Combining Particle Swarm Optimisation with angle modulation to solve binary problems

Gary Pampara
Department of Computer Science
University of Pretoria
South Africa
gpampara@cs.up.ac.za

Nelis Franken
Department of Computer Science
University of Pretoria
South Africa
nfranken@cs.up.ac.za

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

Abstract- The optimisation process of a particular problem generally has many influencing factors including the parameter choices, problem constraints as well as the complexity of the optimistion algorithm and optimisation problem among others. The dimensionality of a problem influences the computational complexity in converging to a valid solution. With problems defined in larger and more abstract dimensions, complexity becomes a problem as the solutions presented by the algorithm are more likely to be sub-optimal. An interesting and unique manner to reduce the complexity of binary problems is developed in this paper: Angle modulation is applied to generate a bit string to solve binary problems, using Particle Swarm Optimisation (PSO) to evolve the function coefficients of a trigonometric model. Instead of evolving a high dimensional bit vector, angle modulation reduces the problem to a four-dimensional problem defined in continuous space. Experimental results show that the angle modulation method is faster than the standard Binary PSO, and that accuracy is improved for most benchmark functions used.

1 Introduction

Particle Swarm Optimisation (PSO) has been used in a variety of applications to a large degree of success. Kennedy and Eberhart [4] adapted the standard continuous PSO algorithm to be applied to binary spaces.

Since there is a direct link between the size of a particle and the dimension of the problem being optimised, an increase in problem dimension usually has a negative effect on the performance of the optimisation algorithm: Higher dimensional problems usually have more convoluted search spaces, where particles may become trapped more easily in local minima.

It stands to reason that a valid lower-dimensional representation of the problem that does not inhibit exploration of the original higher-dimensional search space, will allow for faster optimisation and potentially better solutions.

This paper presents a technique that allows for the representation of high-dimensional bit strings by optimising low-dimensional continuous-valued coefficients for a simple trigonometric function to perform angle modulation. Four coefficients are evolved for a trigonometric function. Using the evolved coefficients and trigonometric function, a bit string is generated as a solution to the original higher-dimensional problem. The task of finding an optimal solution to a binary optimisation problem is therefor reduced to a four-dimensional continuous optimisation problem, with a direct mapping back to binary space.

The remainder of the paper is organised as follows: Section 2 provides an overview of relevant existing PSO algorithms that will be used for the experimental work of this study. The various core PSO parameters are introduced and guidelines to parameter selection is given. Section 3 introduces the use of angle modulation to produce high-dimensional bit strings. Section 4 discusses the experimental procedure. Section 5 presents the results of the AMPSO and compares its performance against other PSO algorithms. The paper concludes in section 6 with a summary of the results and a brief look at future work resulting from this study.

2 Background

This section introduces the PSO algorithm and a variant, the Binary PSO. Performance criteria of the algorithms is discussed by considering the effect of algorithmic parameters. Particle organisation and knowledge sharing through particle neighbourhoods is also discussed. The algorithmic dif-

0-7803-9363-5/05/$20.00 ©2005 IEEE.

ferences between the continuous valued PSO and the Binary PSO is also introduced.

2.1 PSO

The PSO algorithm was introduced by Kennedy and Eberhart [3] in 1995. The algorithm is modelled on the manner in which a flock of birds move through the air with various individuals leading the flock during its motion for different periods of time. The PSO algorithm consists of a group of individuals named "particles". Each particle is a potential solution to the n-dimensional optimisation problem. A particle, i, has a memory to record the current solution $\mathbf{x}_i$, the previous personal best solution $\mathbf{y}_i$ and a velocity $\mathbf{v}_i$, that represents the speed at which the particle is "flying" through the current search space.

The algorithm operates by each particle determining its fitness and then sharing the information to the rest of the particles within the particle's neighbourhood. The neighbourhood best particle is denoted by $\hat{\mathbf{y}}_i$. This social information sharing causes a change in the direction vector, directing the particle towards more optimal solutions in the search space. This process continues until some stopping condition is met.

