Swarm intelligence versus direct cover algorithms in synthesis of Multi-Valued Logic functions

Mostafa Abd-El-Barr and Kalim Qureshi
Department of Information Science, College of Life Sciences, Kuwait University, Kuwait City, Kuwait, and
Bambang Sarif
Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada

Abstract
Ant Colony Optimization and Particle Swarm Optimization represent two widely used Swarm Intelligence (SI) optimization techniques. Information processing using Multiple-Valued Logic (MVL) is carried out using more than two discrete logic levels. In this paper, we compare two the SI-based algorithms in synthesizing MVL functions. A benchmark consisting of 50,000 randomly generated 2-variable 4-valued functions is used for assessing the performance of the algorithms using the benchmark. Simulation results show that the PSO outperforms the ACO technique in terms of the average number of product terms (PTs) needed. We also compare the results obtained using both ACO-MVL and PSO-MVL with those obtained using Espresso-MV logic minimizer. It is shown that on average, both of the SI-based techniques produced better results compared to those produced by Espresso-MV. We show that the SI-based techniques outperform the conventional direct-cover (DC) techniques in terms of the average number of product terms required.

Keywords Ant Colony (ACO), Particle Swarm Optimization (PSO), Direct Cover Algorithm (DC), Espresso-MVL, Multi-Valued Logic (MVL), MVL Function Synthesis, Non-Binary Digital Information Processing

1. Introduction
It is widely recognized by researchers as well as the chip industry that on-chip complex binary systems exhibit a number of curbs, such as large layout area for interconnections, limitations of data storage, increased power consumption, and limitation of available bandwidth. These limitations can be mostly overcome by using Multiple-Valued Logic (MVL) systems [1–3]. Digital information processing using MVL is carried out using more than two
discrete logic levels. MVL-based information processing include arithmetic operations/
processors, memory, and special purpose processors [1]. Reported MVL (sub)-systems have
shown considerable reduction both in time and speed compared to their binary counterparts
[2–5]. It should however be noted that the search space for MVL synthesis problem is huge
when compared to the binary one. There are \( p^{r_{(n)}} = 4^{2^2} = 2^{32} \) 2-variable 4-valued functions as compared to 16 2-variable binary functions. This makes exact minimization of MVL
functions prohibitively expensive both in time and complexity. Therefore, modern heuristic
optimization techniques were employed. These algorithms can be classified into iterative
based [6], decomposition-based [7], algebraic based [8], direct cover-based [9–11], and
evolutionary-based algorithms. The latter include Genetic algorithms (GAs) [12–13], Ant
Colony (ACO) [14–15], and PSO-based [16]. Fuzzy-based synthesis of MVL functions has also
been reported in the literature [21].

The Direct Cover (DC) algorithm is an iterative heuristic in the synthesis of MVL
functions [9–11]. Swarm intelligence models (SIMs) are computational models inspired by
real life swarm systems. The two main SIMs are the Ant Colony Optimization (ACO) [25]
and the Particle Swarm Optimization (PSO) [26]. The main idea in the ACO is to model a
given problem as a search conducted by ants for the optimal path guided by a substance
deposited on the path called pheromone. Real ants were found to be skillful in finding the
shortest path between their nests and a food source in the presence of hurdles. Particle
swarm optimization (PSO) is inspired by the observation that birds fly in large groups
and for long distances without collision. It is hypothesized that birds are able to achieve
that through experience-sharing by maintaining an optimum distance between
themselves and their neighbors. Hence, PSO is a search strategy that uses a set of
flying particles with velocities that are dynamically adjusted based on past experience of
individual particle and that of their neighbors in the search space. The PSO algorithm
was first introduced in 1995 [21] and was later on extended as in [22,23]. There has been
growing interest in the use of PSO for optimization problems [25,28,29,35–40,44–46] and
in synthesis of circuits [26,27]. A number of recent research work, books, and book
chapters on the issue of swarm and nature-inspired algorithms have been published in
the literature [47–49]. However, very few articles were reported on the synthesis of MVL
functions, see for example [16].

The paper provides a concise coverage and comparison of both the conventional synthesis
techniques of MVL functions represented by the Direct Cover algorithm and those based on
the emerging functional synthesis techniques using the swarm intelligent-based algorithms.
We aim to provide a useful data analytics for researchers in the area of digital synthesis for
high-radix (beyond binary) logic functions. The paper also illustrates the adaptation of the
discrete PSO algorithms in the area of MVL functional synthesis while showing the adopted
processes used in the selection of the appropriate minterms and the appropriate implicants
(product terms) to cover them. We also provide a comparison among the ACO and the PSO on
one hand and the Espresso-MV standard algorithm on the other hand.

It should be noted that the two swarm intelligence based algorithms ACO and the PSO are
selected in this work because of them being among the well-known swarm intelligence
techniques and also in order to align our work with similar research work identified in the
literature, e.g. [50].

