International Journal of Innovative Computing, Information and Control
Volume **8**, Number **1(B)**, January **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 715–726

MULTI-OBJECTIVE BEE SWARM OPTIMIZATION

Reza Akbari and Koorush Ziarati

Department of Computer Science and Engineering
Shiraz University
Mollasadra St., Shiraz, Iran
rakbari@cse.shirazu.ac.ir; ziarati@shirazu.ac.ir

Received June 2010; revised October 2010

Abstract. *This work presents a novel multi-objective bee swarm optimization (MOBSO) method. The proposed method divides a swarm as experienced foragers, onlookers and scouts. An adaptive windowing mechanism is used by the experienced foragers in order to select their own leaders and adjust their next positions. Also, the adaptive windowing is used for truncating the most crowded members of the archive. A new way is proposed in which the scouts and adaptive windowing are used to maintain diversity over the Pareto front. A scout creates a hypercube using knowledge provided by a pair of archive members, and flies spontaneously in it. The provided knowledge by the experienced foragers is used by the onlookers in order to adjust their flying trajectories. The proposed algorithm was compared with existing multi-objective optimization methods. The experimental results indicate that the proposed approach not only presents a uniformly distributed Pareto front but also identifies results with greater accuracy.*
Keywords: Bee swarm optimization, Multi-objective optimization

1. **Introduction.** Optimization problems with more than one objective function are common in many engineering problems. Our goal is to find a set of solutions that represent a trade-off among the objectives; because in such areas there is no single solution available for these problems.

The multi-objective optimization approaches can be classified as aggregative, lexicographic, sub-population, pareto-based and hybrid methods. The aggregative methods work by combining all the objectives into a single one. The behavior of an aggregative method depends on the aggregative function which can be linear, dynamic or bang bang [1,2]. In lexicographic methods, the objectives are ranked based on their importance. These objective functions are minimized separately in order to obtain the optimum solution [3,4]. Lexicographic ordering has deficiency in optimizing problems with a large number of objectives. A sub-population method subdivides a population in sub-populations. Each sub-population is responsible for optimizing one of the objective functions. Trade-off among different solutions is obtained by exchanging information between sub-populations [5,6]. Pareto-based methods select non-dominated individuals based on Pareto dominance notion as leaders. Usually leaders are maintained in an external archive [7-12]. Finally, a hybrid method is designed as a combination of two or more techniques in order to exploit their advantages [13,14].

Among the multi-objective methods, the majority of researches are concentrated on Pareto-based approaches. Due to computational complexity, the evolutionary search methods have received significant attention to optimize multi-objective problems [8-11].

In this paper, we present a novel method based on intelligent behaviors of honey bees to handle problems with multiple objectives. In MOBSO, a swarm is divided into three types of bees, and a fixed-sized archive is used to maintain good solutions. This archive

which acts as a leader is used by experienced forager bees to adjust their trajectories. An experienced forager employs a decision making process to select an archive member as its own leader. A new windowing mechanism has been provided, which is used by experienced foragers to select their own leaders. Also a pair of archive members are used by a scout to create a hypercube and fly randomly in it. Onlookers use the provided knowledge by the experienced foragers. A new adaptive windowing mechanism and the scouts are used as a new way to provide appropriate diversity over the Pareto front.

The rest of the paper is organized as follows. The basic concepts are presented in Section 2. Section 3 describes the details of the proposed algorithm. Next, the experimental results are discussed in Section 4. Finally, Section 5 concludes the paper.

2. Basic Concepts.

Let $x = (x_1, x_2, \ldots, x_D)$ be a D-dimensional vector of decision variables and S be a search space. Multi-objective optimization (MOO) concerns optimization problems with multiple and often conflicting objectives. A MOO problem can be stated as finding a solution vector $\vec{x} \in S$ which minimizes the function z with conflicting objectives:

$$z = f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \tag{1}$$

subject to k inequality constraint

$$g(x) = (g_1(x), g_2(x), \ldots, g_k(x)) \geq 0 \tag{2}$$

and m equality constraint

$$h(x) = (h_1(x), h_2(x), \ldots, h_m(x)) = 0. \tag{3}$$

Unlike single objective optimization, MOO solutions are in such a way that the performance of each objective cannot be improved without sacrificing performance of at least another one. Hence, the solution to a MOO problem exists in the form of an alternate tradeoff known as a Pareto optimal set. The Pareto Optimal set is defined based on Pareto Dominance. A decision vector x_a dominates another vector x_b (denote as $x_a \prec x_b$) if $f(x_{a,i}) \leq f(x_{b,i})\ \forall i = \{1, 2, \ldots, D\}$ and $\exists j \in \{1, 2, \ldots, D\}$ where $f(x_{a,i}) < f(x_{b,i})$.