2.1.1 Acceleration and Inertia constants

The standard velocity update equation,

$$\begin{aligned} v_{ij}(t+1) &= \phi \cdot v_{ij}(t) \\ &\quad + c_1 r_1 (y_{ij}(t) - x_{ij}(t)) \\ &\quad + c_2 r_2 (\hat{y}_{ij}(t) - x_{ij}(t)) \end{aligned} \quad (1)$$

uses the acceleration constants c_1 and c_2, together with $r_1, r_2 \sim U(0,1)$ to define the step size of the particle. Within equation (1), c_1 and c_2 determine the influence of the personal best and the neighbourhood best solutions to the particle's current velocity vector. The cognitive component is defined to be the influence of the personal best position vector. If the cognitive component has a relatively large value, the flight of the particle will be steered in the direction of the personal best position - limiting the influence of the social component. The social component is defined to be the stochastically weighted difference between the neighbourhood best position and the current position.

The inertia parameter [8], ϕ, controls the influence of the previous velocity on the current velocity. Without the inertia parameter, the particle erratically jumps around in the search space making it more difficult for the algorithm to converge.

The inertia parameter can drastically improve the manner in which the PSO algorithm converges to a solution by smoothing particle trajectories. Van den Bergh [9] showed that for convergence to occur, the following inequality should hold:

$$\phi > \tfrac{1}{2}(c_1 + c_2) - 1 \text{ and } \phi < 1$$

where c_1 and c_2 are the acceleration constants in the velocity update equation.

2.1.2 $\mathbf{V_{max}}$

A restriction [8] may be placed on the individual velocity component values to enforce that a particle does not exceed a certain acceleration in either the positive or the negative direction. This constraint (referred to as the clamping of the velocity component) reduces the chance that a particle may accelerate uncontrollably and explode off the bounds of the search space. Velocity values that are not within the range $[-V_{max}, V_{max}]$ are clamped. Care should be taken in selection of the value of V_{max}. If it is too small, step sizes are very small which may cause the algorithm to be trapped in a local minimum, and may increase the number of iterations to reach a good solution.

2.1.3 Neighbourhoods

Various types of neighbourhood structures exist for the PSO algorithm. The neighbourhood is a structure that enables the social information sharing of the particles within the swarm [5].

The global best (*GBest*) topology defines a fully connected graph where every particle is connected to every other particle. Due to the interconnection between the particles, the *GBest* neighbourhood topology frequently causes the particles to converge on a sub-optimal solution. The current global best will influence all the particles in the population.

The local best (*LBest*) topology can be described as a one-dimensional lattice where all the particles are aligned to have neighbours to the left and to the right. The *LBest* topology defines neighbours based on indices in variable space and particles are influenced by the neighbourhood best particle. Information is shared between the neighbourhoods by having the neighbourhoods overlap with each other, permitting particles to be

members of more than one neighbourhood. *LBest* causes social information to be propagated in a rather slow, but effective, fashion. *LBest* promotes better exploration of the problem space and does, in general, give better results when compared to the *GBest* topology [6].

The *Von Neumann* topology arranges particles in a two-dimensional lattice enabling the particle to share information with more particles and in more directions. When compared to the other topologies, the benefit from additional social information improves the probability of converging on a good solution. The *Von Neumann* topology has experimentally been shown to be at least as effective as the *LBest* topology [5], even outperforming it on various occasions.

2.2 Binary PSO

The Binary PSO algorithm (BinPSO) was introduced by Kennedy and Eberhart [4] to allow the PSO algorithm to operate in binary problem spaces. The BinPSO has a structure almost identical to the standard PSO, where the velocity is still defined to be in the continuous space. However, the BinPSO does not regard the velocities as velocities, but rather uses it to define the probability that a bit flip will occur. The only changes to the standard PSO algorithm is that the position vector of the particle is a vector of binary digits, rather than a vector of continuous values, and that the position update equation changes to:

$$x_{ij}(t+1) = \begin{cases} 0 & \text{if } r_i(t) \geq f(v_{ij}(t)) \\ 1 & \text{if } r_i(t) < f(v_{ij}(t)) \end{cases} \quad (2)$$

where

$$f(v_{ij}(t)) = \frac{1}{1+e^{-v_{ij}(t)}} \quad (3)$$

and $x_{ij}(t)$ is the j-th position value of the i-th particle at time step t, $v_{ij}(t)$ is the corresponding velocity, and $r_{ij}(t) \sim U(0,1)$.