The paper is organized as follows. Section 2 provides some background material. In
Section 3 we provide a coverage of the related work published in the literature. Section 4
covers simulation results conducted using the ACO and the PSO algorithms using a
benchmark consisting of 50,000 4-valued 2-varaible functions. In Section 5 we provide
comparison among the results obtained out of the simulation conducted in Section 4. In
Section 6 we provide a number of concluding remarks.
2. Background material

This section provides enough of the foundation material needed to make the paper self-stand and to help the reader to follow through the paper easily and with the minimum interruption.

Table 1 shows the MVL logic operators used in this paper [15].

<table>
<thead>
<tr>
<th>The operator</th>
<th>The logic equation</th>
</tr>
</thead>
</table>
| The window literal | $a \cdot x$ = \begin{cases} (r-1) & \text{if} (a \leq x \leq b) \\ 0 & \text{otherwise} \end{cases} 
where $a, b \in R$ and $a \leq b$. It should be noted that $r = 4$ for 4-valued logic. |
| The truncated sum (tsum) | $tsum(a_1, a_2, \ldots, a_n) = a_1 \oplus_a a_2 \oplus \ldots \oplus_a a_n = \begin{cases} a_1 + a_2 + \ldots + a_n & \text{if} \ a_1 + a_2 + \ldots + a_n < r - 1 \\ r - 1 & \text{otherwise} \end{cases}$ 
where $a_i \in R$ and $\oplus_a$ represents the truncated sum operation. |
| The maximum (MAX) of two MVL variables | $\text{MAX}(x_1, x_2) = \begin{cases} x_1 & \text{if} \ x_1 \geq x_2 \\ x_2 & \text{otherwise} \end{cases}$ |
| The minimum (MIN) of two MVL variables | $\text{MIN}(x_1, x_2) = \begin{cases} x_1 & \text{if} \ x_1 \leq x_2 \\ x_2 & \text{otherwise} \end{cases}$ |

**Example:** The product terms in Figure 1 include $1, 1 \cdot 2 \cdot x_1^0 \cdot x_2^1, 2 \cdot 0 \cdot x_1^0 \cdot x_2^3$ and $3 \cdot 1 \cdot x_1^1 \cdot x_2^2$. In the first case $c = 1$, $a_1 = 2$, $b_1 = 0$, $a_2 = 1$, $b_2 = 2$. In the second case $c = 2$, $a_1 = 0$, $b_1 = 1$, $a_2 = 2$, $b_2 = 0$. In the third case $c = 3$, $a_1 = 1$, $b_1 = 2$, $a_2 = 1$. The objective of MVL synthesis is to find the minimum form of an MVL function that can be included in a product terms in $\text{MIN}$.

**Definition 1** [15]: Define an n-variable r-valued function, $f(X)$, as a mapping $f : R^n \rightarrow R$. In the above definition $R = \{0, 1, \ldots, r-1\}$ is a set of $r$ logic values with $r \geq 2$ and $X = \{x_1, x_2, \ldots, x_n\}$ is a set of r-valued $n$ variables. An example 4-valued ($r = 4$), 2-variable ($n = 2$) function is shown in Figure 1.

**Definition 2** [15]: Define a product term (PT), $P(x_1, x_2, \ldots, x_n)$, as the minimum of a set of window literals such that $P(x_1, x_2, \ldots, x_n) = C \cdot a_1 x_1^{b_1} \cdot a_2 x_2^{b_2} \cdot a_n x_n^{b_n} = \min(C, a_1 x_1^{b_1}, a_2 x_2^{b_2}, \ldots, a_n x_n^{b_n})$, where $a_i, b_i \in R$, $a_i \leq b_i$ and $c \in \{1, 2, \ldots, r-1\}$ is the value of the product term.

**Definition 3** [15]: Define a minterm as the assignment $x_1 = a_1, x_2 = a_2, \ldots, x_n = a_n$, in an MVL function $f(x_1, x_2, \ldots, x_n)$ if and only if $f(x_1, x_2, \ldots, x_n) \neq 0$. In the above definition $a_i \in \{0, 1, \ldots, r-1\}$. If the value of a minterm is $r$, then it is considered as don’t care and is represented as $d$. There are 12 minterms in the function shown in Figure 1.

**Definition 4** [15]: A PT, $I(x_1, x_2, \ldots, x_n)$ of a function $f(x_1, x_2, \ldots, x_n)$ is called an implican if $f(x_1, x_2, \ldots, x_n) \geq I(x_1, x_2, \ldots, x_n)$ for all assignments of $x_i$'s.

