Received 30 December 2019
Revised 12 February 2020
Accepted 2 March 2020

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

Paper type Original Article

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

Publishers note: The publisher wishes to inform readers that the article "Swarm intelligence versus direct cover algorithms in synthesis of Multi-Valued Logic functions" was originally published by the previous publisher of *Applied Computing and Informatics* and the pagination of this article has been subsequently changed. There has been no change to the content of the article. This change was necessary for the journal to transition from the previous publisher to the new one. The publisher sincerely apologises for any inconvenience caused. To access and cite this article, please use Al-kahtani, M.S., Karim, L., Khan, N. (2020), "Swarm intelligence versus direct cover algorithms in synthesis of Multi-Valued Logic functions", *Applied Computing and Informatics*. Vol. ahead-of-print No. ahead-of-print. https://10.1016/j.aci.2020.03.002. The original publication date for this paper was 10/03/2020.

Applied Computing and Informatics
Vol. 20 No. 1/2, 2024
pp. 2-19
Emerald Publishing Limited
e-ISSN: 2210-8327
p-ISSN: 2634-1964
DOI 10.1016/j.aci.2020.03.002

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 $r^{(r^n)} = 4^{4^2} = 2^{32}$ 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].

Definition 1 [15]: Define an n-variable r-valued function, $f(X)$, as a mapping $f: R^n \rightarrow R$. In the above definition $R = \{0, 1, \cdots, r-1\}$ is a set of $= r$ logic values with $r \geq 2$ and $X = \{x_1, x_2, \cdots, x_n\}$ is a set of r-valued n variables. An example 4-valued ($r4$), 2-variable ($n = 2$) function is shown in Figure 1.

Definition 2 [15]: Define a product term (PT), $P(x_1, x_2, \cdots, x_n)$, 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.

Example: ***the*** *product terms in* Figure 1 *include* $1, 1 \cdot {}^{2}x_1^{3} \cdot {}^{1}x_2^{2}, 2 \cdot {}^{0}x_1^{0} \cdot {}^{0}x_2^{3}$, *and* $3 \cdot {}^{1}x_1^{2} \cdot {}^{1}x_2^{2}$. ***In the first case*** $c = 1, a_1 = 2, b_1 = 3, a_2 = 1, b_2 = 2$, ***in the second case*** $c = 2, a_1 = 0, b_1 = 0, a_2 = 0, b_2 = 3$ ***and in the third case*** $c = 3, a_1 = 1, b_1 = 2, a_2 = 1, b_2 = 2$.

Definition 3 [15]: Define a minterm as the assignment $x_1 = a_1, x_2 = a_2, \cdots, x_n = a_n$, in an MVL function $f(x_1, x_2, \cdots, x_n)$ if and only if: $f(x_1, x_2, \cdots, x_n) \neq 0$.
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, $I(x_1, x_2, \cdots, x_n)$ of a function $f(x_1, x_2, \cdots, x_n)$ is called an implican if $f(x_1, x_2, \cdots, x_n) \geq I(x_1, x_2, \cdots, x_n)$ for all assignments of x_i's.

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

The objective of MVL synthesis 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

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

Table 1.
MVL operators.

functions. However the following table (adapted from [50]) illustrates the fundamental difference between the two models of computation.

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

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

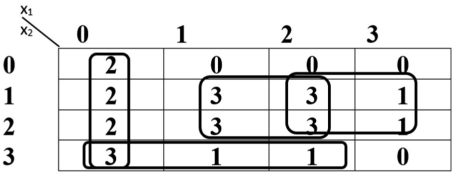


Figure 1.
An example 2-variable 4-valued function.

DC Heuristics Steps (adapted from [9])

1. ***Select*** a MVL function
2. ***Repeat*** until no more minterms remain uncovered
   (a) **Select** a minterm
   (b) **Identify** a suitable implicant that covers that minterm
   (c) **Obtain** a reduced function by removing the identified implicant

Figure 2.
Algorithm for synthesis of MVL functions using DC heuristics.

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.