The BinPSO is susceptible to sigmoid function saturation, which occurs when velocity values are either too large or too small. In such cases the probability of a change in bit value approaches zero, thereby limiting exploration. For a velocity of 0, the sigmoid function returns a probability of 0.5, implying that there is a 50% chance for the bit to flip. Velocity clamping will delay the sigmoid function saturation from occurring.

3 Angle Modulated PSO

The Angle Modulated PSO (AMPSO) is a PSO algorithm that employs a trigonometric function as a bit string generator. The function is derived from a technique used in the field of signal processing from the telecommunications industry and is based on *angle modulation* [7]. It was first applied in PSO to represent strategies in the Iterated Prisoner's Dilemma [2]. The technique uses a composed sin/cos generating function:

$$g(x) = sin(2\pi(x-a) \times b \times cos(A)) + d \quad (4)$$

where

$$A = 2\pi \times c(x-a)$$

and x is a single element from a set of evenly separated intervals determined by the number of bits specified to be generated. For example, using 10 intervals between 0 and 10. The coefficient a represents the horizontal shift of the function, b represents the maximum frequency of the sin function, c represents the frequency of the cos function (which changes the rate at which the frequency of the sin function changes) and d represents the vertical shift of the function. The function with default coefficient parameters $(a = 0, b = 1, c = 1, d = 0)$ is depicted in figure 1.

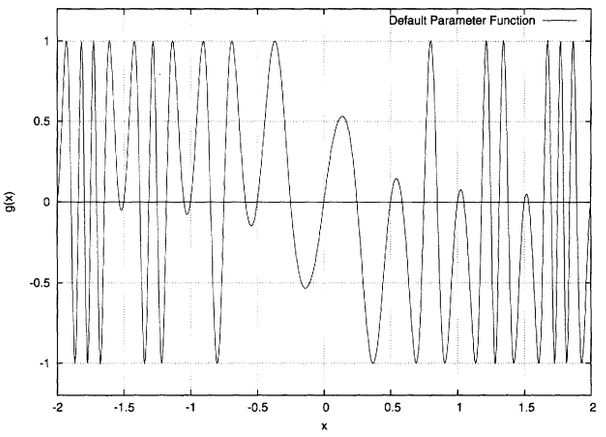

Figure 1: Fitness function plotted with default values $(a = 0, b = 1, c = 1, d = 0)$

The standard PSO algorithm is applied to optimise a simpler 4-dimensional tuple (a, b, c, d) representing the parameters of equation (4) instead of evolving the actual bit string. After the PSO iteration, the parameters are substituted back into equation (4). The resultant function is then sampled at the evenly spaced intervals to generate a

bit for each interval, with the set of all generated bits representing the binary vector solution to the original problem.

Generation of the bit string is a simple procedure that involves taking the sampled points, feeding them into the equation and evaluating the output of equation (4). If the output value is positive, the corresponding bit value is noted as bit 1, else it is noted as bit 0.

The benefit of the AMPSO is that a larger dimensional binary space can be represented by a smaller 4-dimensional continuous space. Results from the optimisation are produced in a shorter period of time, as only 4 parameters need to be optimised instead of the original n-dimensions.

Other strategies to handle high-dimensional problems do exist. For example, the co-operative PSO [10] follows a "divide and conquer" approach whereby the particle is broken up into smaller pieces, optimised separately and later recombined to form the original particle. Applying the co-operative PSO to a complex binary problem makes little sense as within the binary strings the bits have an interdependency that must be maintained. The performance of the co-operative PSO has been shown to deteriorate when a dependence among variables exist [9]. Therefor, the co-operative PSO does not perform well for such binary problems.

