2015 IEEE Symposium Series on Computational Intelligence

Frequency Distribution of Candidate Solutions in Angle Modulated Particle Swarms

Barend J. Leonard
Department of Computer Science
University of Pretoria
Pretoria
South Africa
Email: bleonard@cs.up.ac.za

Andries P. Engelbrecht
Department of Computer Science
University of Pretoria
Pretoria
South Africa
Email: engel@cs.up.ac.za

***Abstract*—This paper investigates the frequency distribution of candidate solutions in the search space when angle modulation is applied to particle swarm optimisation (PSO). It is shown that angle modulated particle swarm optimisers (AMPSO) have non-uniform solution frequency distributions. A new technique is introduced to ensure that the frequency distribution of candidate solutions is uniform. The new technique is compared with AMPSO and three AMPSO variants, as well as binary PSO (BPSO) on a number of problem cases. It is shown that AMPSO algorithms obtain lower average fitness values on binary minimisation problems whose optimal solutions contain repetition. However, when optimal solutions do not contain repetition, AMPSO and its variants are at a clear disadvantage.**

I. Introduction

In optimisation theory, an optimisation problem is referred to as a *continuous optimisation problem*, if the input variables to the objective function f are real values. That is, for some objective function $f(\boldsymbol{x}) : \mathcal{S} \rightarrow \mathbb{R}$, $\mathcal{S} = \mathbb{R}^{n_x}$ is the search space, and $x_j \in \mathbb{R}$ for each dimension $j = 1, \ldots, n_x$. When $\mathcal{S}$ is restricted to some discrete subset of $\mathbb{R}$, f is referred to as a *discrete optimisation problem*. In particular, if $\mathcal{S} = \mathbb{B}^{n_x}$, where $\mathbb{B} = \{0, 1\}$, then $x_j \in \mathbb{B}$, and f is a *binary optimisation problem*. In this paper, the term "continuous optimisation problem" is used synonymously with "continuous problem". Similarly the terms "binary optimisation problem" and "binary problem" are used interchangeably.

The particle swarm optimisation (PSO) algorithm was introduced in by Kennedy and Eberhart in 1995 [2], [6]. The algorithm was later modified by Shi and Eberhart [16]. PSO is defined for continuous optimisation problems. In order to solve binary problems, Kennedy and Eberhart introduced a discrete version of PSO that they called binary particle swarm optimisation (BPSO) [7]. The BPSO algorithm has been criticised in literature for varying too much from the original PSO algorithm. Specifically, a new interpretation of particle velocities means that particle swarm optimisation theory—which describes the trajectories and convergence behaviour of particles—does not apply to BPSO [13]. Additionally, the control parameters (ω, c_1, and c_2) of the PSO algorithm have different meanings in the context of BPSO [4], [8].

Pampara *et al.* introduced angle modulated particle swarm optimisation (AMPSO) as an alternative method to optimise binary problems using PSO [14]. However, the concept of angle modulation in the context of optimisation algorithms was originally suggested by Franken [5]. In AMPSO, a standard PSO is used to optimise the coefficients of a trigonometric function g. The coefficients control the shape of g. In order to generate a binary solution, the coefficients found by PSO are first substituted into the function. Subsequently, the function is sampled at regular intervals. The value of g at each sampling interval is then mapped to binary digit. Thus, AMPSO enables PSO to solve n_b-dimensional binary problems in g_c-dimensional real-valued space, where g_c is the number of coefficients of g. Because the coefficients of g are continuous, AMPSO requires no modifications to the PSO algorithm. That is, particle positions and velocities are still real-valued vectors. Therefore, PSO theory conveniently applies to AMPSO.

The AMPSO algorithm has been shown to outperform other binary optimisation algorithms on a number of binary problems [13], [14]. In addition the algorithm has been successfully applied to real-world scenarios [11], [12].

Despite the success of AMPSO, Leonard and Engelbrecht identified a number of potential flaws in the algorithm and introduced three variants to overcome those issues [9]. While the AMPSO variants showed better performance in some specific problem cases, the study failed to provide clear reasons as to why the normal AMPSO algorithm performed poorly in those cases. In order to better understand the underlying reasons that causes AMPSO to fail on some problems, Leonard *et al.* investigated various aspects of AMPSO, including the effect of the generating function on the algorithm's search capabilities [10]. It was hypothesised that the generating function causes a non-uniform frequency distribution of candidate binary solutions in the search space. If this is indeed the case, it would mean that some binary solutions are easier to find simply because they exist in the search space with higher frequencies than other candidate solutions. This paper investigates how the non-uniform frequency distribution of binary solutions arises and how it affects the performance of the AMPSO algorithm. In addition, a new method is suggested, which guarantees that all solutions exist in the search space with the same frequency.

The rest of this paper is structured as follows: section II gives an overview of the PSO algorithm, while the BPSO algorithm is explained in section III. The AMPSO algorithm and its variants are discussed in sections IV and V, respectively. In section VI the frequency distribution of binary solutions in the AMPSO search landscape is investigated. Section VII introduces a technique to ensure that the frequency distribution of solutions is uniform. Section VIII explains the experimental setup, while the results are discussed in section IX. The paper is concluded in section X.