**Example:** Figure 1 shows an example 4-valued 2-variable function.

The objective of MVL synthesis is to find the minimum form of an MVL function that can be completely or incompletely specified. As indicated above exact minimization of MVL functions is prohibitively expensive both in time and complexity and that a number of modern heuristic optimization techniques were employed. These include the direct cover-based (DC) [9–11], the Ant Colony (ACO) [14,15], and PSO-based [16]. In Section 5 we provide a comparison of the performance of the ACO and the PSO in the context of synthesis of MVL.
functions. However the following table (adapted from [50]) illustrates the fundamental difference between the two models of computation.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>ACO</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Indirect communication through the environment (Stigmergy).</td>
<td>Direct communication.</td>
</tr>
<tr>
<td>Mechanism</td>
<td>Both discrete (combinatorial) and continuous optimization problems.</td>
<td>Both Continuous and discrete (combinatorial) optimization problems.</td>
</tr>
<tr>
<td>Problem Type</td>
<td>Using weighted (construction) graph.</td>
<td>Set of n-dimensional points.</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Problems with predefined source and destination</td>
<td>Problems where previous and next particle positions at each point are uniquely defined.</td>
</tr>
<tr>
<td>Applicability</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Having covered the needed background material, we provide a coverage of the DC, ACO, and the PSO algorithms in Section 3.

3. Related work

3.1 The Direct cover heuristic algorithm for synthesis of MVL functions

The Direct Cover (DC) algorithm is an iterative heuristic in the synthesis of MVL functions [9–11]. Figure 2 shows an adapted version of the DC.

A detailed analysis of the DC-based algorithms has been conducted jointly by the author in [17]. The results of such analysis have indicated that improvements to the conventional DC are still possible [20] and [24]. In the next section, we present the SI-based improvements of the DC in the form of the Ant Colony (ACO) and the Particle Swarm Optimization (POS).

3.2 The ACO algorithm for synthesis of MVL functions

The main idea in the ACO is to model a given problem as a search conducted by ants for the optimal path guided by a substance deposited on the path called pheromone. Real ants were found to be skillful in finding the shortest path between their nests and a food source in the presence of hurdles. The search is made possible by an indirect communication amongst the ants. While traveling their way, ants deposit a chemical substance, called pheromone, on
the ground. When they arrive at a decision point, they make a probabilistic choice, biased by the intensity of pheromone they smell, see Figure 3(a). This behavior has an autocatalytic effect since choosing a given path increases the likelihood that the same path will be chosen again by future ants. On their way back, ants are expected to choose the same path with high probability (due to the increase in the amount of pheromone), see Figure 3(b). As further pheromone is released, the chosen path will become more attractive for future ants. See Figure 3(c) [34].

The basic algorithmic steps in an ACO are shown in Figure 4.

A key point in the development of ACO is the selection of an appropriate fitness function. The fitness function is often formulated as cost minimization of the solution of the given problem. The ACO algorithm has received much attention and has been incorporated in a number of optimization problems [16–21,30–33,43].

3.3 The PSO algorithm for synthesis of MVL functions

Particle swarm optimization (PSO) is inspired by the observation that birds fly in large groups and for long distances without collision. It is hypothesized that birds are able to achieve that through experience-sharing by maintaining an optimum distance between themselves and their neighbors. Hence, PSO is a search strategy that uses a set of flying particles with velocities that are dynamically adjusted based on past experience of individual particle and that of their neighbors in the search space. The PSO algorithm was first introduced in 1995 [21] and was later on extended as in [22,23]. There has been growing interest in the use of PSO for optimization problems [25,28,29,35–40] and in synthesis of circuits [26,27]. However, very few articles were reported on the synthesis of MVL functions, see for example [16]. The basic algorithmic steps in a PSO are shown in Figure 5.
4. Simulation of the algorithms

In this section we present a coverage of the simulation process conducted using the ACO and the PSO algorithms introduced in Section 3 above. The simulation conducted among the algorithms is made using a set of a benchmark consisting of 50,000 randomly generated 2-variable 4-valued functions. The main characteristics of the generated benchmark are illustrated below. As can be seen the number of minterms in the benchmark ranges from 6 to 16. The number of functions in each range is illustrated in the table below. For example there are 500 functions each has 16 minterms, 9000 function search has 11 minterms, and 75 functions each has 6 minterms.

<table>
<thead>
<tr>
<th># functions</th>
<th>500</th>
<th>2700</th>
<th>6600</th>
<th>1050</th>
<th>1130</th>
<th>9000</th>
<th>5450</th>
<th>2500</th>
<th>1100</th>
<th>275</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td># minterms</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

4.1 Ant Colony synthesis technique (ACO-MVL)

In the ACO-MVL, use is made of the ants searching strategy in finding the best minterms cover using the right set of implicants. The minterms coverage process is performed iteratively, until all minterms in the table is covered. Upon selecting a minterm (or an implicant) an ant puts some pheromone trails on it. This will cause the next ant performs selection based on the additional pheromone information. This positive feedback process drives the selection to the best solution (see Figure 6) [15].

The daemon_action( ) function is used to initialize the pheromone value at the beginning of all ant’s movement and also whenever stagnancy in the solution is found. Based on the output of the fitness calculations the selected minterms and implicants are updated. The next generation of ants will then use the newly updated pheromone values to guide their selection.

