Enhanced Performance of Particle Swarm Optimization with Generalized Generation Gap Model with Parent-Centric Recombination Operator

Chukiat Worasucheep[1], **Charinrat Pipopwatthana**[2], **Sujitra Srimontha**[3], and **Wilasini Phanmak**[4], Non-members

ABSTRACT

Particle Swarm Optimization (PSO) algorithm has recently gained more attention in the global optimization research due to its simplicity and global search ability. This paper proposes a hybrid of PSO and Generalized Generation Gap model with Parent-Centric Recombination operator (G3PCX) [25], a well-known real-coded genetic algorithm. The proposed hybrid algorithm, namely PSPG, combines fast convergence and rotational invariance of G3PCX as well as global search ability of PSO. The performance of PSPG algorithm is evaluated using 8 widely-used nonlinear benchmark functions of 30 and 200 decision variables having different properties. The experiments study the effects of its new probability parameter Px and swarm size for optimizing those functions. The results are analyzed and compared with those from the Standard PSO [14] and G3PCX algorithms. The proposed PSPG with $Px = 0.10$ and 0.15 can outperform both algorithms with a statistical significance for most functions. In addition, the PSPG is not much sensitive to its swarm size as most PSO algorithms are. The best swarm sizes are 40 and 50 for unimodal and multimodal functions, respectively, of 30 decision variables.

Keywords: Particle Swarm Optimization, Real-Coded Genetic Algorithm, Parent-Centric Recombination, Hybrid algorithm, Optimization.

1. INTRODUCTION

Particle Swarm Optimization (PSO) , originally proposed by Eberhart and Kennedy, is a population-based algorithm for optimization over continuous search space [1]. Inspired by the social behavior of bird flocking, PSO consists of a swarm of particles, or a group of candidate solutions, each of which moves through the multi-dimensional search space. The position of each particle is updated with a velocity, which is constantly updated by the previous best performance of that particle and its neighbors. Due to its effectiveness and simple concept, PSO has become popular for solving nonlinear optimization problems in various domains, e.g. power system [2], economics [3], financial [4], and manufacturing [5]. Many attempts have been made to further improve the performance of original PSO, resulting in a number of well-known variants [6]-[12], including the Standard PSO 2007 (SPSO07 thereafter) [13][14]. Some variants are discussed in section 2.

The conventional PSO algorithms usually provide a good convergence to the search subspace that an optimum is located. However, the oscillating behavior of particles often prolongs the swarm from reaching such optimum [15]. In addition, the conventional PSO variants often face troubles with some complex problems because they are generally not rotational invariant, as all computations are performed coordinate by coordinate [16]. Also, the performance of conventional PSO deteriorates quickly due to degeneracy of particles' velocities when the number of decision variables is high [17]. As the result, several studies have incorporated some specialized operators of Evolutionary Algorithms (EAs) into PSO, in an attempt to improve its performance or to compensate the limitations of algorithm. Some successful hybridizations of PSO are as follows. In Takahama *et al.* [18]'s hybrid PSO and GA, the processes of PSO update and GA's operations were performed in each iteration. Kao *et al.* [19] proposed a method of running genetic operations and particle swarm operations to each halve of the population in each generation. Cai and Wunsch [20] introduced the PSO's velocity update into EAs, while Esmin *et al.* [21] introduced the mutation operator into PSO. Chiam *et al.* [4] proposed the implementation of PSO as a local optimizer in evolutionary search for training neural networks. Many hybrids of PSO with other population-based algorithms also exist such as PSO with Ant Colony Optimization [22], PSO with Extremal Optimization [23], PSO with Differential Evolution [24], etc.

In this work, we propose the hybrid of PSO

Manuscript received on September 1, 2011 ; revised on January 31, 2012.

[1,2,3,4] The authors are with Applied Computer Science program, Department of Mathematics, Faculty of Science, King Monkut's University of Technology Thonburi, Bangkok, Thailand, E-mail: chukiat.wor@kmutt.ac.th, charinrat_p@hotmail.com, susiexx56@hotmail.com and wilasinip@gmail.com

and Generalized Generation Gap model with Parent-Centric Recombination operator (G3PCX) evolutionary algorithm [25]. G3PCX is a well-known rotational invariant Real-Coded Genetic Algorithms (RCGA) [26]. G3PCX has shown its superior performance over many other real-valued Evolutionary Algorithms [25]. However, G3PCX is occasionally prone to get trapped in local optima in some complex functions. For this reason, this paper focuses on enhancing the performance of conventional PSO algorithm with G3PCX. The goal is to take advantage of rotational invariance capability and the convergence speed of G3PCX while maintaining the global diversity with conventional PSO. The proposed hybrid algorithm, namely *PSPG*, blends G3PCX into the particles framework of PSO. In each generation, the particle's position is updated either by PSO's particle movement equations or G3PCX's recombination operator, depending on a new probability parameter. This predefined probability balances between the exploration ability of PSO and the exploitation ability of G3PCX. The performance of proposed hybrid algorithm is evaluated using 8 widely-used scalable nonlinear benchmark functions of 30 and 200 dimensions (indicating the number of decision variables). The 200 dimensions are sufficient to handle almost all complex problems in real world. Tackling problems with a higher-dimensionality will require specific algorithm design and mechanics, such as problem decomposition and cooperative coevolution [27], and thus is beyond the scope of this work. Our experiments study the effects of an introduced probability parameter and the swarm size. The results are compared with those from pure G3PCX [25] and SPSO07 [14]. Although an improved performance can be anticipated from the proposed hybrid algorithm, we do not intend to compare with the state-off-the-art stochastic algorithms such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [28][29].