II. Particle Swarm Optimisation

The PSO algorithm is a stochastic, population-based search algorithm. PSO maintains a population of candidate solutions, known as *particles*. The population of particles is referred to as the *swarm*. Each particle has a *position* $\boldsymbol{x}_i$ and a *velocity* $\boldsymbol{v}_i$ in the n_x-dimensional search space. Additionally, every particle in the swarm keeps track of its *personal best position* $\boldsymbol{y}_i$, which is the best solution it has found during the search process. The best position $\hat{\boldsymbol{y}}$ found by the swarm is referred to as the *global best position*.

At every time step $t+1$, the velocity of every particle in the swarm is updated using the following equation:

$$\boldsymbol{v}_i(t+1) = \omega \boldsymbol{v}_i(t) + F_{cog} + F_{soc}, \tag{1}$$

where

$$F_{cog} = c_1 \boldsymbol{r}_1(t)[\boldsymbol{y}_i(t) - \boldsymbol{x}_i(t)] \tag{2}$$

is the cognitive component,

$$F_{soc} = c_2 \boldsymbol{r}_2(t)[\hat{\boldsymbol{y}}(t) - \boldsymbol{x}_i(t)] \tag{3}$$

is the social component, ω is an inertia weight, c_1 and c_2 are acceleration coefficients, and r_{1j} and r_{2j} are random values, sampled from $U(0,1)$, in each dimension $j = 1, \ldots, n_x$.

After the velocities have been updated, each particle's position in the search space is updated as follows:

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1). \tag{4}$$

The cognitive attraction force F_{cog} attracts the particle towards its own personal best position. The c_1 acceleration coefficient scales F_{cog}, with larger values of c_1 encouraging exploration.

The force F_{soc} is an attraction force in the direction of the global best position $\hat{\boldsymbol{y}}$. The c_2 acceleration coefficient is used to scale F_{soc}. Larger values for c_2 enables faster exploitation.

The inertia weight ω scales the influence of a particle's previous velocity. The purpose of ω in PSO is to smooth particle trajectories to prevent sudden, large changes in direction until enough evidence has been gathered to justify such changes. For specific configurations of ω, c_1, and c_2, swarm convergence is theoretically guaranteed [1], [3], [15], [17].

III. Binary Particle Swarm Optimisation

In order to solve binary problems with PSO, the algorithm has to be modified such that the positions of particles are binary vectors. That is, for any particle i, $\boldsymbol{x}_i \in \mathbb{B}^{n_x}$. However, to achieve this, a new interpretation of the velocity vector is required.

Kennedy and Eberhart proposed that the velocity v_{ij} in each dimension should be interpreted as the probability that the position x_{ij} in that dimension is equal to 1. The immediate implication of this interpretation is that particle velocities must be restricted, such that $v_{ij} \in [0,1]$. In BPSO, particle velocities are normalised to the range $[0,1]$ by using the sigmoid function:

$$v'_{ij}(t) = sig(v_{ij}(t)) = \frac{1}{1+e^{-v_{ij}(t)}}. \tag{5}$$

The position of a particle is then updated using

$$x_{ij}(t+1) = \begin{cases} 1 & \text{if } r_{3j} < v'_{ij}(t) \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $r_{3j} \sim U(0,1)$ is a random variable.

Under BPSO's interpretation of particle velocities, the acceleration coefficients c_1 and c_2 no longer control the trade-off between exploration and exploitation. Instead, maximum exploration capabilities are achieved when $v'_{ij} = 0.5$, so that every bit of a particle's position has a 50% chance of being 1. Note that, because of the sigmoid normalisation, $v'_{ij} = 0.5$ iff $v_{ij} = 0$.

Furthermore, the meaning of the inertia weight ω changes, because particles no longer follow a trajectory through the search space. The inertia weight in BPSO should be small initially to prevent particle velocities from growing too fast. In this way, exploration is facilitated during the beginning of the search process.

Finally, the concept of a particle's position in the search space no longer makes sense. This is because a particle does not have a position until its velocity is evaluated using equation (6). Furthermore, because of the random variable r_{3j} in equation (6), repeated evaluations of the same particle are likely to result in different binary position vectors. Therefore the two attraction forces F_{cog} and F_{soc} in equation (1)—that are supposed to guide particles to good regions in the search space—become meaningless in this context.

IV. Angle Modulated Particle Swarm Optimisation

AMPSO is a binary optimisation technique that makes use of PSO to optimise the coefficients of the following trigonometric function:

$$g(x) = sin[2\pi(x-a)b\ cos(2\pi(x-a)c)] + d. \tag{7}$$

This function is referred to as the *generating function*. The coefficients of g control its shape by acting on the function's frequency and displacement. To optimise the coefficients of g, particle positions in PSO take the form $\boldsymbol{x}_i = (a,b,c,d)$.