Therefore, a set of M decision vectors $\{w_i\}$ is a non-dominate set if:

$$w_i \prec w_j \quad i, j = 1, 2, \ldots, M. \tag{4}$$

By considering Pareto Dominance, decision vector x_a is Pareto optimal if for every decision vector x_b and $I = \{1, 2, \ldots, D\}$ either $\forall_{i \in I} (f_i(x_b) = f_i(x_a))$ or there is at least one $i \in I$ such that $f_i(x_b) > f_i(x_a)$. The Pareto Optimal Set ($P*$) for a given multi-objective problem $f(\vec{x})$ which contains all Pareto Optimal solutions is defined as:

$$P* = \{x \in S | \neg \exists x' \in S \text{ and } f(x') \prec f(x)\}. \tag{5}$$

Therefore, a Pareto front ($PF*$) for a given multi-objective problem $f(\vec{x})$ and Pareto optimal set $P*$ is defined as:

$$PF* = \{f(x) | x \in P*\}. \tag{6}$$

A Pareto-based MOO method tries to find the optimal Pareto front. A Pareto front can be convex, concave or partially convex and/or concave and/or discontinuous [1]. The determination of true Pareto front is a difficult task due to large number of suboptimal Pareto fronts, the existing memory constraints, the nature of Pareto front and computational complexity.

3. **Multi-objective Bee Swarm Optimization.** The MOBSO algorithm is designed based on the BSO method proposed by Akbari et al. in [15]. The MOBSO divides a swarm to experienced forager, onlooker, and scout bees which fly in a D-dimensional search space $S \subset R^D$ to find the optimal Pareto front. Assume that we have a set of bees, β, in a swarm, these bees are partitioned as $\beta = \vartheta \cup \kappa \cup \xi$, where ξ, κ and ϑ respectively represent the sets of experienced forager, onlooker and scout bees. We use a small part of bees as scouts. The other part of bees is divided dynamically as onlooker and experienced forager bees throughout iterations.

In MOBSO, each bee i is associated with a position vector $\vec{x}(\beta, i)$ which represents a feasible solution in a D-dimensional search space S. Each position vector $\vec{x}(\beta, i)$ represents a food source with an associated quality vector $f(\vec{x}(\beta, i))$.

Figure 1 presents the pseudocode of MOBSO algorithm. MOBSO has three main phases: initialization, update and termination. The first phase initiates the algorithm. The second phase is the body of algorithm and iteratively updates the swarm and the external archive. Finally, the third phase terminates the algorithm and returns the known Pareto front. The details of different phases are described in the following subsections.

3.1. **Initialization.** MOBSO receives number of bees, $n(\beta)$, percentage of scouts, ps, number of dimensions, D, maximum number of iterations, MAX-ITER, and maximum archive size, MAX_ARCH_SIZE as inputs. In the initialization phase, the number of scouts and the number of onlookers and experienced forager, $n(\xi \cup \kappa)$, are respectively determined as $n(\vartheta) \leftarrow ps \times n(\beta)$ and $n(\xi \cup \kappa) \leftarrow n(\beta) - n(\vartheta)$.

Except the scouts, the types of other bees are dynamically determined at each cycle of the algorithm as described in Subsection 3.2. After that, all the bees are positioned randomly in the search space S. Then MOBSO initializes the archive of size $n(ARCH)$. Each of the bees in the swarm SW_0 (the initial swarm) is evaluated using function $add\_non\_dominated(SW_0)$, and the positions of the non-dominated bees are stored in the archive $ARCH_0$ (the initial archive).