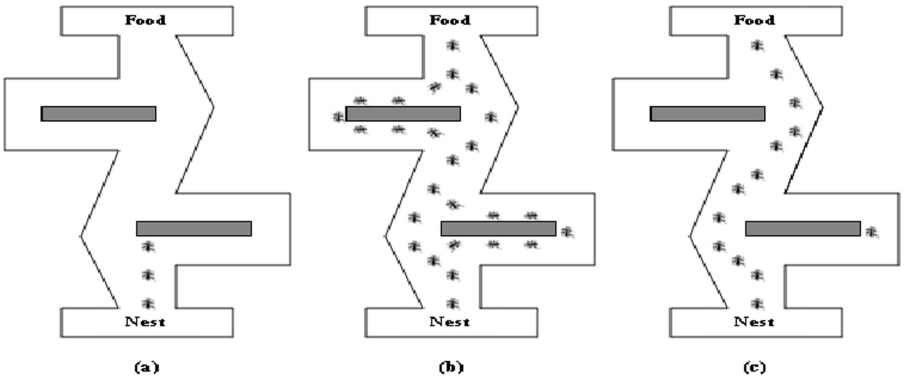


Figure 3.
Illustration of the ants' behavior [34].

ACO Algorithmic Steps (adapted from [34])
1. Representation of the solution space (usually using Construction Graph, CG)
2. Setting the ACO parameters and initializing the pheromone trails
3. Repeat until convergence or meeting termination conditions
(a) Generating Ant solutions using ant's walk along CG guided by pheromone trails
(b) Updating pheromone concentrations

Figure 4.
ACO basic algorithmic steps.

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 seach has 11 minterms, and 75 functions each has 6 minterms.

# functions	500	2700	6600	1050	1130	9000	5450	2500	1100	275	75
# minterms	16	15	14	13	12	11	10	9	8	7	6

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.

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.

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 ← ant[a].L + L; }
      (d) calculate_fitness(ant[a]);
   3.2. pheromone_update_minterm();
   3.3. pheromone_update_implicant();

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 $1.\overset{12}{\underset{1}{x}}.\overset{13}{x_2}$, 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 η_M and the pheromone evaporation τ_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:

$$p = \frac{\tau_M \cdot \eta_M}{\sum \tau_M \cdot \eta_M} \tag{1}$$

where τ_M represents the pheromone value of the minterm M and η_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's selection criteria. The probability of choosing implicant L is computed as:

$$p = \frac{\tau_L \cdot \eta_L}{\sum \tau_L \cdot \eta_L} \tag{2}$$

where τ_L represents the pheromone value of implicant L and η_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, $\boldsymbol{F_f = N_h - N_m}$, where $\boldsymbol{N_h}$ = the number of minterm matching between the two truth tables and $\boldsymbol{N_m}$ = the number of mismatch between the two truth tables. The Objective Fitness, $\boldsymbol{F_o = (100 - N_p)/100}$, where $\boldsymbol{N_p}$ 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 ($\boldsymbol{NP}$) = 20

Maximum number of Iterations (***MI***) = 100
Maximum number of Runs = (***MR***) 10
Pheromone evaporation rate (***RHO***) = 0.95
The following two measures were recorded:

1. ***SR***μ ≡ Successful Rate (number of successful iterations divided by the total number of runs)
2. ***PT*** ≡ 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: c1 = 1, c2 = c3 = 0.5 and the number of runs = 10. Notice that c1 and c2 represent the cognitive coefficients. As a result of our experimentations, we found out that most appropriate value for V_{max} is either 4 or 5. A value of $V_{max} = 5$ 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 ***Ff*** = ***Nh*** − ***Nm,*** where ***Nh*** is the number of minterm match and ***Nm*** is the number of minterms mismatch between the truth table of the original and the obtained functions. The objective fitness is computed as ***Fo*** = ***(100*** − ***Np)/100,*** where ***Np*** 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 2.
Average #PTs needed for realization of MVL Functions of the benchmark.