The rest of this article is organized as follows. Section 2 briefly reviews the concepts of PSO and some recombination operators of real-coded genetic algorithms. Section 3 presents the proposed PSPG algorithm. Section 4 explains the experimentation of performance evaluation using the benchmark functions. Section 5 reports and analyzes the experimental results of the proposed PSPG against a conventional PSO and G3PCX. Finally, section 6 concludes this work with suggested parameter tuning.

2. PSO AND REAL-CODED GENETIC ALGORITHMS

2.1 Particle Swarm Optimization Algorithm

PSO is a population-based optimization technique, where the population is called a *swarm* of particles. Each particle represents a possible solution to the optimization task at hand. This paper will base on the unconstrained minimization problems,

$$\min_{x\in\Omega} f(x), \tag{1}$$

where $\Omega \subset R^D \rightarrow R$ is a continuous real-valued function, and the search space Ω is a multidimensional interval specified by a lower and an upper bounds x_l, $x_u \in R^D$, respectively.

The original version of PSO, introduced by its originators [1], consists of a group or a swarm of N particles. The position of particle i at iteration t is represented by vector $\vec{x}_i(t) = (x_{i1}, x_{i2}, \ldots, x_{iD})$ and its velocity $\vec{v}_i(t) = (v_{i1}, v_{i2}, \ldots, v_{iD})$, where D is the dimension of optimization problem. During each iteration, each particle adjusts its position in a direction toward its personal best (pbest) position and the position of "some other" particles. There are many ways to identify such "some other" particles, resulting in different forms of PSO. The most commonly used forms are *gbest* and *lbest*. This paper bases on *gbest* PSO with an inertia weight w [6], in which the trajectory of particles is influenced by the best particle of entire swarm. For a particle i, its velocity is calculated as

$$\begin{aligned} v_{ij}(t+1) = w\cdot v_{ij}(t) + c_1\cdot r_{1j}(t)\cdot(pbest_{ij}(t) - x_{ij}(t)) \\ + c_2\cdot r_{2j}(t)\cdot(gbest_{ij}(t) - x_{ij}(t)) \end{aligned} \tag{2}$$

The particle's position is changed according to

$$\vec{x_i}(t+1) = \vec{x_i}(t) + \vec{v_i}(t+1) \tag{3}$$

where *pbest* and *gbest* are the personal best and global best positions, respectively, of that particle i. Subscript j indicates dimension and $j \in \{1, \ldots, D\}$. The c_1 and c_2 parameters, termed as cognitive and social components, are acceleration coefficients that pull the particle toward pbest and gbest respectively. r_1 and r_2 are vectors of random numbers with uniform distribution between 0 and 1. w is an inertia weight that balances the exploration and exploitation abilities of the swarm in search of an optimal solution. The adjustment of velocity and position for each particle is repeated until some stopping criteria are met.

A generic PSO algorithm can be implemented based on the following steps.

1. Initialize the swarm by assigning a random position in the multidimensional problem space to each particle.
2. Evaluate the fitness function for each particle.
3. For each individual particle, compare the particle's fitness value with its *pbest*'s value. If the current value is better than its *pbest*'s value, then set this particle as the *pbest*.
4. Identify the particle that has the best fitness value as *gbest*.
5. Update the velocities and positions of all particles using (2) and (3).

6. Repeat steps 2-5 until a stopping criterion is met (e.g. maximum number of iterations, or a satisfied fitness value).

Many attempts have been made to further improve the performance of original PSO, resulting in a number of well-known variants, e.g. time-varying inertia weight [6], constriction factor [7], self-organizing hierarchical PSO with time-varying acceleration coefficients [8], cellular-based PSO [9], co-evolutionary PSO [10] for constrained optimization problems, PSO for multi-objective problems [11], and adaptive multi-swarm approach for optimization problems with dynamic environment [12]. In 2007, M. Clerc *et al.* had published a version of Standard PSO 2007 on http://www.particleswarm.info, which was aimed to replace the original versions. This standard version accommodates several improvements in PSO community since its inception, and is intended for use as a baseline for performance testing of further improvement [14]. Some major improvements in this version are the randomized number of neighbours for communication, instead of using the whole swarm, and slight changes in values of both the weight and acceleration coefficients [14]. Thus we will use this Standard PSO 2007 [14] are a baseline for performance comparison.

2.2 G3PCX

In a Real-Coded Genetic Algorithm (RCGA), the solution is represented as a vector of real-valued decision variables. First, a population of solutions is randomly created within the search space. Then genetic operations (such as recombination and mutation) are performed to create a new population in an iterative manner. Most RCGA differs from each other mainly by the means of their recombination. The commonly used recombination operators in RCGAs are SBX [30], UNDX [31], and SPX [32]. In 2002, Deb *et al.* proposed the parent-centric recombination (PCX), which is an extension of two-parent SBX operator for any number of parents. The PCX operator assigns more probability for creating an offspring close to each parent, not the centroid of them [25], and can be briefly described as follows. For each offspring, the direction vector $\vec{d}^{(p)}$ is calculated from a randomly chosen parent to the centroid of them, $\vec{g}$, i.e., $\vec{d}^{(p)} = \vec{x}^{(p)} - \vec{g}$. The offspring is then created from the orthonormal bases that span the subspace perpendicular to each of the direction vectors above. Please refer to [25] for more details of the PCX algorithm.

PCX was introduced along with the Generalized Generation Gap (G3) evolutionary model, which is a steady-state and elite-preserving algorithm. The G3 method always selects the best individual to participate as a parent. First, two other parents are randomly selected. After offspring are created, the method replaces those randomly selected parents with the best two solutions from a combined population, which includes two parents and the newly generated offspring. The performance of G3 with PCX was demonstrated to be superior to the standard Differential Evolution algorithms and many real-coded GA proposed earlier [25].