3.2. **Update.** After initiation, the bees of the swarm employ the following processes to adjust their positions throughout iterations until the termination condition is met. At each iteration of the algorithm, a predefined percentage of the swarm SW acts as scouts, while the remaining bees are dynamically partitioned as experienced foragers and onlookers. The non-domination process is carried out using function $select\_non\_dominated(SW_{iter-1})$ to select experienced foragers. The whole swarm except scouts is evaluated and the non-dominated ones are selected as experienced foragers while the others are selected as onlookers. The classification of bees into three categories provides a swarm with highly dynamic behaviour which can use different flying behaviours. The experienced forager, onlooker, and scout bees use these flying behaviours to probabilistically adjust their trajectories in the search space for finding new food sources with better nectars. The flying patterns of the bees are explained below.

3.2.1. *Experienced foragers.* An experienced forager bee uses the good solutions maintained by external archive ARCH to adjust its movement trajectory at the next time. At each cycle of the algorithm, the nectar information about food sources and their positions (social knowledge) which are maintained by external archive are shared between experienced foragers. After that, an experienced forager evaluates the provided nectar information by archive members, employs a probabilistic approach to choose one of that food sources as its own leader using function $select\_leader\_bee(ARCH_{iter-1})$, and adjust its flying trajectory towards the selected food source. In other words, an experienced

Algorithm MOBSO($n(\beta)$, ps ,D, *MAX-ITER*, *MAX_ARCH_SIZE*) /* $n(\beta)$: *number of bees*, ps : *percentage of scouts*, D: *dimensions of the search space S*, *MAX-ITER*: *max number of iterations*, *MAX_ARCH_SIZE: max size of archive*/*

(1) Initialization

$iter = 0$, **Initialize** swarm SW_0 randomly /* SW_0 *is the swarm at iteration 0*/*

Calculate $n(\mathcal{G}) \leftarrow ps \times n(\beta)$, $n(\xi \cup \kappa) \leftarrow n(\beta) - n(\mathcal{G})$

*/*determine the number of scout, onlookers and experienced foragers*/*

Initialize $x_0(\beta, ij)$, $\forall i \in \{1,2,...,n(\beta)\}$, $\forall j \in \{1,2,...,D\}$ /* $x(\beta, ij)$ *is the j-th coordinate of the i-th bee at the iter. 0*/*

$ARCH_0 \leftarrow$ add_non_dominated(SW_0) */*store the non-dominated solutions in the* $ARCH_0$**/*

*/**$ARCH_0$ *is the archive at iteration 0*/*

(2) Update

For *iter* =1 **to** *MAX-ITER*

$\xi_{iter-1} \leftarrow$ select_non_dominanted(SW_{iter-1})

*/*select non-dominated bees at the previous iteration as experienced foragers*/*

$\kappa_{iter-1} \leftarrow \beta_{iter-1} - \{\xi_{iter-1} \cup \mathcal{G}\}$ */*dominated solutions at the previous iteration constitute the onlookers*/*

(2.1) Experienced Foragers

For $i = 1$ **to** $n(\xi_{iter-1})$ */*do for each experienced forager bee*/*

$\ell(ARCH_{iter-1}, i) \leftarrow$ select_leader_bee$(ARCH_{iter-1})$ */*select a leader bee from archive for experienced forager i*/*

For $d = 1$ **to** D

$x_{iter}(\xi_{iter-1}, id) = x_{iter-1}(\xi_{iter-1}, id) + \omega_e r_e(\ell(ARCH_{iter-1}, id) - x_{iter-1}(\xi_{iter-1}, id))$

*/*update d-th coordinate of the experienced forager bee i*/*

Next d

Next i

(2.2) Onlookers

For $i = 1$ **to** $n(\kappa_{iter-1})$*/*do for each onlooker bee*/*

$e(\xi_{iter-1}, i) \leftarrow$ select_elite_bee(ξ_{iter-1}) */*select an elite bee from experienced forager bees for onlooker i*/*

For $d = 1$ **to** D

$x_{iter}(\kappa_{iter-1}, id) = x_{iter-1}(\kappa_{iter-1}, id) + \omega_e r_e(e(\xi_{iter-1}, id) - x_{iter-1}(\kappa_{iter-1}, id))$