# minterms	16	15	14	13	12	11	10	9	8	7	6
# functions	500	2700	6600	1050	1130	9000	5450	2500	1100	275	75
#PT (ARM) [11]	7.59	8.30	8.36	8.28	8.05	7.71	7.32	6.87	6.31	5.73	5.13
#PT (BS) [9]	7.56	8.31	8.41	8.35	8.10	7.76	7.37	6.88	6.32	5.75	5.15
#PT (DM) [10]	7	7.51	7.57	7.54	7.38	7.13	6.83	6.47	6.02	5.53	4.97
#PT (ACO-MVL) [14]	6.73	7.05	7.16	7.18	7.09	6.9	6.66	6.36	5.93	5.48	4.96
Max improvement (%)	12.8	12.4	12.4	12.4	12.4	12.4	12.4	12.4	12.4	12.4	12.4
Min improvement (%)	4	6.5	5.7	5	4.2	3.3	2.6	1.8	1.5	0.9	0.3

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 ($D_{t+1,i}$):

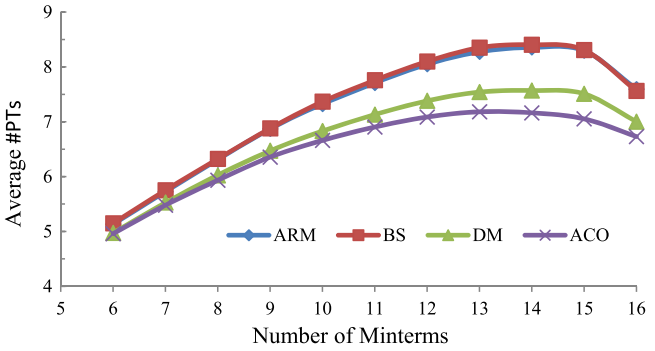


Figure 7.
Average #PTs using ACO-MVL versus those achieved using DC-based technique.

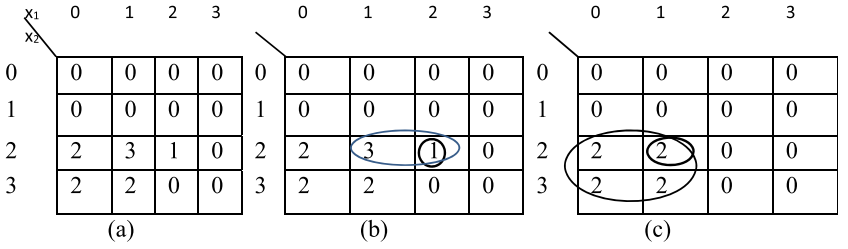


Figure 8.
Example MVL function.

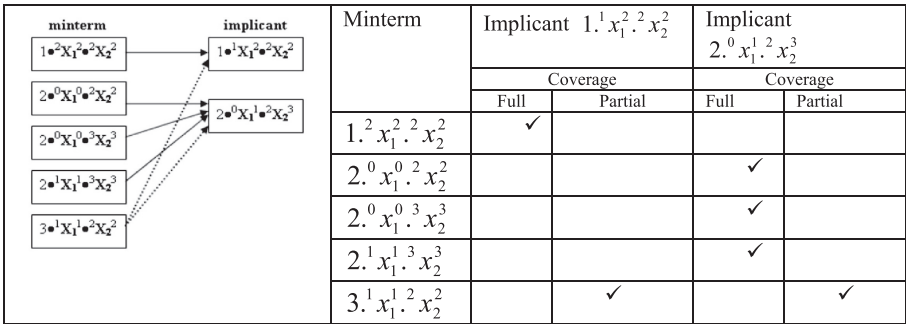


Figure 9.
Coverage relation between minterms and implicants for function shown in Figure 8.

Particle 1	Particle 2	Particle 3
00000	00000	00000
00000	00000	00000
00000	00000	00000
00000	00000	00000
00000	00000	00000
00000	00000	00000
00000	00000	00000
00000	00000	00000
20122	20122	20123
00000	00000	11222
11222	11222	00000
00000	00000	00000
20123	20033	00000
00000	21133	00000
00000	00000	00000
00000	00000	00000

Figure 10.
Example of particle representation.

Table 3.
Fitness calculation for particle shown in Figure 11.

Particle	The obtained Truth Table	N_h	N_m	F_f	N_P	F_o	Fitness
1	0000000033102200	15	1	14	3	0.97	14.97
2	0000000023102200	16	0	16	4	0.96	16.96
3	0000000023102200	16	0	16	2	0.98	16.98

Table 4.
Average numbers of PTs for different values of particles and iterations.