In order to generate a binary solution, the position of a particle is substituted into the generating function. Then, the function is sampled at regular intervals $x = 1,2,3,\ldots,n_b$, where n_b is the required number of binary digits. A binary solution $\mathcal{B} \in \mathbb{B}^{n_b}$ is then constructed by noting the value $g(x)$ at each sampling position:

$$\mathcal{B}_j = \begin{cases} 0 & \text{if } g(x) \leq 0, \\ 1 & \text{otherwise.} \end{cases} \tag{8}$$

This process is illustrated in figure 1. The binary solution is then evaluated in order to assign a fitness value $f(\mathcal{B})$ to the

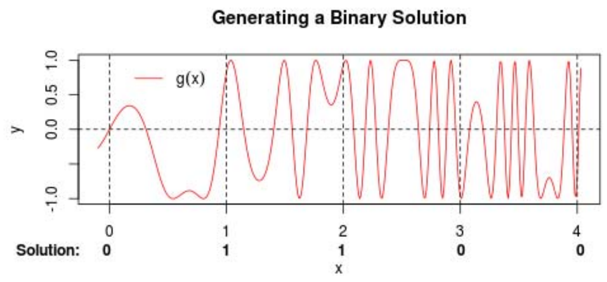


Fig. 1. A 5-bit binary solution is constructed by sampling the generating function at regular intervals. In this illustration, $a = 0$, $b = 0.5$, $c = 0.8$, and $d = 0$. Or $\boldsymbol{x}_i = (0.0, 0.5, 0.8, 0.0)$.

given particle, where f is the n_b-dimensional binary objective function.

In this way, an n_b-dimensional binary problem can be solved using a 4-dimensional PSO. Furthermore, because the coefficients of g are continuous, no modifications have to be made to the PSO algorithm. This implies that the meanings of the PSO parameters and the PSO theory, as discussed in section II, apply to AMPSO without modification.

V. AMPSO Variants

This section describes two variants of AMPSO that were introduced by Leonard and Engelbercht [9], [10]. It was found that these two variants produced better results than AMPSO on a number of specific problem cases. The amplitude AMPSO (A-AMPSO) algorithm is discussed in section V-A. Section V-B gives an overview of the Min-Max AMPSO (MM-AMPSO) algorithm.

A. Amplitude AMPSO

The A-AMPSO algorithm adds an additional coefficient e to the generating function. The e coefficient controls the amplitude of the generating function. Thus, equation (7) changes to

$$g(x) = e\ sin[2\pi(x - a)b\ cos(2\pi(x - a)c)] + d. \quad (9)$$

Particle positions in PSO then become five-dimensional vectors of the form $\boldsymbol{x} = (a, b, c, d, e)$.

The ability to control the amplitude of the generating effectively influences the effect of the vertical shift parameter d. For smaller values of e, d has a greater effect on the binary solution that is generated.

B. Min-Max AMPSO

Min-Max AMPSO (MM-AMPSO) allows the algorithm to control the sampling range within which the generating function is sampled to produce a binary solution. To achieve this, particle positions in PSO become six-dimensional vectors of the form $\boldsymbol{x} = (a, b, c, d, \alpha_1, \alpha_2)$.

Let $\alpha_l = min\{\alpha_1, \alpha_2\}$ and $\alpha_u = max\{\alpha_1, \alpha_2\}$. Then, the generating function is sampled at every δ^{th} interval in the range $[\alpha_l, \alpha_u)$, where

$$\delta = \frac{\alpha_u - \alpha_l}{n_b}. \quad (10)$$

Controlling the sampling range of g allows the MM-AMPSO algorithm to focus on specific parts of the generating function, making some solutions easier to generate.

C. Ensemble AMPSO

Ensemble AMPSO (E-AMPSO) was suggested by Leonard *et al.* as a way to solve high dimensional binary problems by dividing the problem into multiple parts. A generating function is then assigned to each binary sub-problem, and PSO is used to optimise the coefficients of all the generating functions.

If an n_b-dimensional binary problem is divided into ϕ parts, then the position of a particle in PSO has the form

$$\begin{aligned}\boldsymbol{x} = (&\psi_{g_1 1}, \psi_{g_1 2}, \ldots, \psi_{g_1 c_g}, \psi_{g_2 1}, \psi_{g_2 2}, \ldots, \psi_{g_2 c_g}, \\ &\ldots \psi_{g_\phi 1}, \psi_{g_\phi 2}, \ldots, \psi_{g_\phi c_g}),\end{aligned} \quad (11)$$

where $\psi_{g_i j}$ is the j^{th} coefficient of the i^{th} generating function, and c_g is the number of coefficients of g.

This approach was suggested as a way to solve a specific issue with AMPSO that was identified and discussed in [10], but has not yet been tested. Essentially, E-AMPSO ensures that all possible candidate solutions exist in the AMPSO search space, whereas this is not guaranteed when solving binary problems with more than 16 dimensions using a single generating function, assuming that equation (7) is used as the generating function.

VI. Frequency Distribution of Binary Solutions

In order for PSO to find good solutions, it is desirable that every potential candidate solution must exist in the search space with the same frequency. Without a uniform solution frequency distribution, PSO may not be able to find good solutions if they happen to be rare compared to bad solutions. When solving continuous problems, the solution frequency distribution is uniform, because every real number exists in $\mathbb{R}$ exactly once. However, when angle modulation is applied to PSO, the relationship between the continuous generating function and the binary solution space causes a non-uniform frequency distribution of binary solutions in the AMPSO search space. This is because there are more permutations of the coefficients of g than there are binary solutions. The result is that, for any given binary solution, there is very likely more than one permutation of the coefficients that will generate that same binary solution.