Section 5 experimentally compares the AMPSO approach to the BinPSO and standard continuous PSO algorithms on a set of well-known benchmark functions.

4 Experimental Approach

Based on previous experimental work performed by van den Bergh [9], values of 1.496180 have been selected for both the c_1 and the c_2 constants for the purposes of this paper, as well as selecting the inertia parameter to be a value of 0.729844.

A number of experiments were conducted to find the best value for V_{max} and the best neighbourhood topology. Based on these results (not provided here due to space limitations), V_{max} was set to 4.0 and the *Von Neumann* topology was selected. The number of particles used in the experiments has been selected to be 40. Each experiment was conducted 30 times, in order to report statistically relevant results. A confidence interval of 95% was measured.

4.1 Benchmark functions

The specific benchmark functions used in the following experiments were selected due to their use in previous publications, as well as being functions that are robust enough to describe a wide selection of problem conditions.

The functions have been implemented within the Computational Intelligence Library (CIlib) framework (`http://cilib.sourceforge.net`) and all experiments have been performed using the library simulator.

Ten benchmark functions have been used for this paper. The first 5 functions were taken from [4], and are adaptions of De Jong's functions [1]. These functions, summarised in table 1, were maximised (the results are discussed in Section 5). The last 5 functions, are listed in table 2. Both the functions in table 1 and 2 were minimised.

For the maximisation problems, the results of AMPSO are compared against that of the BinPSO. For the minimisation problems, AMPSO is compared against the minimisation results of the continuous valued PSO and the BinPSO.

Table 3 summarises characteristics of the functions. To test the AMPSO on problems of varying dimensions, problems of low dimension and others with high dimension have been used.

For each particle the four parameters of equation (4) were randomly initialised within the interval [-1000.0, 1000.0]. Initialisation of these parameters to this domain ensured greater variation in the generating functions obtained during the PSO optimistaion process.

Each benchmark function is defined to operate in n dimensions. Each variable is represented by m bits. The size of m is dependant on the domain of the function, and the required decimal accuracy after conversion. For the BinPSO, the particle represents a $n \times m$ bit vector. For the AMPSO, the particle contains four coefficients (a, b, c, d) which are

Table 1: De Jong's Functions

f_1	$f_1(\mathbf{x}) = \Sigma_{i=1}^{n} x_i^2$
f_2	$f_2(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$
f_3	$f_3(\mathbf{x}) = 6 \cdot \Sigma_{i=0}^{5} \lfloor x_i \rfloor$
f_4	$f_4(\mathbf{x}) = \Sigma_{i=1}^{n} i \cdot x_i^4 + U(0,1)$
f_5	$f_5(\mathbf{x}) = \frac{1}{0.002} + \Sigma_{j=1}^{25} \frac{1}{j + \Sigma_{i=1}^{2}(x_i - a_{ij})^6}$ $a_{ij} = \begin{bmatrix} -32.0, -16.0, 0.0, 16.0, 32.0 \\ -32.0, -16.0, 0.0, 16.0, 32.0 \end{bmatrix}$

Table 2: Multi-modal benchmark functions

f_6	$f_6(x) = 0.5 + \frac{(sin^2\sqrt{x^2+y^2}) - 0.5}{(1.0 + 0.001(x^2+y^2))^2}$
f_7	$f_7(x) = \frac{1}{4000}\Sigma_{i=1}^{n}(x_i - 100)^2 - \Pi_{i=1}^{n} cos(\frac{x_i - 100}{\sqrt{i}}) + 1$
f_8	$f_8(x) = -20 \cdot \exp(-0.2\sqrt{\frac{1}{n} \cdot \Sigma_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\Sigma_{i=1}^{n} cos(2\pi \cdot x_i)) + 20 + e$
f_9	$f_9(x) = \Sigma_{i=1}^{n} 100((x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$
f_{10}	$f_{10}(x) = \Sigma_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i) + 10)$

substituted into equation 4 to produce the $n \times m$ bit vector. The bit vector is converted into n real numbers in accordance to the reference implementation of Spears[1]. These real numbers are then used as input to the benchmark functions to determine particle fitness.