	Number of Particles		
Number of iterations	20	50	100
1000	7.92	7.59	7.40
2000	7.67	7.39	7.25
5000	7.40	7.20	7.11

$$\begin{aligned} DS_{t,i} &= c_2 r_2 (P_{i,t} - X_{t,i}) \\ DG_{t,i} &= c_3 r_3 (P_{g,t} - X_{t,i}) \\ D_{t+1,i} &= c_1 D_{t,i} + DS_{t,i} + DG_{t,i} \end{aligned} \tag{4}$$

In the above computation we use $DS_{t,i}$ to represent the displacement of particle i at time step t due to particle best self- experience and $DG_{t,i}$ 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, $P_{i,t}$ to represent the current best position at time step t, and $P_{g,t}$ 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: ($D_{t,i}$), ($DS_{t,i}$), and ($DG_{t,i}$). The new particle position is formulated as follows:

$$X_{t+1,i} = (X_{t,i} + DS_{t,i}) + DG_{t,i} + c_1 D_{t,i}. \tag{5}$$

The PSO algorithm is illustrated in Figure 11 [16].

4.2.4 Numerical results. We have tested the proposed PSO-MVL algorithm using a bench mark 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: c1 = 1, c2 = c3 = 0.5 and the number of runs = 10. As a result of our experimentations, we found out that most appropriate value for V_{max} is either 4 or 5. A value of $V_{max} = 5$ 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 F1requires 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.

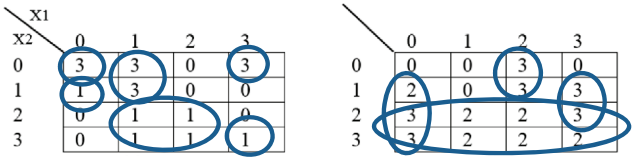
Table 5 shows the results obtained using the modified PSO-MVL in comparison with the DC-based conventional heuristics for different functions having different number of minters.

In this table we present the average number of minterms required to synthesize a bench mark 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.

Algorithm PSO-MVL [16]

1. Initialize each particle's parameters
2. Repeat while maximum iterations or minimum error criteria is not attained
3. Repeat for each particle
   3.1. Calculate Particle Fitness value
   3.2. Compare the obtained Fitness with current Fitness values
   3.3. If the obtained fitness value is better then set current value as the new $P_{i,t}$
4. Choose the particle with the best fitness value of all as $P_{g,t}$
5. Repeat for each particle
   5.1. Calculate particle displacement using the equation $D_{t+1,i} = c_1 D_{t,i} + DS_{t,i} + DG_{t,i}$
   5.2. Update particle position according to the equation $X_{t+1,i} = (X_{t,i} + DS_{t,i}) + DG_{t,i} + c_1 D_{t,i}$.

Figure 11.
The PSO algorithm for MVL synthesis.



(a) Example function F1 (b) The Complement function F2

Figure 12.
Example function and its complement function.

Table 5.
Average PT with Respect to Different Number of Minterms of MVL Functions.

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

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-vauled 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 percentage improvement decreases. This can be attributed to the observation that as the number of minterms 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

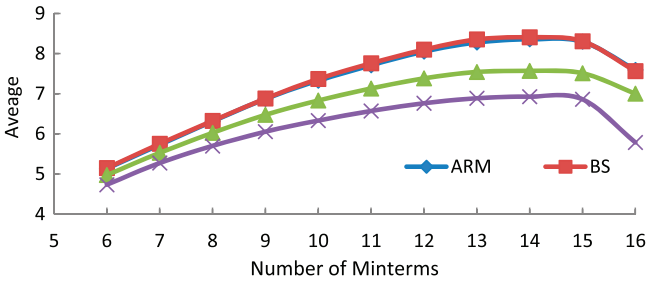


Figure 13. Average #PTs achieved using PSO versus those achieved using existing synthesis techniques.

#minterms	# functions	#PTS (ACO-MVL)	#PTs (PSO-MVL)	(%) improvement
16	500	6.730	5.786	**16.3**
15	2700	7.054	6.860	2.8
14	6600	7.163	6.927	3.4
13	10,500	7.182	6.886	4.3
12	11,300	7.086	6.762	4.8
11	9000	6.904	6.569	5.1
10	5450	6.660	6.331	5.
9	2500	6.356	6.055	5.0
8	1100	5.934	5.697	4.2
7	275	5.480	5.274	**3.9**
6	75	4.960	4.733	4.8
	Total = 50000			Average = 5.4%

Table 6. Comparison between ACO-MVL and PSO-MVL in synthesis of the benchmark functions.