---

**Algorithm ACO-MVL (adapted from [15])**

1. Set the number of runs
2. Set the number of ants
3. Repeat for the set number of runs
   3.1. Repeat for the set number of ants
      (a) daemon_action();
      (b) ant[a].L = {};
      (c) Repeat while (checkTable()) do {
         c.1. M = selectMintermACO();
         c.2. L = selectImplicantACO(M);
         c.3. ant[a].L $\leftarrow$ ant[a].L + L;
      }
      (d) calculate_fitness(ant[a]);
   3.2. pheromone_update_minterm();
   3.3. pheromone_update_implicant();

---

**PSO Algorithmic Steps (adapted from [39])**

1. Initializing Swarm’s velocity and position in each dimension of the solution space.
2. Repeat until convergence or meeting termination conditions.
   (a) Evaluating the fitness function for each particle’s position.
   (b) Updating the best position for each particle based on past and current position
   (c) Updating the Swarm’s globally best particle based on fitness function
   (d) Updating all particles’ velocities
   (e) Moving each particle to its new position

---

**Figure 5.**
PSO basic algorithmic steps.

**Figure 6.**
ACO-MVL algorithm.
4.1.1 Solution representation. According to the ACO-MVL, a tabular form is used to represent a given MVL function. A bag whose size is equal to the size of the function’s truth table is used to store all selected implicants. Each implicant is represented by a string of integers consisting five integer attributes as follows (in sequence): the constant value of the implicant, the first and second value of the literal (a and b in Table 1) of the first variable in the implicant representation, and followed by the first and second value of the literal (a and b in Table 1) of the second variable in the implicant representation. For example, if the implicant is \(1 \times x_1^{12} x_2^{13}\), then the five integer attributes would be 1 1 2 1 3.

It should be noted that performance of the ACO algorithm is dependent on the selection of a number of parameters including the expectation heuristic factor, the pheromone evaporation rate, and the number of ants. The ACO parameter selection techniques differ from a study to the other depending on the types of optimization and the scales of the problem considered. In our case we have conducted experimentations on the parameter value selection using the (expectation) heuristic value \(\eta_M\) and the pheromone evaporation \(\tau_M\) and we have developed a measure of the probability of choosing a minterm \(M\) as explained below in Eq. (1). We have also adopted a similar approach in the selection of the parameters for the implicant (product term) and have developed to measure the probability for choosing an implicant, \(L\) as explained below in Eq. (2). Our approach in developing the measures below is inspired by the approach used in [51].

4.1.2 Minterm selection. A stochastic process of Roulette Wheel is used in the selection of a minterm. Two factors affect the probability of choosing a minterm. These are the pheromone value and a heuristic value. The first factor is obtained from pheromone trails of previous ants. The second factor is the fuzzy weighted averaging of minterm’s selection criteria. The probability of choosing minterm \(M\) is computed as:

\[
p = \frac{\tau_M \cdot \eta_M}{\sum \tau_M \cdot \eta_M}
\]  

where \(\tau_M\) represents the pheromone value of the minterm \(M\) and \(\eta_M\) represents the heuristic value of minterm \(M\).

4.1.3 Implicant selection. A stochastic process of Roulette Wheel is used in the selection of an implicant. Two factors affect the probability of choosing an implicant. These are the pheromone value and a heuristic value. The first factor is obtained from pheromone trails of previous ants. The second factor is the fuzzy weighted averaging of implicants’ selection criteria. The probability of choosing implicant \(L\) is computed as:

\[
p = \frac{\tau_L \cdot \eta_L}{\sum \tau_L \cdot \eta_L}
\]  

where \(\tau_L\) represents the pheromone value of implicant \(L\) and \(\eta_L\) represents the heuristic value of literal \(L\) (fuzzy weighted averaging of literal’s \(L\) selection criteria).

4.1.4 Fitness function. The Functional fitness, \(F_f = N_h - N_m\), where \(N_h\) = the number of minterm matching between the two truth tables and \(N_m\) = the number of mismatch between the two truth tables. The Objective Fitness, \(F_o = (100 - N_p) / 100\), where \(N_p\) is the number of product terms needed to cover the function. A chromosome with the highest functional fitness is considered as the best solution. In case of a tie, the chromosome with the highest objective functional is selected.

4.1.5 Numerical results. The ACO-MVL algorithm was applied to a benchmark consisting of 50,000 randomly generated 2-variable 4-valued functions. The following parameters were used:

- Number of populations (\(NP\)) = 20
Maximum number of Iterations ($\text{MI}$) = 100

Maximum number of Runs = ($\text{MR}$) 10

Pheromone evaporation rate ($\text{RHO}$) = 0.95

The following two measures were recorded:

1. $\text{SR}_m \equiv \text{Successful Rate}$ (number of successful iterations divided by the total number of runs)

2. $\text{PT} \equiv \text{Number of Product Terms}$.

Table 2 shows the average number of Product Terms (#PT) needed with respect to the number of minterms of the MVL function used in the benchmark (functions with minterms ranging from 6 to 16 are used) as well as for the conventional DC-based algorithms (ARM [11], BS [9], and DM [10]).