5 Results

Experimental results are discussed separately with respect to the maximisation and the minimisation problems, respectively in Sections 5.1 and 5.2. The results are tabulated in table 4 which is divided into sections for minimisation and maximisation. Results for the maximisation problems are compared against the results of the BinPSO. The experimental conditions for the simulations of this paper were kept the same as those of [4].

Table 5 summarises the number of iterations required by the different algorithms before convergence occurs.

5.1 Maximisation Results

Table 4 lists the results for the maximisation functions f_1 to f_5, compared to the results of the BinPSO. The experimental results of the BinPSO are similar to the results of the BinPSO obtained by Kennedy and Eberhart [4].

Using the AMPSO for function f_1, a consistent value of 78.64320000000001 was reached, from the very first iteration, whereas the BinPSO required many more iterations to achieve the target result (refer to table 5). The trailing decimals can be attributed to the manner in which the continuous number is represented as the number of bits used influences the precision of the continuous number.

According to Kennedy and Eberhart [4], f_2 is the most difficult function (from f_1 to f_5) to optimise, with a target value of 3905.93. The BinPSO managed to solve this problem effectively and returned a best evaluation of exactly 3905.93 in the performed experiments. The AMPSO also obtained the same value, however, with significantly fewer iterations.

f_3 is defined to be an integer function (due to the use of the mathematical floor function) with a target value of 55.0. Again, the AMPSO solved the problem, converging onto the solution from the very first iteration in all experiments performed. The BinPSO achieved the correct result as well, but using significantly more iterations. Figure 2 demonstrates the learning profile of the AMPSO and the BinPSO. It is clear that the BinPSO required more function evaluations before convergence onto the target value.

f_4 is irregular due to the uniform noise added to the function. The AMPSO is affected by this noise, but adequately compensated for it. The target solution is valued at 1248.2. The BinPSO experienced similar results with an average value of 1247.88 with far more iterations required. The AMPSO achieved the target result, within 20 iterations.

f_5 has a target value of 500.0. The BinPSO and the AMPSO managed to attain the value with little effort, both coming close to the goal solution, with results such as 499.9999999996911 for the AMPSO and a value of 499.9999980926696 for the BinPSO. The AMPSO achieved this results faster than the BinPSO.

5.2 Minimisation Results

All benchmark functions are defined to have the global minimum value of 0.0. The only exception is f_5 with a minimum of 1.0. Iterations were limited to 10000.

Function f_1 was simple to minimise for the AMPSO, providing a correct solution almost immediately. The BinPSO managed to achieve the

[1]Reference C source code provided by William Spears at http://www.cs.uwyo.edu/~wspears/functs.dejong.html

Table 3: Details of benchmark functions

Function	Dimension	Optimisation type	Common name	Function domain
f_1	3	Min & Max	Spherical	(-5.12, 5.12)
f_2	2	Min & Max	2D Rosenbrock	(-2.048, 2.048)
f_3	5	Min & Max	Step Function	(-5.12, 5.12)
f_4	30	Min & Max	Quadric	(-1.28, 1.28)
f_5	2	Min & Max	Foxholes	(-65536.0, 65536.0)
f_6	2	Min	Schaffer's F6	(-100.0, 100.0)
f_7	30	Min	Griewank	(-300.0, 300.0)
f_8	30	Min	Ackley	(-30.0, 30.0)
f_9	30	Min	Rosenbrock	(-2.048, 2.048)
f_{10}	30	Min	Rastrigin	(-5.12, 5.12)

target result using many more iterations. The results presented indicated that the AMPSO and the BinPSO outperform the continuous valued PSO.

f_2 illustrated that the AMPSO performs extremely well when compared to the BinPSO and the continuous PSO.

Function f_3 had the target minimum reached with the AMPSO, with the BinPSO almost achieving the goal. The continuous PSO performed poorly in comparison to AMPSO and BinPSO.

f_4 illustrates that the BinPSO and the continuous valued PSO can outperform the AMPSO.