3. THE PROPOSED HYBRID PSO WITH G3PCX

This section describes the proposed hybrid PSO algorithm with G3PCX module, namely PSPG, as shown in Fig. 1. The probability P_x is first introduced here and used to select the running of either PSO module or G3PCX module in each generation. If a uniform randomized value is greater than the predefined probability P_x, then the velocity and position updates of the PSO module is performed; otherwise, the G3PCX module takes action. The μ parents used by G3PCX are chosen from the *gbest* particle and $\mu-1$ other particles randomly selected. Next, the G3 module works as described in subsection 2.2. That is, μ parents will generate λ offspring using the PCX operator. Then, the best λ particles from the combined $(\mu + \lambda)$ particles will replace the previously chosen subset RP, which is composed of the *gbest* particle and other $\lambda - 1$ particles randomly selected from the swarm. At the end of evolution process, the G3PCX module is run again in order to perform further refinement. In this work, the selection probability P_x is set constant, and Clerc and Kennedy's PSO algorithm with constriction factor [7], which helps ensure the convergence, is used as the PSO module. With the constriction factor χ, the velocity update equation (2) is replaced with

$$v_{ij} = \chi\left[w \cdot v_{ij} + c_1 \cdot r_{1j} \cdot (pbest_{ij} - x_{ij}) + c_2 \cdot r_{2j} \cdot (gbest_{ij} - x_{ij})\right], \quad (4)$$

with the explicit time step t omitted for notational convenience. The constriction factor χ is defined as [7],

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}, \varphi = c_1 + c_2, \varphi \geq 4. \quad (5)$$

4. PERFORMANCE EVALUATION

4.1 Test Functions

The proposed hybrid PSO with G3PCX algorithm is evaluated using 8 nonlinear benchmark functions selected from [33] and [34].These functions, listed in Table 1, are scalable minimization problems with a wide range of characteristics such as unimodal, multimodal, separable and non-separable, as well as discontinuity and noises. While there exist many benchmark functions in literature, most of them share similar characteristics with those used here. Ref. [33] and [34] give detailed descriptions of those functions.

PSPG Algorithm

```
Begin
    P = current population
    μ = number of parents participating in PCX
    λ = number of offspring created
    while ( NFC < MaxNFC )
        if ( rand(0.0, 1.0) < Px )
            call g3pcx_module
        else
            create new P using PSO's equations
            evaluate P
        end if
        update pbest and gbest
    end while
    call g3pcx_module
    update pbest and gbest
End

Subroutine g3pcx_module
Begin
    select μ parents using G3 scheme, one of which is the gbest
    generate λ offspring using PCX operator
    build a subset RP of the gbest particle plus the λ-1 particles randomly from
        P-{gbest}
    replace RP with the best λ ones from (λ + μ) offspring union with parents
End
```

Fig.1: *Algorithm of the proposed PSPG.*

Table 1: *Benchmark function. D: dimensions, C: Characteristics, U: Unimodal, M: Multimodal, S: Separable, N: Non-separable.*

No.	Name		Range	C
f1	Schwefel's P2.22	$f(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}x_i$	[–10, 10]	US
f2	Step function	$f(x)=\sum_{i=1}^{D}(\lfloor x_i+0.5\rfloor)^2$	[–100, 100]	US
f3	Quartic with noise	$f(x)=rand[0,1)+\sum_{i=1}^{D}i\cdot x_i^4$	[–1.28, 1.28]	US
f4	Rosenbrock	$f(x)=\sum_{i=1}^{D-1}\left[100(x_{i+1}-x_i^2)^2+(1-x_i)^2\right]$	[–30, 30]	MN
f5	Rastrigin	$f(x)=\sum_{i=1}^{D}\left[x_i^2-10\cos(2\pi x_i)+10\right]$	[–5.12, 5.12]	MS
f6	Ackley	$f(x)=-20\cdot\exp\left(-0.2\cdot\sqrt{\frac{1}{D}\cdot\sum_{i=1}^{D}x_i^2}\right)-\exp\left(\frac{1}{D}\cdot\sum_{i=1}^{D}\cos(2\pi x_i)\right)+20+e$	[–320, 32]	MN
f7	Griewank	$f(x)=\frac{1}{4000}\sum_{i=1}^{D}x_i^2-\prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}})+1$	[–600, 600]	MN
f8	Penalized	$f(x)=\frac{\pi}{D}\left\{10\sin^2(\pi y_1)+\sum_{i=1}^{D-1}(x_i-1)^2\cdot\left[1+10\sin^2(\pi y_{i+1}+1)\right]+(y_D-1)^2\right\}$ $+\sum_{i=1}^{D}u(x_i,10,100,4),\ y_i=1+(x_i+1)/4,\ u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a,\\ 0, & -a\le x_i\le a,\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	[–50, 50]	MN

Table 2: *Profile of the experiments.*

Experiment	D	MaxNFC	# runs
1. Effects of probability P_x	30	100,000	100
	200	500,000	50
2. Effects of swarm size	30	100,000	100

4.2 Experimental Setups and Performance Criteria

Two experiments are performed to test the performance of proposed hybrid algorithm. Experiment 1 studies the effects of probability Px that controls the selection of running either PSO or G3PCX modules in each generation. The simulation is set with its dimension or number of decision variables D equal to 30 and 200. Experiment 2 studies the effects of swarm size. Table 2 summarizes the configurations of each experiment. The detailed configuration, results and discussions of these experiments are given in Section 5. For each algorithm or variant, 100 independent runs with different seeds for the random number generator are performed per function tested at $30D$, and 50 independent runs at $200D$ to avoid excessive computational time.