The results in the table show the ACO-MVL algorithm outperforms the DC-based algorithms for all cases. The table also shows that the maximum percentage improvement and the minimum percentage improvement achieved by the ACO-MVL are 12.8% and 0.3%, respectively. It should be noted that the achieved improvements should be considered in the context that the simulated benchmark is representing a population of $4^{16}$ functions. Figure 7 illustrates this comparison. An overall comparison between the SI-based techniques and the espresso-MV logic minimizer tools is provided in Section 3.

4.2 The particle swarm synthesis technique (PSO-MVL)

We motivate the discussion in this section by considering the function given in Figure 8.

Notice that based on the representation introduced in [13], the MVL function shown in Figure 8 is represented as: '0000000023102200. The coverage relation between the minterms of the function and the implicants is depicted in Figure 9.

4.2.1 Particle representation. A string consisting of five numbers is used to represent an implicant [13]: the first number represents the value of the implicant, the next two numbers represent the range of the literal of the first variable (a and b in Table 1) while the last two numbers represent the range of the literal of the second variable (a and b in Table 1). Consider, for example, the 4-valued function given in Figure 9. The two implicants (fully or partially) that cover the minterm 31,122 are 11,222 and 20,123 (see Figure 10).

It should be that the performance of the PSO algorithm is dependent on a number of parameters including the swarm size, the particles velocities, the cognitive coefficient, and the particle positions. In our case, we have used the same list of parameters except that for the particle velocities. In our extermination and due to the discrete nature of the MVL synthesis problem we have adopted the discrete PSO approach introduced in [22,23]. Accordingly, we introduce the term particle displacement (rather than particle velocity) and used it instead. Please refer to the set of Eq. (4) in the paper. We have presented the obtained results due to the effect of the swarm size and the number of iterations in Table 4 in the paper. In our test we used: $c_1 = 1, c_2 = c_3 = 0.5$ and the number of runs $= 10$. Notice that $c_1$ and $c_2$ represent the cognitive coefficients. As a result of our experimentations, we found out that most appropriate value for $V_{\text{max}}$ is either 4 or 5. A value of $V_{\text{max}} = 5$ has been used in testing the PSO-MVL Algorithm. Our developed model to compute the particle displacement is shown below in Eq. (4).

4.2.2 Particle fitness function. The Functional fitness is computed as $F_f = N_h - N_m$, where $N_h$ is the number of minterm match and $N_m$ is the number of minterms mismatch between the truth table of the original and the obtained functions. The objective fitness is computed as $F_o = (100 - N_p)/100$, where $N_p$ is the number of product terms needed to cover the function. The selection of the best obtained solution is done on two levels: First the highest functional fitness function and in case of a tie the value of objective fitness will used.
<table>
<thead>
<tr>
<th># minterms</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td># functions</td>
<td>500</td>
<td>2700</td>
<td>6600</td>
<td>1050</td>
<td>1130</td>
<td>9000</td>
<td>5450</td>
<td>2500</td>
<td>1100</td>
<td>275</td>
<td>75</td>
</tr>
<tr>
<td>#PT (ARM) [11]</td>
<td>7.59</td>
<td>8.20</td>
<td>8.36</td>
<td>8.28</td>
<td>8.05</td>
<td>7.71</td>
<td>7.32</td>
<td>6.87</td>
<td>6.31</td>
<td>5.73</td>
<td>5.13</td>
</tr>
<tr>
<td>#PT (BS) [9]</td>
<td>7.56</td>
<td>8.31</td>
<td>8.41</td>
<td>8.35</td>
<td>8.10</td>
<td>7.76</td>
<td>7.37</td>
<td>6.88</td>
<td>6.32</td>
<td>5.75</td>
<td>5.15</td>
</tr>
<tr>
<td>#PT (DM) [10]</td>
<td>7</td>
<td>7.51</td>
<td>7.57</td>
<td>7.54</td>
<td>7.38</td>
<td>7.13</td>
<td>6.83</td>
<td>6.47</td>
<td>6.02</td>
<td>5.53</td>
<td>4.97</td>
</tr>
<tr>
<td>#PT (ACO-MVL) [14]</td>
<td>6.73</td>
<td>7.05</td>
<td>7.16</td>
<td>7.18</td>
<td>7.09</td>
<td>6.9</td>
<td>6.66</td>
<td>6.36</td>
<td>5.93</td>
<td>5.48</td>
<td>4.96</td>
</tr>
<tr>
<td>Max improvement (%)</td>
<td>12.8</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
</tr>
<tr>
<td>Min improvement (%)</td>
<td>4</td>
<td>6.5</td>
<td>5.7</td>
<td>5</td>
<td>4.2</td>
<td>3.3</td>
<td>2.6</td>
<td>1.8</td>
<td>1.5</td>
<td>0.9</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Table 2. Average #PTs needed for realization of MVL functions of the benchmark.
Example: Figure 10 shows an example of three particles in the swarm whose elements at time $t$.

Table 3 illustrates the computation of the particle fitness function as explained above. The results displayed in Table 3 indicate that particles 2 and 3 supersede particle 1 in representing the example function. Although the functional fitness of the two particles show a tie (a score of 16 in each) however the overall fitness shows superiority of particle 3 due to the weight added by the objective function. Therefore, particle 3 is considered the best possible solution for the given MVL function.