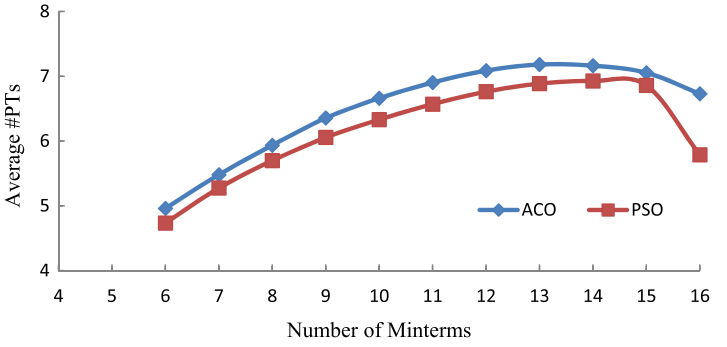


Figure 14.
Average PT achieved using ACO-MVL and PSOThe second comparison we have conducted was to compare the results obtained using both of the Swarm-Intelligence-based techniques against those obtained using *Espresso-MV* logic minimizer [40].

Note(s): 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.

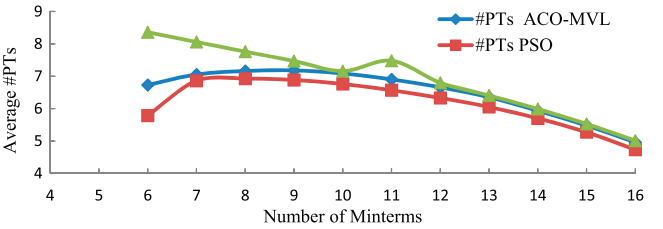


Figure 15.
Average #PT achieved using ACO-MVL and PSO-MVL versus those achieved using Espresso-MV.

# minterms	# functions	#PTs ACO-MVL	#PTs PSO-MVL	#PTs Espresso-MV	% Improvement ACO-MVL vs. Espresso-MV	% Improvement PSO vs. Espresso-MV
16	500	6.730	5.786	8.36	24.2	44.5
15	2700	7.054	6.860	8.06	14.3	17.5
14	6600	7.163	6.927	7.76	8.3	12.0
13	10,500	7.182	6.886	7.47	4.0	8.5
12	11,300	7.086	6.762	7.16	1.0	5.9
11	9000	6.904	6.569	7.48	8.3	13.9
10	5450	6.660	6.331	6.80	2.1	7.4
9	2500	6.356	6.055	6.41	0.8	5.9
8	1100	5.934	5.697	5.99	0.9	5.1
7	275	5.480	5.274	5.53	0.9	4.9
6	75	4.960	4.733	5.01	1.0	5.9
	Total = 50000				Average = 6.0	Average = 11.9

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

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

[1] (Eds.), "Beyond Two: Theory and Applications of Multiple-Valued Logic", 2004.

[2] E. Dubrova, Multiple-valued logic in VLSI design, J. Soft Comput. (2002) 1–17.

[3] K. Naiff, D. Rich, K. Smalley, A four-state ROM using multilevel process technology, IEEE J. Solid-State Circ. 19 (2) (1984) 174–179.

[4] T. Hanyu, M. Kameyama, A 200 MHz pipelined multiplier using 1.5 V-supply multiple-valued MOS current-mode circuits with dual-rail source-coupled logic, IEEE J. Solid-State Circ. 30 (11) (1995) 1239–1245.

[5] V. Patel, K. Gurmurthy, Arithmetic operations in multi-valued logic, Int. J. VLSI Commun. Syst. (VLSICS) 1 (1) (2010) 21–32.

[6] R. Hong, D. Ostapko, A heuristic approach for logic minimization, IBM J. Res. Dev. 18 (5) (1974) 443–458.

[7] C. Files, M. Perkowski, New multi-valued functional decomposition algorithms based on MDDs, IEEE Trans. CAD 19 (9) (2000) 1081–1086.

[8] J.-H. Goa, J.-H. Jiang, Y. Jiang, Y. Li, A. Mishchenko, S. Sinha, T. Villa, and R. Brayton, "Optimization of multi-valued multi-level networks", In Proceedings of the International Symposium on Multiple-Valued Logic, pp. 168–177, 2002.