Results for function f_5 indicate that the AMPSO obtained the target every time, with the BinPSO approaching the target. Both the AMPSO and the BinPSO could not gain the accuracy of the continuous valued PSO when optimising f_5.

For f_6 the obtained results, 0.024139 for the AMPSO, improve on the effectiveness of the BinPSO which returns a value of 0.073079. However, the results of the continuous valued PSO outperforms the AMPSO. The number of iterations required by the AMPSO is more than the iterations needed by the BinPSO.

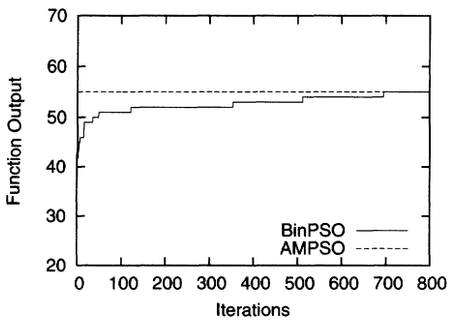

Figure 2: f_3: BinPSO vs AMPSO

The results for f_7 obtained by the AMPSO is worse than the value reported by the continuous valued PSO and the BinPSO. The cause of this result will be explored in future research.

The result obtained for f_8 is comparable to that of BinPSO, whilst the final result for the continuous valued PSO outperforms the results for both the AMPSO and BinPSO.

f_9 is a generalised version of f_2, and for our experiments with $n = 30$. The AMPSO produces results that are on average better than the BinPSO, however, the results obtained by the continuous valued PSO outperform the AMPSO results. Figure 3 illustrates the adjustments made by the AMPSO to converge to the target, compared to the BinPSO. Again, AMPSO was significantly faster than BinPSO.

For f_{10}, AMPSO performed moderately better than the BinPSO. The results obtained for the continuous valued PSO are significantly better than both the AMPSO and the BinPSO. Figure 4 illustrates the performance of the AMPSO and the BinPSO.

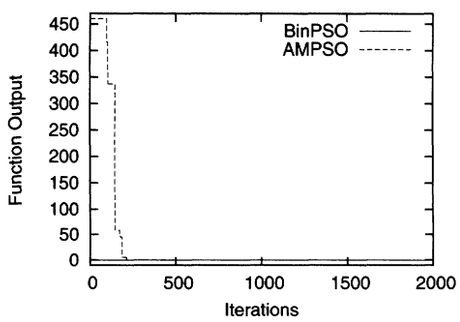

Figure 3: f_9: AMPSO vs BinPSO

Table 4: Comparison of the results (95% confidence)

Function	AMPSO	BinPSO	PSO ($V_{max} = 4.0$)
Maximisation			
f_1	78.643200 ± 0	78.599998	–
f_2	3905.926227 ± 3.047558	3905.93	–
f_3	55.0 ± 0	55.0	–
f_4	1247.629068 ± 0.99	1246.88	–
f_5	499.999999 ± 0	499.999999	–
Minimisation			
f_1	0.000300 ± 0	0.000300 ± 0	0.003027 ± 0.000467
f_2	0.000049 ± 0.000107	0.000142 ± 0.000088	0.002562 ± 0.000628
f_3	0.0 ± 0.0	0.000017 ± 0.000015	0.223188 ± 0.009976
f_4	41.469003 ± 1.922524	20.945429 ± 1.917975	0.233908 ± 0.060900
f_5	$5.25E-011 \pm 0$	$5.25E-011 \pm 9.65E-015$	$3.16E-030 \pm 0$
f_6	0.024139 ± 0.004212	0.073079 ± 0.010559	0.001976 ± 0.000419
f_7	106.118084 ± 4.398166	67.920632 ± 1.976655	2.643743 ± 0.145003
f_8	19.660458 ± 0.056733	20.109245 ± 0.048899	1.617440 ± 0.056456
f_9	$2198.368745 \pm 86.302836$	$2244.457155 \pm 77.423982$	125.378152 ± 6.400559
f_{10}	224.765413 ± 35.262816	308.256744 ± 2.813918	87.012601 ± 5.048148