4.2.3 Particle movement. The discrete nature of the MVL synthesis problem necessitates adopting the discrete PSO approach introduced in [22,23]. Accordingly, we introduce the term particle displacement (rather than particle velocity) and introduce the following to compute the particle displacement ($D_{t+1,i}$):

![Figure 7. Average #PTs using ACO-MVL versus those achieved using DC-based technique.](image)

![Figure 8. Example MVL function.](image)

![Figure 9. Coverage relation between minterms and implicants for function shown in Figure 8.](image)

<table>
<thead>
<tr>
<th>Minterm</th>
<th>Implicant $1.x_1^2.x_2^2$</th>
<th>Implicant $2.x_1^1.x_2^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>Full</td>
<td>Partial</td>
</tr>
<tr>
<td>1. $x_1^2.x_2^2$</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>2. $x_1^0.x_2^2$</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>2. $x_1^0.x_2^3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. $x_1^1.x_2^3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. $x_1^1.x_2^2$</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
In the above computation we use \( D_{St,i} \) to represent the displacement of particle \( i \) at time step \( t \) due to particle best self-experience and \( DG_{Gt,i} \) to represent the displacement of particle \( i \) at time step \( t \) due to the global experience. We use \( X_{t,i} \) to represent the position of particle \( i \) at time step \( t \), \( P_{t,i} \) to represent the current best position at time step \( t \), and \( P_{Gt,i} \) to represent the current global best position at time step \( t \), while \( c_1, c_2, c_3 \) represent the social/cognitive confidence coefficients, and \( r_2, r_3 \) are two random numbers.

As can be seen, the particle displacement \( (D_{t+1,i}) \) consists of three components: \( (D_{St,i}), (DS_{t,i}), \) and \( (DG_{Gt,i}) \). The new particle position is formulated as follows:

\[
X_{t+1,i} = (X_{t,i} + D_{St,i}) + DG_{Gt,i} + c_1 D_{t,i}.
\]  

The PSO algorithm is illustrated in Figure 11 [16].
4.2.4 Numerical results. We have tested the proposed PSO-MVL algorithm using a benchmark consisting of 50,000 randomly generated 2-variable 4-valued functions. The results obtained are reported in Table 4.

We report in this table the average number of product terms (PTs) needed to cover a given function. We report the obtained results for different number of particles and different number of iterations. In our test we used: \( c_1 = 1 \), \( c_2 = c_3 = 0.5 \) and the number of runs = 10. As a result of our experimentations, we found out that most appropriate value for \( V_{\text{max}} \) is either 4 or 5. A value of \( V_{\text{max}} = 5 \) has been used in testing the PSO-MVL Algorithm. It should be noted that a gradual decrease in the number of product terms (PTs) needed to synthesize a given function occurs as we increase the number of particles and/or iterations used.

4.2.4.1 An observation that lead to a modification in the PSO-MVL. Two example 4-valued 2-variable functions are shown in Figure 12. Notice that F2 is the complement of F1. While F1 requires at least six implicants to represent, F2 requires only four implicants (circled in the figure). If we synthesize F2 and use an extra inverter we will need a total of 5 gates. This is still less than synthesizing F1. We have made use of such cases by modifying PSO-MVL algorithm in a way to synthesize both the given MVL function and its complement and collect the synthesis which requires less number of implicants taking into consideration the extra inverter gate needed if the complement of the function is synthesized.

Table 5 shows the results obtained using the modified PSO-MVL in comparison with the DC-based conventional heuristics for different functions having different number of minterms. In this table we present the average number of minterms required to synthesize a benchmark consisting of 50,000 2-variable 4-valued functions having minterms ranging from 6 to 16. For a comparison purpose, we also report the results obtained for three other synthesis algorithms: the ARM [11], the BS [9], and the DM [10]. The maximum and minimum percentage improvement for each of the 11 categories of functions is also computed and reported in the table. The maximum improvement achieved by the modified PSO-MVL is 31.2% (in the case of functions having 11 minterms) and the minimum improvement achieved is 7.9% (in the case of functions having 6 minterms). The results shown in Table 5 is illustrated in Figure 13.
Table 5. Average PT with Respect to Different Number of Minterms of MVL Functions.