*/*update d-th coordinate of an onlooker bee i*/*

Next d

Next i

(2.3) Scouts

For $i = 1$ **to** $n(\mathcal{G})$ */*do for each scout bee*/*

$b_1 \leftarrow$ select_boundary$(ARCH_{iter-1})$ */*randomly select a non-dominated solution as boundary 1*/*

$b_2 \leftarrow$ select_boundary$(ARCH_{iter-1})$ */*randomly select a non-dominated solution as boundary 2*/*

For $d = 1$ **to** D

$Ld, Ud \leftarrow$ select_lower_upper_bound(b_1, b_2) */*select lower and upper bounds for dimension d*/*

Next d

$x_{iter}(\mathcal{G}, i) = Sf(L, U)$ */* fly spontaneously in the hypercube*/*

Next i

(2.4) Archiving

$ARCH_{iter} \leftarrow$ add_non_dominated$(ARCH_{iter-1}, SW_{iter})$ */*update the archive by adding non-dominated solutions*/*

If($n(ARCH_{iter}) > MAX\_ARCH\_SIZE$ **) Then** truncate_archive$(ARCH_{iter})$

Next *iter*

(3) Termination

Return $ARCH_{iter}$ */*return the known Pareto front and exit*/*

FIGURE 1. Pseudocode of MOBSO algorithm

forager bee i selects an archive member j from the archive ARCH_{iter-1} as its own interesting leader bee, denoted as $\vec{\ell}(ARCH_{iter-1}, i)$, with probability p_j. The probability p_j is defined as a relative fitness of the selected archive member j:

$$p_j = \frac{fit(\vec{x}(ARCH_{iter-1}, j))}{\sum_{c=1}^{n(ARCH_{iter-1})} fit(\vec{x}(ARCH_{iter-1}, c))} \tag{7}$$

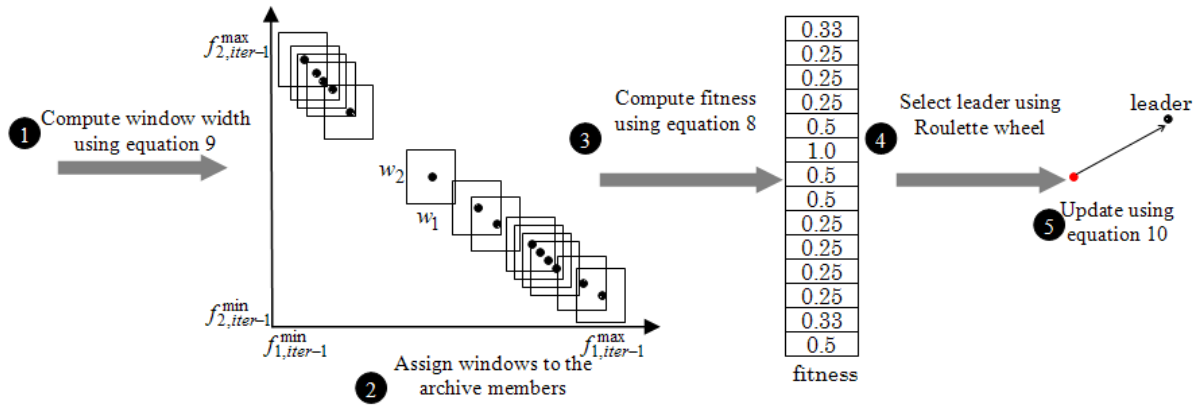


FIGURE 2. Adaptive windowing mechanism is performed in five steps

where $ARCH_{iter-1}$ is the archive at the previous iteration, $n\left(ARCH_{iter-1}\right)$ is the number of archive members at the previous iteration, and $fit\left(\vec{x}\left(ARCH_{iter-1}, j\right)\right)$ is the fitness value of the food source proposed by the archive member j which is proportional to the quality of food source. The quality of food source depends on the number of food sources in its neighborhood:

$$fit\left(\vec{x}\left(ARCH_{iter-1}, j\right)\right) = \frac{1}{Neighbors_{iter-1}\left(j\right) + 1} \tag{8}$$

where $Neighbors_{iter-1}\left(j\right)$ is the number of neighbors of the archive member j at the previous iteration. To count the number of food sources in the neighborhood of archive member j, an adaptive windowing mechanism is used. As shown in Figure 2, for each archive member, a window is determined and centered at the position of that member. These windows are determined based on the approximated knowledge and the size of the archive. For a MOO problem with n objectives, the window width in the i-th objective dimension can be calculated as:

$$w_i = \frac{\left|f^{\max}_{i,iter-1}\left(x\right) - f^{\min}_{i,iter-1}\left(x\right)\right|}{n\left(ARCH_{iter-1}\right)}, \quad i \in \{1, 2, \ldots, n\} \tag{9}$$

where $f^{\max}_{i,iter-1}\left(x\right)$ and $f^{\min}_{i,iter-1}$ respectively presents the maximum and minimum values for the i-th objective function obtained at the previous iteration. The $Neighbors_{iter-1}\left(j\right)$ for the archive member j is considered as the number of archive members located at its local window. The roulette wheel approach is used by an experienced forager for selecting an interesting archive member as its own leader. In this approach, as the fitness of an archive member increases, the probability of its selection increases, too. After selecting the leader, the flying trajectory of an experienced forager i is controlled using the following equation:

$$\vec{x}_{iter}\left(\xi_{iter-1}, i\right) = \vec{x}_{iter-1}\left(\xi_{iter-1}, i\right) + \omega_\ell r_\ell \left(\vec{\ell}\left(ARCH_{iter-1}, i\right) - \vec{x}_{iter-1}\left(\xi_{iter-1}, i\right)\right) \tag{10}$$

where $\vec{\ell}\left(ARCH_{iter-1}, i\right)$ is the position vector of the interesting leader bee selected by experienced forager $i \in \xi$ using Equation (7), $\vec{x}_{iter-1}\left(\xi_{iter-1}, i\right)$ and $\vec{x}_{iter}\left(\xi_{iter-1}, i\right)$ respectively represents the position of the old food source and the new one which are selected by the experienced forager i. Finally, r_ℓ is a random variables of uniform distribution in range of [0, 1], and parameter ω_ℓ controls the importance of the knowledge provided by the leader bee. The product of theses two parameter probabilistically controls the attraction of the experienced forager bee towards its interesting food source area.

3.2.2. *Onlookers.* At each cycle of MOBSO, the bees with the poor quality (i.e., dominated ones) are selected as onlookers. The onlooker bees use the social knowledge provided by experienced forager bees. An onlooker bee i randomly selects an experienced forager bee using function $select\_elite\_bee\,(\xi_{iter-1})$ as its own elite bee. The position of the selected elite bee is represented as $\vec{e}\,(\xi_{iter-1}, i)$. The position of an onlooker bee $i \in \kappa$ is updated as:

$$\vec{x}_{iter}\,(\kappa_{iter-1}, i) = \vec{x}_{iter-1}\,(\kappa_{iter-1}, i) + \omega_e r_e\,(\bar{e}\,(\xi_{iter-1}, i) - \vec{x}_{iter-1}\,(\kappa_{iter-1}, i)) \tag{11}$$

where $\vec{x}_{iter-1}\,(\kappa_{iter-1}, i)$ and $\vec{x}_{iter}\,(\kappa_{iter-1}, i)$ respectively represents the position of the old food source and the new one which are found by the onlooker bee $i \in \kappa$, r_e is a random variables of uniform distribution in range of [0, 1] which model the stochasticity of flying pattern, and parameter ω_e respectively control the importance of the knowledge provided by the elite bee.

3.2.3. *Scouts.* A scout bee flies randomly in the search space in order to find more profitable regions. The spontaneous fly is performed in a hypercube which represents a region in the search space. MOPSO determines the hypercube by selecting two archive members b_1 and b_2 using function $select\_boundary\,(ARCH_{iter-1})$. This function selects an archive member at random. After that the lower and upper bounds (i.e., Ld and Ud) of the hypercube along each dimension d are determined by comparing the archive members b_1 and b_2 using function $select\_lower\_upper\_bound\,(b_1, b_2)$. By iterating this process on all dimensions, a hypercube will be created for the scout bee $i \in \vartheta$.

Maintaining diversity over the Pareto front has an important role in approximating a well distributed Pareto fronts. We use scout bees as well as the adaptive windowing mechanism as a new way for maintaining diversity over the Pareto front. This approach provides the MOBSO with high exploration capability while the exploitation is controlled at an appropriate level. Under this approach, a well distributed Pareto front will be obtained for most of the test functions.