To illustrate, consider an arbitrary 8-bit binary problem being optimised with AMPSO. Any given permutation of the coefficients of g will result in a specific binary solution. The frequency distribution of binary solutions can then be seen by generating a solution for every possible permutation of the coefficients of g and counting the number of times that each unique solution is generated. Obviously this is not possible, because the coefficients of g are real values, which means that there is an infinite number of permutations. However, the frequency distribution of solutions in the AMPSO search space can be estimated by limiting the values that any coefficient may assume to some subset $\mathbb{C} \subseteq \mathbb{R}$. Figure 2 shows the frequency distribution of 8-bit binary solutions in the AMPSO search space when $\mathbb{C} = \{-1, -0.99, 0.98 \ldots 0.0, 0.01, 0.02, \ldots 1.0\}$. The range $[-1, 1]$ was chosen, because this is the range within which AMPSO is generally initialised [9], [13], [14]. The total number of permutations of the coefficients of g in this estimation is $|\mathbb{C}|^4 = 21^4 = 194,481$. The total number of 8-bit binary solutions is $2^8 = 256$. In figure 2, every value on the x-axis is an integer which represents the corresponding binary solution. For example, the value 0 corresponds to the binary string '00000000', the value 1 corresponds to '00000001', etc.

It is clear from figure 2 that—although every 8-bit binary solution exists—the frequency distribution of binary solutions in the AMPSO search space is not close to uniform. In fact, 20% of the search space is dominated by the '00000000' and '11111111' solutions. A number of smaller spikes in the frequency distribution plot indicate that some solutions are much more common than others. The twelve most common solutions (not counting the two solutions mentioned above) are listed below:

$$\begin{aligned}
4_{10} &= 00000100_2,\\
8_{10} &= 00001000_2,\\
16_{10} &= 00010000_2,\\
32_{10} &= 00100000_2,\\
64_{10} &= 01000000_2,\\
127_{10} &= 01111111_2,\\
128_{10} &= 10000000_2,\\
191_{10} &= 10111111_2,\\
223_{10} &= 11011111_2,\\
239_{10} &= 11101111_2,\\
247_{10} &= 11110111_2,\\
251_{10} &= 11111011_2.
\end{aligned}$$

The two least common solutions are:

$$\begin{aligned}
71_{10} &= 01000111_2, \text{ and}\\
184_{10} &= 10111000_2.
\end{aligned}$$

It is obvious from the above observations that the most common solutions in the AMPSO search space are the ones that contain a lot of repetition. This trend is also true for binary solutions of higher dimensionality, although the frequency distribution plots become difficult to read because of the exponential growth in the number of binary solutions.

The cause of this trend is the generating function. Regularities in the shape of the generating function mean that some solutions are much easier to generate than others by varying the values of the coefficients. The solutions that are easy to generate may overshadow good solutions in the search space. Furthermore, common solutions may cause plateaus in the fitness landscape, thereby complicating the search even more.

This problem can potentially be alleviated by using a different generating function. However, it is difficult to imagine a generating function that would produce a uniform solution frequency distribution. For this reason, a simpler approach might be to replace angle modulation with a different method that naturally produces a uniform search space.

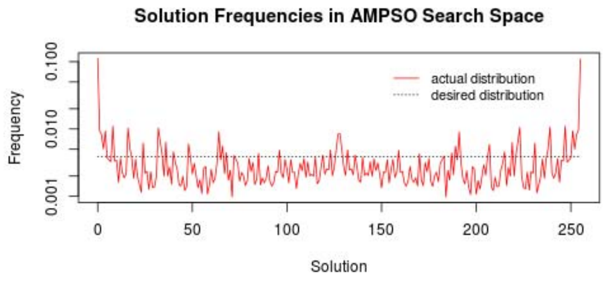


Fig. 2. An estimation of the frequency distribution of binary solutions in the AMPSO search space, for 8-dimensional binary problems.

VII. Obtaining A Uniform Solution Frequency Distribution with Natural Numbers

This section details a new approach to solving binary optimisation problems with PSO. The method is explained in section VII-A, while two potential problems with the new approach are addressed in sections VII-B and VII-C.

A. *PSO with Natural Numbers*

To obtain a uniform solution frequency distribution, some uniform mapping from $\mathbb{R}^{n_x}$ to $\mathbb{B}^{n_b}$ is required. That is, a mapping $\mathcal{M} : \mathbb{R}^{n_x} \to \mathbb{B}^{n_b}$ is needed, such that for every $\mathcal{B} \in \mathbb{B}^{n_b}$, there is an equal number of values $\boldsymbol{x} \in \mathbb{R}^{n_x}$ that map to $\mathcal{B}$.