<table>
<thead>
<tr>
<th># minterms</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td># functions</td>
<td>500</td>
<td>2700</td>
<td>6600</td>
<td>10,500</td>
<td>11,300</td>
<td>9000</td>
<td>5450</td>
<td>2500</td>
<td>1100</td>
<td>275</td>
<td>75</td>
</tr>
<tr>
<td>#PT (ARM) [11]</td>
<td>7.59</td>
<td>8.30</td>
<td>8.36</td>
<td>8.28</td>
<td>8.05</td>
<td>7.71</td>
<td>7.32</td>
<td>6.87</td>
<td>6.31</td>
<td>5.73</td>
<td>5.13</td>
</tr>
<tr>
<td>#PT (BS) [9]</td>
<td>7.56</td>
<td>8.31</td>
<td>8.41</td>
<td>8.35</td>
<td>8.1</td>
<td>7.76</td>
<td>7.37</td>
<td>6.88</td>
<td>6.32</td>
<td>5.75</td>
<td>5.15</td>
</tr>
<tr>
<td>#PT (DM) [10]</td>
<td>7</td>
<td>7.51</td>
<td>7.57</td>
<td>7.54</td>
<td>7.38</td>
<td>7.13</td>
<td>6.83</td>
<td>6.47</td>
<td>6.02</td>
<td>5.53</td>
<td>4.97</td>
</tr>
<tr>
<td>#PT (PSO-MVL) [15]</td>
<td>5.79</td>
<td>6.86</td>
<td>6.93</td>
<td>6.89</td>
<td>6.76</td>
<td>6.57</td>
<td>6.33</td>
<td>6.06</td>
<td>5.70</td>
<td>5.27</td>
<td>4.73</td>
</tr>
<tr>
<td>Max improvement (%)</td>
<td>31.2</td>
<td>30.7</td>
<td>30.7</td>
<td>30.7</td>
<td>30.7</td>
<td><strong>31.2</strong></td>
<td>30.7</td>
<td>30.7</td>
<td>30.7</td>
<td>30.7</td>
<td>30.7</td>
</tr>
<tr>
<td>Min improvement (%)</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>9.5</td>
<td>9.3</td>
<td>9.5</td>
<td>9.2</td>
<td>8.5</td>
<td><strong>7.9</strong></td>
</tr>
</tbody>
</table>
5. A comparison

In this section we provide two main comparisons. The first is among the two Swarm Intelligence based algorithms used for synthesis of MVL functions, i.e. the ACO-MVL and the PSO-MVL. The second comparison is among the PSO-MVL and ACO-MVL on one side with the results obtained using espresso-MVL logic minimizer. Our basis for comparison is the benchmark consisting of 50,000 2-variable 4-valued randomly generated function. Table 6 shows a summary of the results obtained for the results obtained for PSO-MVL and those obtained for ACO-MVL for synthesizing the 50,000 benchmark functions.

The results shown in Table 6 indicate that on average the number of product terms (#PTs) needed to synthesize a given function using the PSO-MVL is lower than the number of those needed to synthesize the same function using ACO-MVL. The maximum percentage improvement in the number of product terms obtained using PSO-MVL over the ACO-MVL is 16.3% (in the case of functions with 16 minterms) while the minimum percentage improvement is 3.9% (in the case of functions having 7 minterms). Over all simulated 50,000 functions, the average percentage improvement is 5.4%. It should be noted that 5.4% represents a considerable improvement given that the simulated randomly generated functions are representative of a population consisting of $4^{16}$ functions. We also observe that as the number of minterms in a given function increases, the likelihood of an ant finding a solution increases and hence the improvement in the number of PTs achieved by the ACO-MVL algorithm. Figure 14 illustrates this comparison.

The results show that on average both ACO-MVL and PSO produce better number of product terms for the benchmark functions. The average percentage improvement is 6% in

![Figure 13. Average #PTs achieved using PSO versus those achieved using existing synthesis techniques.](image)

<table>
<thead>
<tr>
<th>#minterms</th>
<th># functions</th>
<th>#PTS (ACO-MVL)</th>
<th>#PTs (PSO-MVL)</th>
<th>(%) improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>500</td>
<td>6.730</td>
<td>5.786</td>
<td>16.3</td>
</tr>
<tr>
<td>15</td>
<td>2700</td>
<td>7.054</td>
<td>6.860</td>
<td>2.8</td>
</tr>
<tr>
<td>14</td>
<td>6600</td>
<td>7.163</td>
<td>6.927</td>
<td>3.4</td>
</tr>
<tr>
<td>13</td>
<td>10,500</td>
<td>7.182</td>
<td>6.886</td>
<td>4.3</td>
</tr>
<tr>
<td>12</td>
<td>11,300</td>
<td>7.086</td>
<td>6.762</td>
<td>4.8</td>
</tr>
<tr>
<td>11</td>
<td>9000</td>
<td>6.904</td>
<td>6.569</td>
<td>5.1</td>
</tr>
<tr>
<td>10</td>
<td>5450</td>
<td>6.660</td>
<td>6.331</td>
<td>5.0</td>
</tr>
<tr>
<td>9</td>
<td>2500</td>
<td>6.356</td>
<td>6.055</td>
<td>5.0</td>
</tr>
<tr>
<td>8</td>
<td>1100</td>
<td>5.934</td>
<td>5.697</td>
<td>4.2</td>
</tr>
<tr>
<td>7</td>
<td>275</td>
<td>5.480</td>
<td>5.274</td>
<td>3.9</td>
</tr>
<tr>
<td>6</td>
<td>75</td>
<td>4.960</td>
<td>4.733</td>
<td>4.8</td>
</tr>
</tbody>
</table>

Total = 50000  
Average = 5.4%

![Table 6. Comparison between ACO-MVL and PSO-MVL in synthesis of the benchmark functions.](image)
The Espresso logic minimizer is a computer program using heuristic and specific algorithms for efficiently reducing the complexity of digital electronic gate circuits. Espresso was originally developed for binary logic at IBM [41]. A variant called Espresso-MV has been introduced in 1986 under the title “Multiple-Valued Logic Minimization for PLA Synthesis” [42]. We have simulated the same set of 50,000 randomly generated benchmark MVL function using Espresso-MV. The obtained results are shown in Table 7. The table also shows the results obtained using both ACO-MVL and PSO-MVL.