The population is asymmetrically initialized to a lower 40% portion of each dimension, so that the global optimum will not lie in the middle of the initial population[2]. Each run will terminate when the maximum number of objective function calls (MaxNFC) (as shown in Table 2) is reached. The average convergence graphs of some important functions are illustrated in Fig. 2 for discussion. The statistical results (means and standard deviations) of the achieved optimal values are reported in Table 3 and analyzed for a comparison with those from pure G3PCX [25] and SPSO07 [14].

The parameters of G3PCX, in both pure and the proposed PSPG, are set to the same values as suggested [25], i.e. $\mu = 3$ and $\lambda = 2$. In fact, other values of these parameters are also tested, but the best results are still from such recommendations. The proposed PSPG algorithms were written in Java language. The C codes for G3PCX and SPSO07 used in this experiment were downloaded from www.iitk.ac.in/kangal/codes.shtml and www.particleswarm.info/Programs.html, respectively. All program codes use the common routine of random number generator, and the simulations are performed on Windows 7 with Core2Duo 2.0 GHz CPU.

5. RESULTS AND DISCUSSIONS

5.1 Experiment 1: Effects of probability Px.

The first experiment aims at comparing the performance of proposed hybrid algorithm (PSPG) against SPSO07 [14] and pure G3PCX [25]. Adding more dimensions to a problem space will result in the exponential growth of associated volume [36]. This rapid growth in volume significantly increases the complexity of problem. The performance of an algorithm with excellent search ability at a moderate dimension (such as 20-30) may deteriorate quickly when the dimension increases beyond 100. Therefore, this experiment tests the scalable benchmark functions at both 30 and 200 dimensions. In the case of testing at 200 dimensions, the MaxNFCs are extended from 100,000 to 500,000 to provide more search opportunity.

The swarm size or population size N is kept unchanged. The suggested population size for using G3PCX, or other evolutionary algorithms, is generally larger than that of a PSO algorithm for optimizing a function having the same dimension [25]. Hence, the swarm sizes of PSPG and SPSO07 are set $N = 25$, whereas the population sizes of pure G3PCX are 150 and 200. In fact, we tested G3PCX at different population sizes of 50, 100, 150, 200 and 250. The optimal results are from those runs with 150 and 200; thus, they are listed in Table 3. In order to investigate the effects of probability Px in PSPG, 18 values of Px were tested: 0.005, 0.01, 0.05, 0.10, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, and 0.99. However, only the results from a range of 0.01 to 0.20 can outperform others, and thus they are included in Table 3.

Table 3 reports the means and standard deviations of the objective function values achieved from running each algorithm for all runs. To determine the significance of difference, we conduct the statistical t-test comparisons with a 95% confidence interval between PSPG against SPSO07 and G3PCX. For each benchmark tested, the better result from either G3PCX-N200 or G3PCX-N150 is used for comparison. From Table 3, the results of PSPG with $P_x = 0.05$ and 0.10 outperform those from other Px values; thus we only conduct statistical t-test for both of them as the representatives of PSPG. The t-test results are reported in Table 4 with the following notations: "+" tag indicates that the average result from PSPG is significantly better than the result from SPSO07 or G3PCX depending on the column; "-" tag indicates that the difference between two results is not significant; and "-" tag indicates the PSPG's result is significantly worse than the result from SPS07 or G3PCX. Fig. 2 illustrates the average convergence graphs for some important functions of 30D in Experiment 1.

From both Table 3 and 4 and the average convergence graphs in Fig. 2, we can observe the following.

1. The performance of PSPG with P_x= 0.05, 0.10 and 0.15 are in fact comparable. On the other hand, the performance of G3PCX with $N = 200$ is somewhat better than G3PCX with $N = 150$. We may observe from Table 3 that the performance of G3PCX degrades quickly when the number of decision variables increases from $D = 30$ to 200, especially in unimodal functions.
2. For 30-dimensional tests, PSPG with P_x= 0.10 performs the best in unimodal functions. PSPG with P_x= 0.10 and 0.15 outperform SPSO07 with a statistical significance for 4 functions, whereas

[2]This initialization scheme was suggested by Angeline [35] and widely used in PSO community since then.

Table 3: *Mean and standard deviation (in parenthesis) of the optimization results from Experiment 1.*