Consider the set of natural numbers $\mathbb{N}^0 = \{0, 1, 2, \ldots\}$. Every element in $\mathbb{N}^0$ can be mapped to a binary value by converting the element from base-10 to base-2. Furthermore, the mapping is a one-to-one mapping, meaning that is always uniform. Now, clearly $\mathbb{N}^0 \neq \mathbb{R}$. However, the floor $\lfloor x \rfloor$ of every $x \in \mathbb{R}_{\geq 0}$ is a natural number. Therefore, PSO can be used to find a real value $x \in \mathbb{R}_{\geq 0}$ such that $\lfloor x \rfloor$ is the natural number that maps to the desired binary solution. Furthermore, to ensure that a mapping exists from $\mathcal{S}$ to every element $\mathcal{B} \in \mathbb{B}^{n_b}$, only a subset of $\mathbb{R}_{\geq 0}$ is required, such that $\mathcal{S} \subseteq \mathbb{R}_{\geq 0} = [0, 2^{n_b})$, and $n_x = 1$.

For example, if $n_b = 3$, then there are $2^3 = 8$ candidate binary solutions, which map to the following eight natural numbers: $\{0, 1, 2, 3, 4, 5, 6, 7\}$. Therefore, the continuous search space $\mathcal{S}$, whose floored elements are equal these eight natural numbers, is $\mathcal{S} = [0, 8)$. A one-dimensional PSO can now be used to search $\mathcal{S}$ for the natural number that maps to the optimal three-bit binary solution. This technique will henceforth be referred to as the *natural numbers PSO* (NNPSO). Note that the frequency distribution of binary solutions in $\mathcal{S}$ is naturally uniform in the case of NNPSO.

Because NNPSO guarantees that all the possible candidate solutions are contained in $\mathcal{S} = [0, 2^{n_b})$, particles in NNPSO should not update their personal best positions unless they are within these bounds.

B. *Hamming Cliffs*

An immediate point of concern in NNPSO is that similar solutions in $\mathbb{R}$ could lead to dissimilar solutions in $\mathbb{B}^{n_b}$. For example, assume that $n_b = 3$ and, consequently, $\mathcal{S} = [0, 8)$. Now consider the values $3.5 \in \mathcal{S}$ and $4.2 \in \mathcal{S}$. To map these values to binary solutions, the floor function is computed before converting the value to base-2. That is,

$$\begin{aligned}\lfloor 3.5 \rfloor &= 3_{10} \rightarrow 011_2, \text{ and}\\ \lfloor 4.2 \rfloor &= 4_{10} \rightarrow 100_2.\end{aligned} \tag{12}$$

The values 3.5 and 4.2 produce consecutive natural numbers, but the hamming distance between the two binary solutions is 3. This is clearly not desirable, because it means that similar solution are not grouped together in the search space, which is a fundamental assumption that PSO exploits to perform optimisation. Fortunately, this problem is easily addressed by using a gray code conversion, instead of a binary conversion. Gray code ensures that consecutive values in $\mathbb{N}^0$ always produce binary strings with a hamming distance of 1. In the case above, gray code conversion produces the following binary strings:

$$\begin{aligned}\lfloor 3.5 \rfloor &= 3_{10} \rightarrow 010, \text{ and}\\ \lfloor 4.2 \rfloor &= 4_{10} \rightarrow 110.\end{aligned} \tag{13}$$

C. Search Space Explosion

Another problem with NNPSO is that the size of the search space $\mathcal{S}$ grows exponentially with n_b. Recall from section VII-A that $\mathcal{S} = [0, 2^{n_b})$. The effect is that the search space quickly becomes so large that one would need a huge number of particles in the swarm in order to get close to a uniform coverage of the search space during swarm initialisation. As a result, PSO becomes ineffective when solving large binary problems with NNPSO.

One potential (although not full-proof) way to address this problem is to divide the binary problem into separate, smaller problems, at the expense of increased particle dimensions. For example, an 8-dimensional binary problem can be split into two four-dimensional binary problems. Then, NNPSO would be used to find *two* natural numbers $\lfloor x_1 \rfloor$ and $\lfloor x_2 \rfloor$ whose gray codes can be concatenated to produce one 8-dimensional binary solution. This obviously implies that particles in NNPSO are now two-dimensional. That is, where it used to be the case that $\mathcal{S} = [0, 256)^1$, now $\mathcal{S} = [0, 16)^2$. This approach will henceforth be referred to as *ensemble NNPSO* (E-NNPSO).

To formalise, the E-NNPSO algorithm can be used to solve n_b-dimensional binary problems in a ϕ-dimensional real-valued search space $\mathcal{S} = [0, 2^{n_b/\phi})^\phi$. As mentioned above, this approach is not full-proof, because the search space still grows exponentially, albeit not in a single dimension. The hope is that PSO will be able to cope better with the growth if the domain can be kept relatively small in each dimension.

VIII. Experiments

This section details the experimental procedure that was followed for this study.

For the purpose of this study, the following algorithms were tested: AMPSO, A-AMPSO, MM-AMPSO, E-AMPSO, E-NNPSO, and BPSO. The NNPSO algorithm was ommited, because the size of the search domain in a single dimension became so large that overflows occurred in extreme cases. The algorithm is therefore deemed impractical. In all cases, algorithms executed for 1000 iterations, and the global best fitness was recorded at every iteration. All reported results are averaged over 30 independent runs.

Three binary problems were considered in this study with varying dimensionality. Each problem—together with the algorithmic setup for that problem—is discussed in its own subsection below.

A. N-Queens

The N-Queens problem is a chess board problem where the goal is to place n queens on an $n \times n$ chess board B in such a way that no queen threatens any other queen, by the standard rules of chess.