### Table 7.
Comparison among ACO-MVL, PSO-MVL, and Espresso-MV in synthesis of the benchmark.

<table>
<thead>
<tr>
<th># minterms</th>
<th># functions</th>
<th>#PTs ACO-MVL</th>
<th>#PTs PSO-MVL</th>
<th>#PTs Espresso-MV</th>
<th>% Improvement ACO-MVL vs. Espresso-MV</th>
<th>% Improvement PSO vs. Espresso-MV</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>500</td>
<td>6.730</td>
<td>5.786</td>
<td>8.36</td>
<td>24.2</td>
<td>44.5</td>
</tr>
<tr>
<td>15</td>
<td>2700</td>
<td>7.054</td>
<td>6.860</td>
<td>8.06</td>
<td>14.3</td>
<td>17.5</td>
</tr>
<tr>
<td>14</td>
<td>6600</td>
<td>7.163</td>
<td>6.927</td>
<td>7.76</td>
<td>8.3</td>
<td>12.0</td>
</tr>
<tr>
<td>13</td>
<td>10,500</td>
<td>7.182</td>
<td>6.886</td>
<td>7.47</td>
<td>4.0</td>
<td>8.5</td>
</tr>
<tr>
<td>12</td>
<td>11,300</td>
<td>7.086</td>
<td>6.762</td>
<td>7.16</td>
<td>1.0</td>
<td>5.9</td>
</tr>
<tr>
<td>11</td>
<td>9000</td>
<td>6.904</td>
<td>6.569</td>
<td>7.48</td>
<td>8.3</td>
<td>13.9</td>
</tr>
<tr>
<td>10</td>
<td>5450</td>
<td>6.660</td>
<td>6.331</td>
<td>6.80</td>
<td>2.1</td>
<td>7.4</td>
</tr>
<tr>
<td>9</td>
<td>2500</td>
<td>6.356</td>
<td>6.055</td>
<td>6.41</td>
<td>0.8</td>
<td>5.9</td>
</tr>
<tr>
<td>8</td>
<td>1100</td>
<td>5.934</td>
<td>5.697</td>
<td>5.99</td>
<td>0.9</td>
<td>5.1</td>
</tr>
<tr>
<td>7</td>
<td>275</td>
<td>5.480</td>
<td>5.274</td>
<td>5.53</td>
<td>0.9</td>
<td>4.9</td>
</tr>
<tr>
<td>6</td>
<td>75</td>
<td>4.960</td>
<td>4.733</td>
<td>5.01</td>
<td>1.0</td>
<td>5.9</td>
</tr>
</tbody>
</table>

Total = 50000

Average = 6.0

Average = 11.9

Figure 14.
Average #PT achieved using ACO-MVL and PSO.
The second comparison we have conducted was to compare the results obtained using both of the Swarm-Intelligence-based techniques against those obtained using Espresso-MV logic minimizer [40].

Figure 15.
Average #PT achieved using ACO-MVL and PSO-MVL versus those achieved using Espresso-MV.
the case of ACO-MVL and it is 11.9% in the case of the PSO-MVL. Figure 15 illustrates this comparison.

6. Concluding remarks
This paper provides a review and comparison on the performance evaluation of the Ant Colony (ACO) and the Particle Swarm Optimization (PSO) heuristic techniques in the synthesis of MVL functions. This evaluation is provided on two levels: Swarm-Intelligence (SI)-based versus conventional Direct-Cover (DC)-based algorithms and between the two SI-based algorithms. The covered algorithms were simulated and tested using a benchmark consisting of 50,000 randomly generated 2-variable 4-valued functions. The average #PTs needed is used as a criterion in comparing the performance of all considered algorithms. The results obtained indicate that the SI-based techniques outperform the conventional DC in terms of the average #PTs required to realize a given MVL function. Between the two SI-based algorithms, the technique based on the PSO produces better overall results. The performance evaluation show also that the results obtained using both the SI-based algorithms outperform the results obtained using the espresso-MVL logic minimizer. It is shown that both the ACO and the PSO outperform the Espresso-MVL algorithm in the minimization of MVL functions. The work presented in this publication considers using ACO and PSO in the synthesis of 4-valued logic functions and compares the results to those obtained conventional using the direct-cover techniques. Application of other swarm intelligence techniques, e.g. Genetic Algorithms or Artificial Bee Colony for 4-valued or higher-radix logic is not covered in the article.

References


**Corresponding author**
Mostafa Abd-El-Barr is the corresponding author.