3.2.4. *Archiving.* MOBSO algorithm uses a fixed size external archive in order to maintain the good solutions found by the swarm. The size of archive, MAX_ARCH_SIZE, can be adjusted according to the desired number of solutions distributed over the Pareto front. The archive is updated at each cycle as follows:

(1) Each candidate solution in the current swarm SW_{iter} is evaluated against the archive $ARCH_{iter-1}$ using function $add\_non\_dominated\,(ARCH_{iter-1}, SW_{iter})$. If the candidate solution is not dominated by any members of the archive, it will be added to the archive.
(2) The function $add\_non\_dominated\,(ARCH_{iter-1}, SW_{iter})$ also evaluates each member of the archive. Any archive member dominated by the added solution will be removed from the archive.
(3) If the predetermined archive size, MAX_ARCH_SIZE, is reached, the function $truncate\_archive(ARCH_{iter})$ will be used in order to perform truncation process. The truncation process eliminates the most crowded members of the archive in order to maintain good distribution of solution over the Pareto front. For this purpose, the adaptive windowing mechanism as described in Subsection 3.2.1 is used and the fitness of each archive member is calculated based on the archive boundary and the archive size. The roulette wheel approach is conducted in order to eliminate the most crowded members. In this way, the probability of eliminating a member increases as its neighbors increases.

3.3. **Termination.** The algorithm terminates after predefined number of iterations. After termination, the external archive $ARCH_{MAX-ITER}$ contains the known Pareto front.

4. Experimental Results.

4.1. **Performance metrics.** We use Generational Distance and Spacing metrics in order to provide a quantitative assessment of the MOBSO.

Generational Distance: This metric estimates the gap between discovered non-dominated vectors and the optimal Pareto front [10]. This metric is defined as:

$$GD = \frac{1}{n}\sqrt{\sum_{i=1}^{n} d_i^2} \tag{12}$$

where n is the number of non-dominated vectors in the Preto-front, and d_i is the Euclidean distance between i-th non-dominated vectors in the Pareto-front and its nearest member in the Pareto-optimal set.

Spacing: This metric measures the distribution of non-dominated vectors throughout discovered Pareto-front [10]. This metric is defined as:

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(d_i - \bar{d}\right)^2} \tag{13}$$

where n is the number of non-dominated vectors, d_i is the Euclidean distance between i-th non-dominated vectors in the Pareto-front and its nearest member in the Pareto-optimal set, and $\bar{d}$ is the mean of all d_i.

4.2. **Benchmarks.** Several test functions are used in order to investigate performance of the proposed method. We choose eight test problems: Fonseca and Fleming's study, Kursawe's study, Deb's study, Ziztler's study, Kita's study and Binh's study [16]. All the problems have two objective functions. Binh's and Kita's problems are side constrained while the other problems have no constraints. All the problems are minimization except Kita's study.

4.3. **Settings of the algorithms.** The performance of our algorithm is compared against two most representative multi-objective algorithms so called MOPSO [10] and NSGAII [11]. The settings of the algorithms are given as follows: all the algorithms used a population of 100 individuals and the archive size was set at 100. For all test functions, we report the results obtained from performing 30 independent runs of each algorithm compared. The total number of evaluations for Fonseca, Kursawe and Kita is set at 1000, and Deb1, Deb2, ZDT1, ZDT6, Binh2 are respectively evaluated after 3000, 500, 10000, 5000 and 3000 evaluations.

MOPSO uses 30 divisions for the adaptive grid. Also, mutation rate in MOPSO was set at 0.5. For NSGAII, a binary crossover operator with the crossover rate of 0.8, and a polynomial mutation with the mutation rate of $1/\ell$, where ℓ is the number of real variables was used in order to generate offsprings. In MOBSO, the scouts constitute 10 percent of the population and the other part of population is subdivided dynamically as experienced foragers and onlookers at each cycle of the algorithm. The weighting coefficients ω_ℓ and ω_e were set at 2.5 and 2.15 respectively.

