Increasing Area Utilization.
2. Reducing CPU time.
3. Reducing the number of CMOL cells.
4. Reducing the Area.

5. ACKNOWLEDGEMENTS

Useful discussions with Prof. Anil Kumar Sahu and Electronics department are gratefully acknowledged. The work has been supported by the International Journal of Research-Granthaalayah (IJRG).

6. REFERENCES