[9] P. Besslich, Heuristic minimization of MVL functions: a direct cover approach, IEEE Trans. Comput. 35 (2) (1986) 134–144.

[10] G. Dueck, D. Miller, A Direct Cover MVL Minimization Using the Truncated Sum, In Proceedings of the International Symposium on Multiple-Valued Logic, pp. 221-226, 1987.

[11] G. Promper, J. Armstrong, Representation of multiple-valued functions using the direct cover method, IEEE Transactions on Computers, pp. 674–679, 1981.

[12] Y. Hata, T. Hayase, N. Hozumi, K. Yamato, Multiple-Valued Logic Minimization by Genetic Algorithms, In Proceedings of 27th IEEE International Symposium on Multiple-Valued Logic, pp. 97–102, 1998.

[13] B. Sarif, M. Abd-El-Barr, Synthesis of MVL Functions – Part I: The Genetic Algorithm Approach, In Proceedings of the International Conference on Microelectronics, pp. 154–157, 2006.

[14] M. Abd-El-Barr, B. Sarif, "Synthesis of MVL Functions – Part II: The Ant Colony Optimization Approach", In Proceedings of the International Conference on Microelectronics, pp. 158–161, 2006.

[15] M. Abd-El-Barr, Evolutionary techniques in synthesis of multiple-valued logic functions, Int. J. New Comput. Arch. Appl. 2 (3) (2012) 411–422.

[16] B. Sarif, M. Abd-El-Barr, Functional Synthesis using Discrete Particle Swarm Optimization, In Proceedings 2008 IEEE Swarm Intelligence Symposium, St. Louis, USA< September 2008, pp. 1–8.

[17] M. Abd-El-Barr, L. Al-Awami, Analysis of Direct Cover Algorithms for Minimization of MVL Functions, In Proceedings of the 15th International Conference on Microelectronics, pp. 308–312, 2003.

[18] G. Dueck, J. van Rees, On the Maximum Number of Implicants Needed to Cover a Multiple-Valued Logic Function Using Window Literals, In Proceedings of the International Symposium on Multiple-Valued Logic, pp. 280–286, 1991.

[19] M. Dorigo, T. Stutzle, The Ant Colony Optimization Meta-heuristic: Algorithms, Applications and Advances, In Handbook of Meta-heuristics, Kluwer Academic Publishers, pp. 251–285, 2002.

[20] B. Sarif, B, M. Abd-El-Barr, Fuzzy-based Direct Cover Algorithm for Synthesis of Multiple-Valued Logic Functions, In proceedings of the IASTED on Circuits and Systems, Hawaii, pp. 625–630, 2008.

[21] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, Piscataway, 1995, pp. 1942–1948.

[22] J. Kennedy, R. Eberhart, A discrete binary version of the particle swarm algorithm, in: IEEE International Conference on Systems, Man, and Cybernetics, 1997, pp. 4104–4108.

[23] Y. Wang, Y. Wu, Frequency graphs for travelling salesman problem based on ant colony optimization, Int. J. Comput. Intelligence Appl. 18 (3) (2019) 16, https://doi.org/10.1142/S1469026819500160.

[24] M. Abd-El-Barr, B. Sarif, Weighted and ordered direct cover algorithms for minimization of MVL Functions, in: Proceedings 37th International Symposium on Multiple-valued Logic (ISMVL 2007), 2007, pp. 48–53.

[25] H. Ahmed, J. Glasgow, "Swarm Intelligence: Concepts, Models, and Applications", Technical Report 2012–585, Queen's University, School of Computing, 2012.

[26] C. Reis, J. Machado, Computational intelligence in circuit synthesis, J. Comput. Intelligence 11 (9) (2007) 1–6.

[27] E. Osaba, J. Del Sel, A. Iglesias, X.-S. Yang, Soft computing for swarm robotics: new trends and applications, J. Comput. Sci. 39 (2020) 101049.

[28] D. Karaboga, B. Akay, A Survey: algorithms simulating bee swarm intelligence, J. Artific. Intelligence Rev. 31 (2009) 61–85.