For this study, a solution to the n-queens problem was encoded as a binary string $\mathcal{B}$ of length $\lceil \log_2 n \rceil n$. Each $\lceil \log_2 n \rceil$-bit group in $\mathcal{B}$ represents a column on the chess board, with the decimal value of the bits indicating the row number for the queen in that column. Now, every square B_{ij} on the board can be assigned a value '0' if it is empty, or '1' if it contains a queen. To calculate the fitness of a given solution, the following objective function was used:

$$f(\mathcal{B}) = [k + (n - \mathcal{Q})^2] \times (|T_{n-1} - R| + 1), \tag{14}$$

where k is the total number of conflicts between queens on the board, $\mathcal{Q}$ is the number of queens on the board,

$$T_n = \frac{n(n+1)}{2} \tag{15}$$

is the n^{th} triangular number, and

$$R = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} r_{ij}, \tag{16}$$

where

$$r_{ij} = \begin{cases} i & \text{if } B_{ij} = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

Note that $R = T_{n-1}$ iff every row i on the chess board contains exactly one queen. Therefore, the term $|T_{n-1} - R| + 1$ penalises the objective function if the queens are not properly spread out on the chess board.

All algorithms were compared on the N-Queens problem, with n = 8, 15, and 25. In the case of E-AMPSO and E-NNPSO, the problem was divided into n parts. That is, E-AMPSO optimised n generating functions to find the position of the queen in each column, and E-NNPSO searched for n natural numbers that mapped to the position of the queen in each column.

B. Knights' Coverage

Th knights' coverage problem is another chess board problem. In this case, the aim is to place the minimum number of knights on the board such that every square on the board is *covered*. A square is considered to be covered when either of the following two conditions hold: 1) the square contains a knight, or 2) at least one knight can move to the square with a single move, according to the standard rules of chess.

For this problem, the solution was again encoded as an n^2-dimensional bit string. Each bit in $\mathcal{B}$ represented a square on the chess board. A ‘1’ bit indicated the presence of a knight, while a ‘0’ bit indicated an empty square. The following objective function was used:

$$f(\mathcal{B}) = \frac{\mathcal{K}}{\mathcal{C}+1} + \frac{n^2 - \mathcal{C}}{\mathcal{K}+1}, \tag{18}$$

where $\mathcal{K}$ is the number of knights on the board, and $\mathcal{C}$ is the number of covered squares on the board.

The knights’ coverage problem was tested for board sizes of $n = 8$, 15, and 25. For E-AMPSO and E-NNPSO, the problem was divided into n parts, as was the case for the N-Queens problem. However, in the case of the knights’ coverage problem, each column may contain more than one knight.

C. *Random Bit String Matching*

The final problem that was considered in this study is the problem of matching random bit strings. Random target bit strings of length 50, 200, and 300 were generated. The objective in each case was to find a binary solution $\mathcal{B}$, such that the hamming distance between $\mathcal{B}$ and the target bit string was minimised.

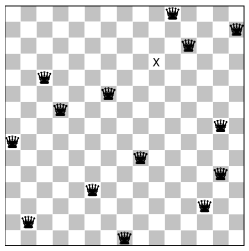
In the case of E-AMPSO and E-NNPSO, the problem was divided into 5-bit parts in all cases.

IX. Results and Discussion

Figures 4 to 6 show the average fitness profiles for all algorithms the N-Queens problems. In all three problem cases, E-NNPSO obtained a much lower average fitness than all the other algorithms. To understand why E-NNPSO obtained the lowest average, it is helpful to visualise a solution to the N-Queens problem. Figure 3(a) shows the best solution found by the E-NNPSO algorithm on the N-Queens problem with $n = 15$. The solution is not complete. An “X” in the figure shows the location where the final queen could have been placed. However, a queen in that position would cause a conflict with another queen to the right and down. Note that there is no obvious repeating pattern in this solution to the N-Queens problem. While the solution in figure 3(a) is not optimal, it resembles the typical non-repetitive pattern that solutions to the N-Queens problem exhibit—at least, when they are encoded as binary strings.

Recall from section VII-A that candidate solutions that contain repetition are relatively common in the AMPSO search space. Therefore, AMPSO and all of its variants have a very hard time in finding those solutions and tend to converge to sub-optimal solutions instead. Essentially, the AMPSO algorithms already fail during the exploration phase of the search. The same argument explains why BPSO obtained a lower average than AMPSO. However, the lack of sensible guides in BPSO ultimately means that the algorithm cannot effectively exploit the search space.

The fitness profiles for the Knights’ Coverage problems are shown in figures 7 to 9. For this problem, the MM-AMPSO and E-AMPSO algorithms obtained the lowest average fitness in all cases. Again, a visualisation of the solution is helpful. Figure 3(b) shows the best solution obtained by E-AMPSO on the Knights’ Coverage problem, with $n = 15$. The repetitive pattern in the solution is immediately obvious. This solution



(a) E-NNPSO solution

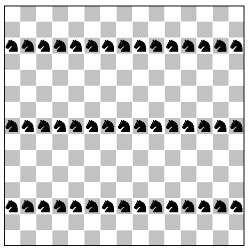

(b) E-AMPSO solution

Fig. 3. Figures (a) and (b) show the best solutions found for N-Queens and Knights’ Coverage, respectively. In both cases, $n = 15$. The “X” in figure (a) marks the location where the final queen should have been placed.