D	Function	SPSO07	G3PCX-200N	G3PCX-150N	PSPG-0.01	PSPG-0.05	PSPG-0.10	PSPG-0.15	PSPG-0.20
30	*f*1 Schwefel's P2.22	5.800	57.490	52.130	9.91E-13	3.69E-15	**1.27E-15**	2.69E-15	3.88E-15
		(8.352)	(12.490)	(12.700)	(2.17E-12)	(7.65E-15)	**(1.48E-15)**	(3.42E-15)	(6.08E-15)
	*f*2 Step function	200.000	29.550	39.650	6.790	4.800	**3.700**	4.050	6.150
		(1414.000)	(3.258)	(5.140)	(4.720)	(8.869)	**(4.184)**	(1.532)	(4.374)
	*f*3 Quartic with noise	**0.011**	0.468	0.591	0.013	0.025	0.036	0.054	0.067
		(0.008)	(0.051)	(0.056)	(0.007)	(0.021)	(0.021)	(0.037)	(0.052)
	*f*4 Rosenbrock	407.200	**0.997**	1.595	40.970	37.020	35.500	39.060	43.800
		(898.900)	**(0.386)**	(0.437)	(27.880)	(29.110)	(25.010)	(25.470)	(32.710)
	*f*5 Rastrigin	90.070	171.300	165.300	**65.030**	71.900	80.970	81.590	86.720
		(45.250)	(6.464)	(8.481)	**(17.220)**	(18.450)	(20.950)	(25.190)	(23.720)
	*f*6 Ackley	19.970	6.052	11.080	1.378	**1.169**	1.204	1.304	1.525
		(0.002)	(1.050)	(1.307)	(1.142)	**(1.045)**	(1.025)	(0.846)	(1.087)
	*f*7 Griewank	**5.17E-03**	9.95E-03	6.28E-03	5.82E-02	4.83E-02	6.24E-02	4.17E-02	6.61E-02
		(7.86E-03)	(3.57E-03)	(1.56E-03)	(6.93E-02)	(4.17E-02)	(5.79E-02)	(3.74E-02)	(5.45E-02)
	*f*8 Penalized	0.183	**0.151**	0.168	0.501	0.272	0.961	0.398	0.799
		(0.306)	**(0.012)**	(0.013)	(1.041)	(0.474)	(2.011)	(0.882)	(1.321)
200	*f*1 Schwefel's P2.22	12.710	4.90E+69	2.00E+73	0.977	0.116	**0.085**	0.094	0.122
		(8.938)	(4.78E+69)	(1.95E+73)	(1.461)	(0.046)	**(0.044)**	(0.027)	(0.025)
	*f*2 Step function	1.28E+03	1.39E+05	1.40E+05	290.60	138.80	120.50	**92.30**	103.40
		(6.74E+02)	(7.48E+02)	(7.23E+02)	(137.60)	(96.43)	(56.29)	**(47.98)**	(35.56)
	*f*3 Quartic with noise	3.130	7692.0	7845.0	0.289	0.238	**0.215**	0.290	0.288
		(3.581)	(76.8)	(107.3)	(0.046)	(0.114)	**(0.083)**	(0.173)	(0.129)
	*f*4 Rosenbrock	2.12E+04	2.43E+09	2.45E+09	1065.0	787.9	**712.9**	785.8	794.8
		(3.37E+04)	(2.19E+07)	(2.57E+07)	(430.6)	(205.9)	**(183.7)**	(129.4)	(118.3)
	*f*5 Rastrigin	**631.7**	3.30E+03	3.29E+03	719.0	739.3	814.6	765.6	838.0
		(68.1)	(15.550)	(17.290)	(82.7)	(63.5)	(64.1)	(65.5)	(102.8)
	*f*6 Ackley	8.076	20.990	21.000	5.111	4.803	**3.093**	3.659	3.498
		(1.033)	(0.009)	(0.004)	(2.142)	(2.862)	**(0.406)**	(1.954)	(2.391)
	*f*7 Griewank	4.612	5017.000	5046.000	0.058	**0.039**	0.066	0.091	0.049
		(6.184)	(32.980)	(42.970)	(0.042)	**(0.030)**	(0.045)	(0.080)	(0.023)
	*f*8 Penalized	20.500	1.378	1.215	2.748	2.043	1.050	1.458	**0.972**
		(28.210)	(0.048)	(0.050)	(1.671)	(1.893)	(0.748)	(1.376)	**(0.582)**

Column heading PSPG-n denotes the proposed algorithm PSPG with $P_x = n$. The best (lowest) mean and standard deviations achieved are highlighted with red bold font.

their differences are not significant for the remaining 4 functions. Being compared to G3PCX, PSPG statistically outperforms in 5 functions, but is defeated only in Rosenbrock function. This function is non-separable and multimodal when the number of variables is more than three [25], and is well known for its difficult fitness landscape. Also, the conventional PSO algorithms are known to have troubles optimizing it [16]. The rotationally invariant crossover in G3PCX, which is implanted into PSPG, can assist the hybrid algorithm in solving this function better than conventional PSO. But the occasional runs of rotational crossover in PSPG cannot take full advantage as compared with the pure G3PCX. In case of the Griewank and Penalized functions, no single algorithm/variant in this test can outperform the others with a statistical significance.

3. For 200-dimensional tests, Table 4 clearly shows the outstanding performance of PSPG with $P_x = 0.10$ and 0.15 over both SPSO07 and G3PCX with a statistical significance for all cases, except one. This exception is the highly-multimodal Rastrigin function, in which the difference between PSPG and SPSO07 is not statistically significant.

5.2 Experiment 2: Effects of swarm size

The swarm size or number of particles in PSO algorithm is known to cause some effects on performance. This experiment tests the performance of PSPG using P_x= 0.05, which is the optimal value observed from previous experiment. The same benchmark functions

Table 4: *Statistical t-test comparison results of PSPG-0.05 and PSPG-0.10 against SPSO07 and G3PCX.*

D	Function	against SPSO07		against G3PCX	
		PSPG-0.10	PSPG-0.15	PSPG-0.10	PSPG-0.15
30	*f*1 Schwefel's P2.22	+	+	+	+
	*f*2 Step function	+	+	+	+
	*f*3 Quartic with noise	O	O	+	+
30	*f*4 Rosenbrock	+	+	–	–
	*f*5 Rastrigin	O	O	+	+
	*f*6 Ackley	+	+	+	+
	*f*7 Griewank	O	O	O	O
	*f*8 Penalized	O	O	O	O
200	*f*1 Schwefel's P2.22	+	+	+	+
	*f*2 Step function	+	+	+	+
	*f*3 Quartic with noise	+	+	+	+
	*f*4 Rosenbrock	+	+	+	+
	*f*5 Rastrigin	O	O	+	+
	*f*6 Ackley	+	+	+	+
	*f*7 Griewank	+	+	+	+
	*f*8 Penalized	+	+	+	+

Symbols "+", "O" and "–" indicate that the PSPG with a corresponding P_x is significantly better than, not different to, and worse than the SPSO07 or G3PCX.