Table 5: Iterations required for convergence

Function	AMPSO Iterations	BinPSO Iterations
Maximisation		
f_1	1.00	801.85
f_2	279.25	1153.71
f_3	1.00	1521.57
f_4	17.67	1017.88
f_5	7.82	107.64
Minimisation		
f_1	27.22	664.27
f_2	1611.78	2058.93
f_3	1.00	2083.20
f_4	812.49	1342.67
f_5	14.22	1018.66
f_6	7801.22	2875.59
f_7	1988.11	3955.87
f_8	827.12	5011.31
f_9	554.33	2054.28
f_{10}	457.18	4302.67

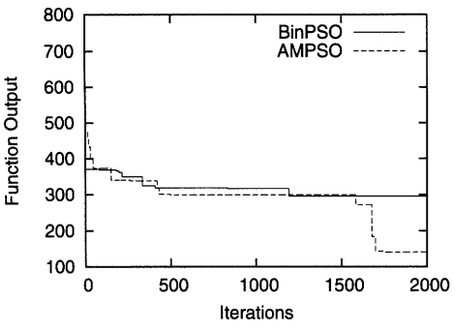

Figure 4: f_{10}: AMPSO vs BinPSO

6 Conclusion and future work

This paper proposes a new technique to optimise binary problems using a continuous valued representation and a generating function. The AMPSO is compared to the BinPSO and standard continuous valued PSO. Parameter choices are examined and experimental results are compared and tabulated.

From the experimental results, it is clear that the AMPSO produced favourable results. Examining table 4, the results of the AMPSO are relatively similar and in some cases better than the results obtained by the BinPSO and in Kennedy and Eberhart [4].

The main benefit of the AMPSO remains the fact that the particle becomes simpler in complexity as fewer dimensions need to be evolved for the problems defined in this paper. Far fewer iterations are required when using the AMPSO to achieve the same if not better accuracy than BinPSO (with the exception of one or two problems).

Further reseach will be performed in order to improve the AMPSO by considering special cases and determining the reasons for poor performance in certain applications.

Following on the game theory application of AMPSO in the IPD [2], other possible future uses may be found in applications to game theory. These include problems such as the El-farol bar problem and the minority game.

Since angle modulation is independent of the optimisation algorithm, the PSO algorithm may be substituted with the Genetic Algorithm (GA) in order to solve binary problems.

References

[1] K.A. De Jong. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[2] N. Franken. PSO-Based Coevolutionary Game Learning. Master's thesis, Department of Computer Science, University of Pretoria, South Africa, 2004.

[3] J. Kennedy and R.C. Eberhart. Particle Swarm Optimisation. In *Proceedings of the International Conference on Neural Networks*, pages 1942–1948, 1995.

[4] J. Kennedy and R.C. Eberhart. A Discrete Binary Version of the Particle Swarm Optimiser. In *IEEE International Conference on Computational Cybernetics and Simulation*, volume 5, pages 4104–4108, 1997.

[5] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1671–1676, May 2002.

[6] E.S. Peer, F. van den Bergh, and A.P. Engelbrecht. Using Neighborhoods with the Guaranteed Convergence PSO. *Proceedings of the 2003 Swarm Intelligence Symposium*, pages 235–242, 24–26 April 2003.

[7] J.G. Proakis and M. Salehi. *Communication System Engineering.* Prentice Hall Publishers, 2nd edition, 2002.

[8] Y. Shi and R.C. Eberhart. A Modified Particle Swarm Optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 69–73, May 1998.

[9] F. van den Bergh. *An Analysis of Particle Swarm Optimizers.* PhD thesis, Department of Computer Science, University of Pretoria, South Africa, 2002.

[10] F. van den Bergh and A.P. Engelbrecht. A Cooperative Approach to Particle Swarm Optimisation. In *IEEE Transactions on Evolutionary Computation*, volume 8, pages 225–239, 2004.