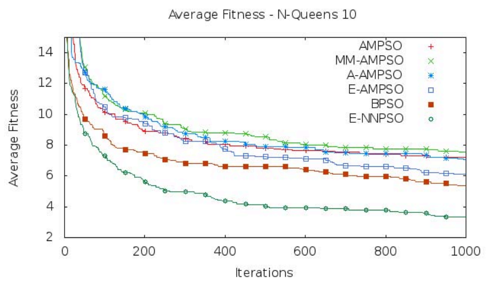


Fig. 4. Fitness profiles for N-Queens with $n = 10$.

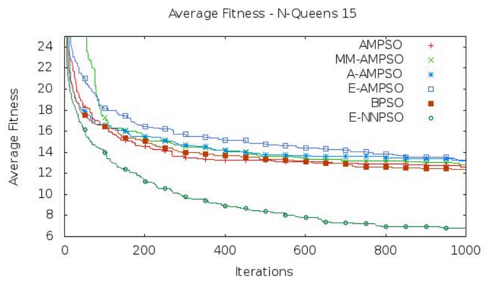


Fig. 5. Fitness profiles for N-Queens with $n = 15$.

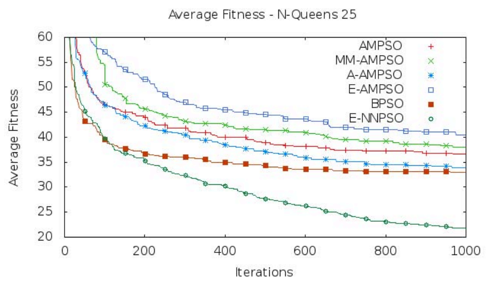


Fig. 6. Fitness profiles for N-Queens with $n = 25$.

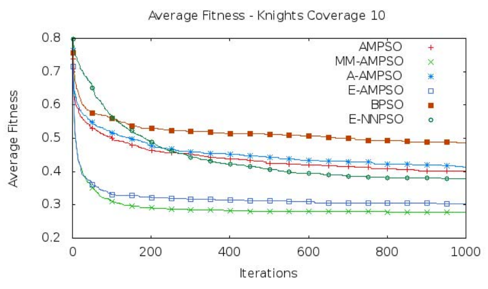


Fig. 7. Fitness profiles for Knights' Coverage with $n = 10$.

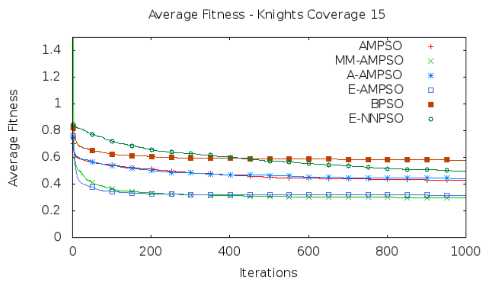


Fig. 8. Fitness profiles for Knights' Coverage with $n = 15$.

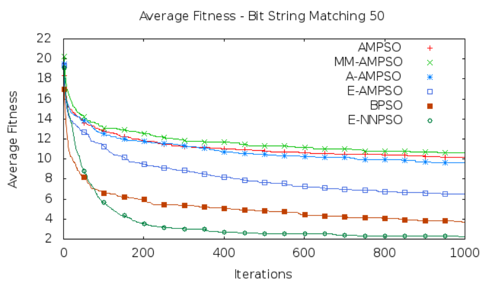


Fig. 10. Fitness profiles for Random Bit String Matching with $n_b = 50$.

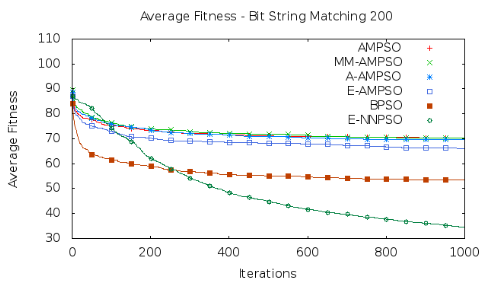


Fig. 11. Fitness profiles for Random Bit String Matching with $n_b = 200$.

also happens to be optimal. The AMPSO algorithms have an advantage on this problem, because the optimal solutions happen to be common in the AMPSO search space. E-AMPSO obtained a lower average fitness in all cases, because the algorithm breaks the problem up into individual columns. In that case, each column's solution is a bit-string of either only '1' bits, or only '0' bits. Recall from section VII-A that those solutions make up 20% of the search space. In the case of MM-AMPSO, the algorithm is able to isolate parts of the generating function that exhibit the required pattern and thus obtained a lower average fitness than other AMPSO or A-AMPSO.

Finaly, figures 10 to 12 show the fitness profiles for the bit string mapping problems. The AMPSO algorithms obtained the high average fitness values across all problem cases. This was expected, because random bit strings are not likely to contain patterns. E-NNPSO obtained the lowest average fitness on 50- and 200-dimensional problem cases. This was also expected for the reasons given above. However, for 300-dimensional problems E-NNPSO obtained the highest average fitness, while the lowest average fitness was obtained by BPSO. The reason for this dramatic change in the relative performance of E-NNPSO is not currently clear, but warrants further investigation.