Table 5: *Mean and standard deviation (in parenthesis) of the optimization results from Experiment 2: Effects of swarm size.*

Function	PSPG-N20	PSPG-N30	PSPG-N40	PSPG-N50	PSPG-N60	PSPG-N70	Significant?
*f*1 Schwefel's P2.22	2.45E-10 (-4.40E-10)	9.00E-08 (-9.83E-08)	3.70E-06 (-2.22E-06)	6.95E-05 (-9.13E-05)	8.36E-26 (-1.99E-25)	3.28E-15 (-4.06E-15)	+
*f*2 Step function	2.5 (-3.248)	1.4 (-2.538)	0.6 (-1.114)	0.25 (-0.622)	9.95 (-8.176)	5.25 (-8.808)	O
*f*3 Quartic with noise	0.0143 (-0.008)	0.0124 (-0.0058)	0.0098 (-0.0033)	0.0103 (-0.0036)	0.113 (-0.0913)	0.0251 (-0.0119)	O
*f*4 Rosenbrock	29.093 (-16.203)	37.475 (-24.764)	55.29 (-28.872)	45.703 (-28.875)	43.622 (-29.288)	35.085 (-26.199)	O
*f*5 Rastrigin	67.4083 (-16.445)	56.7623 (-16.8658)	53.3795 (-13.9339)	50.6931 (-16.3047)	85.5165 (-27.2995)	64.523 (-16.5751)	O
*f*6 Ackley	0.293 (-0.5198)	0.1906 (-0.4581)	0.1866 (-0.4591)	0.0752 (-0.3273)	2.5624 (-1.2325)	1.4273 (-1.0644)	+
*f*7 Griewank	0.0498 (-0.0354)	0.043 (-0.0446)	0.0497 (-0.057)	0.0613 (-0.0611)	0.1197 (-0.1289)	0.0636 (-0.0473)	O
*f*8 Penalized	0.7138 (-1.1371)	1.0031 (-1.5638)	0.3466 (-0.6126)	0.2415 (-0.4713)	0.1738 (-0.056)	0.2411 -0.4199)	O

The results were averaged over 100 runs. Column heading PSPG-N*ss* denotes the proposed algorithm PSPG with swarm size of *ss*. the last column indicates statistical significance of comparing the differences of results between the best and worst variants.

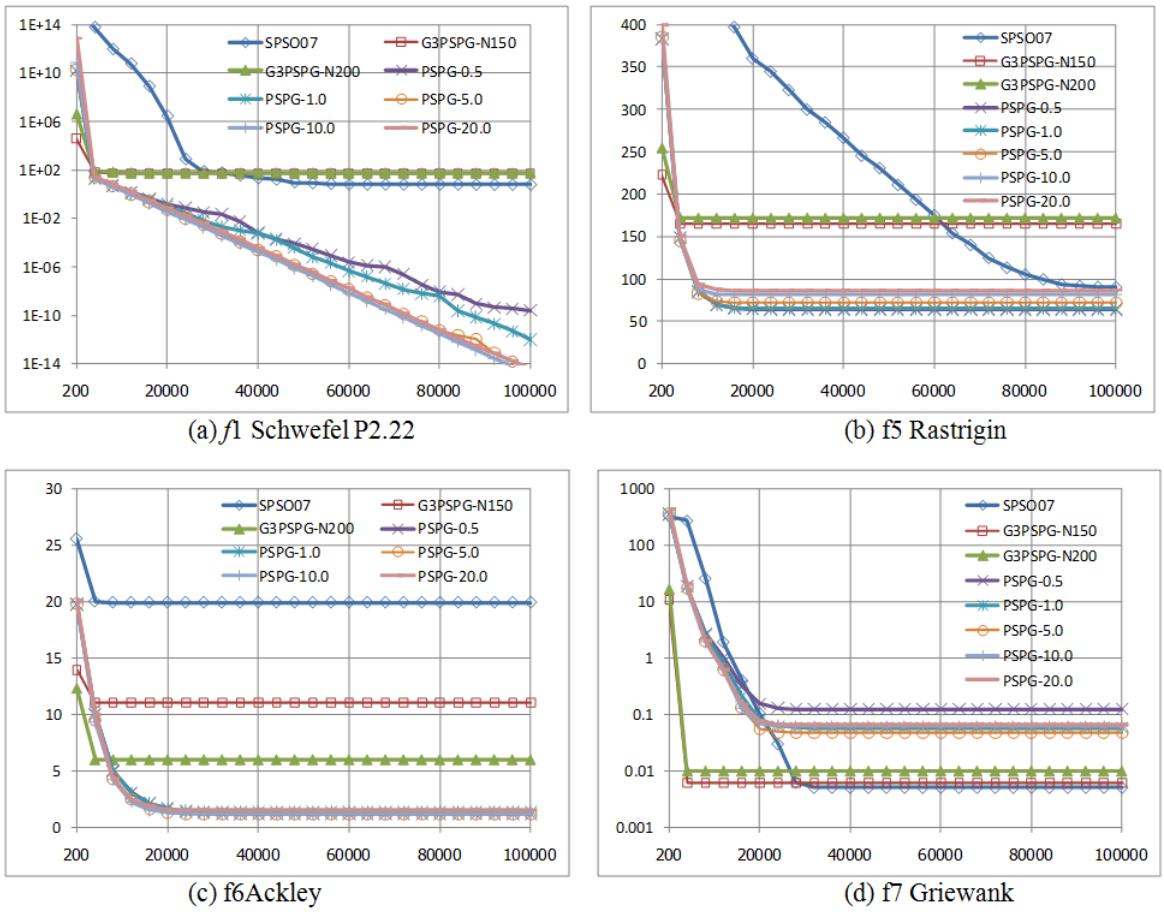

Fig.2: *Average convergence graphs for some functions of 30 dimensions in Experiment 1.*

are selected for this study. The swarm sizes are varied from 20 to 70.