[29] J. Pugh, A. Martinoli, Discrete multi-valued particle swarm optimization, in: Proceedings, IEEE Swarm Intelligence Symposium, 2006, pp. 103–110.

[30] M. Mareli, B. Twala, An adaptive cuckoo search algorithm for optimization, Appl. Comput. Inf. 14 (2018) 103–115.

[31] P. Balaprakash, M. Birattari, T. Stzle, Z. Yuan, M. Dorigo, Estimation-based ant colony optimization algorithm for the travelling salesman problem, Swarm Intelligence 3 (2) (2009) 223–242.

[32] C.-Y. Lee, Z.-J. Lee, S.-W. Lin, K.-C. Ying, An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem, Appl. Intelligence 32 (2010) 88–95.

[33] O. Korb, T. Stutzle, T. Exner, An ant colony optimization approach to flexible protein ligand docking, Swarm Intelligence 1 (2) (2007) 115–134.

[34] M. Dorigo, T. Stutzle, Ant Colony Optimization, MIT Press, Cambridge, 2004.

[35] R. Eberthart, Y. Shi, Tracking and optimizing dynamic systems with particle swarms. Proc. Congress on Evolutionary Computation, Seoul, 2001.

[36] W. Ghraby, M. Wachowiak, R. Smolikova, Y. Zheng, J. Zurada, M. Elmaghraby, An approach of multimodal biomedical image registration utilizing particle swarm optimization, IEEE Trans. Evol. Comput. (2004).

[37] L. Messerschmidt, A. Engelbrecht, Learning to play games using a PSO-based competitive learning approach, IEEE Trans. Evol. Comput. (2004).

[38] T. Blackwell, P. Bently, Improved music with swarms, in: Proc. 2002 Congress on Evolutionary Computation (ECE), 2002, pp. 1462–1467.

[39] Y. Shi, On particle swarm optimization, in: IEEE Neural Network Society Feature Article, 2004, pp. 8–13.

[40] Y. Valle, G. Venayagmoorthly, S. Mohagheghi, J.-C. Hernandez, R. Harley, Particle Swarm optimization: basic concepts, variants and applications in power systems, IEEE Trans. Evol. Comput. 12 (2) (2008) 171–195.

[41] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, ISBN 0-89838-164-9, 1984.

[42] R. Rudell, Multiple-Valued Logic Minimization for PLA Synthesis, Memorandum No. UCB/ERL M86-65 (Berkeley), 1986.

[43] B. Bachir, A. Ali, M. Abdellah, Multi-objective optimization of an operational amplifier by the ant colony optimization algorithm, Electric. Electron. Eng. 2 (4) (2012) 230–235.

[44] J. Wang, S. Jin, S. Qin, H. Jiang, Swarm intelligence-based hybrid models for short-term power load prediction, Math. Probl. Eng. 2014 (2014) https://doi.org/10.1155/2014/712417.

[45] T. Ganesan, I. Elamvazuthi, K. Shari, P. Vasani, Swarm and gravitational search algorithms for multi-objective optimization of synthesis gas production, Appl. Energy (2013) https://doi.org/10.1016/j.apenergy.2012.09.059.

[46] H. Wang, H. Sun, C. Li, S. Rahnamayan, J.-S. Pan, Diversity enhanced particle swarm optimization with neighborhood search, Inf. Sci. 223 (2013) 119–135.

[47] S. Chatterjee, A. Sarkar, S. Hore, N. Dey, A. Ashour, V. Balas, Particle swarm optimization trained neural network for structural failure prediction of multistoried RC buildings, Natural Comput. Appl. 28 (8) (2017) 2005–2016.

[48] N. Dey, A. Ashour, S. Bhattachayya, Applied Nature-Inspired Computing: Algorithms and Case Studies, 2019.

[49] N. Dey, Advancements in Applied Meta-heuristic Computing, IGI, 2017.

[50] H.Ahmed, J.Glasgow, Swarm Intelligence: Concepts, Models, and Applications, Schools of Computing, Queen's University, Technical Report, 201–585.

[51] Y. Hei, P. Du, Optimal choice of the parameters of ant colony algorithm, J. Convergence Inf. Technol. 6 (9) (2011) 96–104.

Corresponding author
Mostafa Abd-El-Barr can be contacted at: mostafa.abdelbarr@gmail.com