X. Conclusion

This paper studied the frequency distribution of solutions in the angle modulated particle swarm optimisation (AMPSO) search space. In addition the effect that non-uniform frequency distributions have on algorithm performance was investigated. It was shown that candidate binary solutions are not uniformly distributed in the AMPSO search space. In fact, some individual solutions may occupy as much as 10% of the AMPSO search space.

A new technique was introduced to solve binary problems using particle swarm optimisation. The new technique makes use of the one-to-one mapping between the natural numbers and binary solutions to ensure that candidate solutions are uniformly distributed in the search space. The technique was named natural numbers PSO (NNPSO). An exponential growth in the size of the search domain makes NNPSO somewhat im-

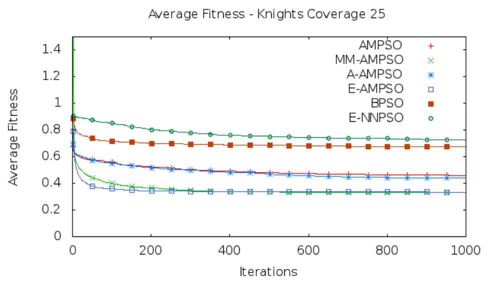


Fig. 9. Fitness profiles for Knights' Coverage with $n = 25$.

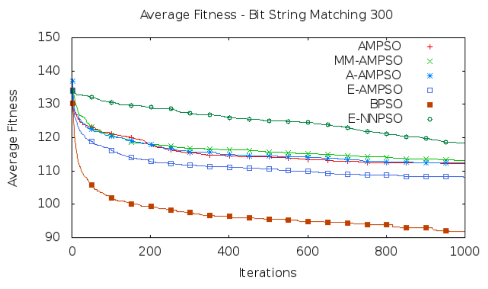


Fig. 12. Fitness profiles for Random Bit String Matching with $n_b = 300$.

practical. However, the problem can be managed by breaking the binary problem up into smaller problems at the expense of increased particle dimensionality in PSO. The resulting algorithm is called ensemble NNPSO (E-NNPSO).

Empirical results showed that the uniform frequency distribution of candidate solutions in the E-NNPSO search space is beneficial to algorithm performance in many problem cases. However, problems whose optimal solutions contain repetitive patterns grant a clear advantage to AMPSO algorithms, because those solutions are common the AMPSO search space.

It was observed that E-NNPSO performance suffers dramatically in the high-dimensional random binary string matching problem cases. The reason for this result is not currently understood. Further investigation into the use of E-NNPSO to solve binary problems is certainly warranted.

Future work will include studying the performance of E-NNPSO relative to other binary optimisation algorithms on a wider range of problems. A thorough scalability study—in terms of problem dimensionality—will also be performed.

REFERENCES

[1] C.W. Cleghorn and A.P. Engelbrecht. A generalized theoretical deterministic particle swarm model. *Swarm Intelligence*, 8(1):35–59, 2014.

[2] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43, 1995.

[3] R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 84–88. IEEE, 2000.

[4] A.P.P. Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2006.

[5] N. Franken. Pso-based coevolutionary game learning. Master's thesis, University of Pretoria, 2004.

[6] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[7] J. Kennedy and R. Eberhart. A discrete binary version of the particle swarm algorithm. In *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 5, pages 4104–4108. IEEE, 1997.

[8] M.A. Khanesar, M. Teshnehlab, and M.A. Shoorehdeli. A novel binary particle swarm optimization. In *Mediterranean Conference on Control & Automation*, pages 1–6. IEEE, 2007.

[9] B.J. Leonard and A.P. Engelbrecht. Angle modulated particle swarm variants. In *Swarm Intelligence*, volume 8667, pages 38–49, 2014.

[10] B.J. Leonard, A.P. Engelbrecht, and C.W. Cleghorn. Critical considerations on angle modulated particle swarm optimisers. *Swarm Intelligence*, submitted.

[11] L. Liu, W. Liu, D. Cartes, and I. Chung. Slow coherency and angle modulated particle swarm optimization based islanding of large-scale power systems. *Advanced Engineering Informatics*, 23(1):45–56, 2009.

[12] W. Liu, L. Liu, D. Cartes, et al. Angle modulated particle swarm optimization based defensive islanding of large scale power systems. In *Power Engineering Society Conference and Exposition in Africa*, pages 1–8. IEEE, 2007.

[13] G. Pampara. Angle modulated population based algorithms to solve binary problems. Master's thesis, University of Pretoria, 2013.

[14] G. Pampara, N. Franken, and A.P. Engelbrecht. Combining particle swarm optimisation with angle modulation to solve binary problems. In *IEEE Congress on Evolutionary Computation, 2005*, volume 1, pages 89–96, 2005.

[15] R. Poli. Mean and variance of the sampling distribution of particle swarm optimizers during stagnation. *IEEE Transactions on Evolutionary Computation*, 13(4):712–721, 2009.

[16] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 69–73. IEEE, 2002.

[17] F. Van den Bergh and A.P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937 – 971, 2006.