Table 5 reports the means and standard deviations of optimization results averaged over 100 runs. In each function, it can be observed that the results are not much different as the swarm size is changed. Consequently, we perform t-test comparison, with a 95% confidence interval, to determine whether the differences of the best and worst results are statistically significant. The last column indicates t-test results: "+" tag means the difference between the best and worst variant is significant, and "O" tag means the difference is not statistically significant. According to the experimental results from running with varied swarm size, there are no statistically differences in 6 out of 8 tested functions. This demonstrates that the proposed algorithm PSPG is not much sensitive to its swarm size.

However, in order to find the optimal swarm size, all PSPG variants are ranked according to their means (in Table 5) for each function. Table 6 reports these ranks, together with their averages grouped by modality: unimodal and multimodal. The variant with a lower value of averaged rank means can better achieve the optima than the variant with a higher averaged rank. It can be observed that the swarm size of 40 offers the best results for unimodal functions (with the average rank of 2.7), whereas the swarm size of 50 is better for multimodal functions (average rank of 2.8). This behaves similarly to the case of generic PSO algorithms.

6. CONCLUSION

PSO algorithm is well-known for its good convergence to the search subspace that an optimum is situated, but it is sometimes prolonged to reach such optimum due to its oscillating behavior. In contrast, G3PCX is well-known for its superior convergence speed particularly in those functions with noncomplex fitness landscape. This paper proposes a hybrid algorithm, namely *PSPG*, which combines both the global search efficiency of PSO with the convergence capability of G3PCX. Two experimentations are conducted to investigate the convergence behavior of three algorithms: Standard PSO 2007 (SPSO07) [14], G3PCX [25], and the proposed PSPG, using eight well known nonlinear benchmark functions of 30 and 200 dimensions. The probability P_x of PSPG allows users to balance the executions of both PSO and G3PCX modules, depending upon the characteristics of problem at hand. According to the experimental

results, a small value of P_x (about 0.01-0.05) is recommended for simple or unimodal functions, while a larger value (0.10) is suitable for multimodal functions. For a problem with unknown characteristics, setting P_x= 0.05 is recommended. Similarly to the case of generic PSO algorithms, a moderate swarm size of about 40 is recommended for optimizing 30 dimensional functions. Slightly more particles are effective for functions with complex fitness landscape or high dimensions, while fewer particles are sufficient for fast convergence in simple functions.

References

[1] J. Kennedy, R. C. Eberhart, "Particle swarm optimization," *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ, 1995, pp. 1942-1948.

[2] J.B. Park, Y.W. Jeong, J.R. Shin, and K.Y. Lee, "An Improved Particle Swarm Optimization for nonconvex economic dispatch problems," *IEEE Transactions on Power Systems*, vol. 25, pp. 156–166, 2010.

[3] T. Zhang, and B.W. Brorsen, "Particle Swarm Optimization Algorithm for Agent-Based Artificial Markets," *Computational Economics*, vol. 34, pp. 399–417, 2009.

[4] S.C. Chiam, K.C. Tan, A.Al. Mamun, "A memetic model of evolutionary PSO for computational finance applications," *Expert Systems with Applications*, vol. 36, pp. 3695–3711, 2009.

[5] T. Navalertporna, N.V. Afzulpurkar, "Optimization of tile manufacturing process using particle swarm optimization," *Swarm and Evolutionary Computation*, vol. 1, pp. 97–109, 2011.

[6] Y. Shi, R. C. Eberhart, "A modified particle swarm optimizer," *Proceedings of the IEEE Congress on Evolutionary Computation 1998*, 1998, pp. 69–73.

[7] M. Clerk and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evolutionary Computation*, vol. 6, pp. 58–73, 2002.

[8] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Trans. Evolutionary Computation*, vol. 8, pp. 240–255, 2004.

[9] Y. Shi, H. Liu, L. Gao, G. Zhang, "Cellular particle swarm optimization", *Information Sciences*, vol. 181, pp. 4460-4493, 2011.

[10] R.A. Krohling, L. dos Santos Coelho, "Coevolutionary Particle Swarm Optimization Using Gaussian Distribution for Solving Constrained Optimization Problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 36, pp. 1407–1416, 2006.

[11] C.A.C. Coello, G.T. Pulido, M.S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Trans. Evolutionary Computation*, vol. 8, pp. 256–279, 2004.

[12] T. Blackwell, "Particle Swarm Optimization in Dynamic Environments," *Studies in Computational Intelligence*, vol. 51, 2007, pp. 29–49, 2007.

[13] D. Bratton, J. Kennedy, "Defining a Standard for Particle Swarm Optimization," *Proc. of the 2007 IEEE Swarm Intelligence Symposium*, 2007, pp. 120–127.

[14] M. Clerc *et al.* Standard PSO 2007. http://www.particleswarm.info (standard pso 2007.c), 2007.

[15] F. van den Bergh, A.P. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, pp. 937–971, 2006.

[16] N. Hansen, R. Ros, N. Mauny, M. Schoenauer, A. Auger, "PSO facing non-separable and ill-conditioned problems," *Research Report RR-6447*, INRIA 2008.

[17] T. Korenaga, T. Hatanaka, and K. Uosaki, "Performance improvement of particle swarm optimization for high-dimensional function optimization," *Proc. of IEEE Congress on Evolutionary Computation*, 2007. Sept. 2007, pp. 3288–3293.

[18] T. Takahama, S. Sakai, and N. Iwane, "Constrained optimization by the constrained hybrid algorithm of particle swarm optimization and genetic algorithm," in *Proc. of the 18th Australian Joint Conference on Artificial Intelligence 2005, LNCS*, 3809, Springer, Berlin, 2005, pp. 389–400.

[19] Y.-T. Kao, E. Zahara, "A hybrid genetic algorithm and particle swarm optimization for multimodal functions," *Applied Soft Computing*, vol. 8, pp. 849–857, 2008.

[20] X. Cai, D.C. Wunsch, "Engine Data Classification with Simultaneous Recurrent Network using a Hybrid PSO-EA Algorithm, in *Procs. of 2005 IEEE International Joint Conference on Neural Networks*, vol. 4, 2005, pp. 2319–2323.

[21] A.A. Esmin, G. Lambert-Torres, G. B. Alvarenga, "Hybrid evolutionary algorithm based on PSO and GA mutation," *Proceedings of the Sixth International Conference on Hybrid Intelligent Systems*, 2006, pp. 57.

[22] P.S. Shelokar, Patrick Siarry, V.K. Jayaraman, B.D. Kulkarni, "Particle swarm and ant colony algorithms hybridized for improved continuous optimization," *Applied Mathematics and Computation*, vol. 188, pp. 129–142, 2007.

[23] M.-R. Chen, X. Li, X. Zhang, Y.Z. Lu, "A novel particle swarm optimizer hybridized with extremal optimization," *Applied Soft Computing*, vol. 10, pp. 367–373, 2010.

[24] C. Zhang, J. Ning, S. Lu, D. Ouyang, T. Ding, "A novel hybrid differential evolution and particle swarm optimization algorithm for unconstrained optimization," *Operations Research Letters*, vol. 37, pp. 117–22, 2009.

[25] K. Deb, A. Anand, and D. Joshi, "A computationally efficient evolutionary algorithm for real-parameter optimization," *IEEE Trans. Evolutionary Computation*, vol. 6, pp. 371–395, 2002.

[26] A. Iorio and X.Li, "Rotationally invariant crossover operators in evolutionary multi-objective optimization," in T.-D. Wang *et al.* (Eds.): *SEAL 2006, LNCS 4247*, 2006, pp. 310–317, Springer-Verlag Berlin Heidelberg, 2006.

[27] Z. Yang, K. Tang, X. Yao, "Large scale evolutionary optimization using cooperative coevolution", *Information Sciences*, vol. 178, pp. 2985–2999, 2008.

[28] N. Hansen, A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, pp. 159–195, 2001.

[29] A. Auger, N. Hansen, "A restart CMA evolution strategy with increasing population size," *Proc. of the IEEE Congress on Evolutionary Computation*, pp. 1777–1784, 2005.

[30] K. Deb, R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, pp. 115–148, 1995.

[31] I. Ono, H. Kita, and S. Kobayashi, "A Real-Coded Genetic Algorithm Using the Unimodal Normal Distribution Crossover," *Advances in Evolutionary Computing*, Springer-Verlag, New York, 2003, pp. 213–237.

[32] T. Higuchi, S. Tsutsui, M. Yamamura, "Theoretical analysis of simplex crossover for real-coded genetic algorithms," *Parallel Problem Solving from Nature (PPSN-VI)*, 2000, pp. 365–374.

[33] X. Yao, Y. Liu, G. Lin, "Evolutionary programming Made Faster," *IEEE Trans. Evolutionary Computation*, vol. 3, pp. 82–102, 1999.

[34] J. Brest, S. Greiner, B. Borko, M. Marjan, *Ž*. Vilgem, "Self-Adapting Control Parameters in Differential Evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evol. Comp.*, vol. 10, no. 6, pp. 646–657, 2006.

[35] P. J. Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences," in V. W. Porto and *et al.*, editors, *Evolutionary Programming*, vol. 1447 of Lecture Notes in Computer Science, pp. 601–610. Springer, 1998.

[36] R. Bellman, *Adaptive control processes: a guided tour*. Princeton University Press, 1961.

Chukiat Worasucheep received B.Eng. (Computer Engineering) from Chulalongkorn University, Thailand, and M.Sc. (Computer Science) from School of Electrical Engineering and Computer Science, Oregon State University, USA. He is currently an assistant professor of Applied computer science program, Faculty of Science, KMUTT, Thailand. His research interests are computational intelligence for business and scientific problems.

Charinrat Pipopwatthana was a student in Applied computer science program, KMUTT. Her senior project is under the supervision of Chukiat Worasucheep in 2008. Since graduation she has been working at Gosoft (Thailand) Co.,Ltd. where she is currently a software testing leader.

Sujitra Srimontha was a student in Applied computer science program, KMUTT. Her senior project is under the supervision of Chukiat Worasucheep in 2008. Now she has been working as a programmer in IT-Consumer Banking Solution department of Bank of Ayudhya PCL.

Wilasini Phanmak was a student in Applied computer science program, KMUTT. Her senior project is under the supervision of Chukiat Worasucheep in 2008. Now she has been working as a programmer in Business Operation Solution Delivery section of Bank of Ayudhya PCL.