4.4. **Performance analysis.** Simulations were conducted on the eight test problems in order to analyze the performances of the algorithms. Table 1 presents the results obtained by the algorithms for Generational Distance and Spacing metrics. The Generational Distance metric shows the average error from the optimal Pareto front. Hence, an algorithm with smaller Generational Distance provides better performance over a test function. Also, in multi-objective optimization problem, we prefer Pareto fronts with uniform distributions. Hence, we have used Spacing metric to show this ability of the algorithms. We say an algorithm has better distribution over the Pareto front if it obtains smaller Spacing metric.

TABLE 1. Experimental results for the generational distance and spacing metrics

Problem	SP	Generational Distance metric			Spacing metric		
		MOBSO	MOPSO	NSGAII	MOBSO	MOPSO	NSGAII
Fonseca	Best	0.001025303	0.001063	0.00122684	0.007781933	0.007893	0.00864307
	Worst	0.001220845	0.001289	0.00122684	0.009170206	0.009851	0.00864307
	Average	**0.001178532**	0.001202	0.00122684	**0.008500765**	0.008992	0.00864307
	Std. Dev.	2.14721E-05	4.82E-05	0.00000000	0.000349263	0.000510	0.00000000
Kursawe	Best	0.002105065	0.002584324	0.00143665	0.019427471	0.024640036	0.01163399
	Worst	0.003022404	0.003485472	0.00397450	0.026572046	0.03949868	0.04865345
	Average	**0.002509865**	0.003053565	0.00342352	**0.001830602**	0.033111776	0.04040135
	Std. Dev.	0.002509865	0.00031664	0.00104039	0.023675545	0.005068935	0.01557745
Deb1	Best	0.000272849	0.000551503	0.00102548	0.00242209	0.004901858	0.00782336
	Worst	0.000483363	0.000640838	0.00102548	0.004475277	0.005771291	0.00782336
	Average	**0.000339286**	0.000588315	0.00102548	**0.003299764**	0.005303136	0.00782336
	Std. Dev.	6.18426E-05	2.55983E-05	0.00000000	0.000640598	0.00025604	0.00000000
Deb2	Best	3.79852E-05	3.86121E-05	0.00016746	0.000267569	0.000293625	0.00159745
	Worst	5.58309E-05	0.000167025	0.00016746	0.000570521	0.001618058	0.00159745
	Average	**4.40065E-05**	0.000110415	0.00016746	**0.000345527**	0.001063151	0.00159745
	Std. Dev.	4.32413E-06	3.47944E-05	0.00000000	5.58173E-05	0.000365194	0.00000000
ZDT1	Best	0.000445354	0.042984011	0.00029493	0.001692201	0.090522091	0.00287370
	Worst	0.000704042	0.077879917	0.00029493	0.003319370	0.126463173	0.00287370
	Average	0.000578208	0.05597199	**0.00029493**	**0.002302196**	0.102113335	0.00287370
	Std. Dev.	6.06105E-05	0.006906754	0.00000000	0.000366320	0.008997605	0.00000000
ZDT6	Best	0.000190253	0.027296395	0.00712553	0.000844696	3.35404E-16	0.00642766
	Worst	0.00033428	0.125910138	0.00712553	0.002042493	0.456130031	0.00642766
	Average	**0.000236312**	0.041443009	0.00712553	**0.001089539**	0.191229333	0.00642766
	Std. Dev.	3.54026E-05	0.019230312	0.00000000	0.000276969	0.120909724	0.00000000
Kita	Best	0.000594202	0.000584562	0.00427810	0.001176267	0.00395703	0.04329636
	Worst	0.001858466	0.041693183	0.31075102	0.001324301	0.416320714	1.49035931
	Average	**0.000979901**	0.007128203	0.021028169	**0.001228209**	0.07033469	0.20581371
	Std. Dev.	0.000448417	0.008426248	0.016208540	1.58370E-05	0.08429853	0.13169046
Binh2	Best	0.027599568	0.031167808	0.04361098	0.199936183	0.186194005	0.31837609
	Worst	0.032997496	0.040076849	0.04361098	0.232800131	0.26390553	0.31837609
	Average	**0.030688537**	0.03537692	0.04361098	**0.219349531**	0.224332123	0.31837609
	Std. Dev.	0.001658485	0.002351925	0.00000000	0.009673602	0.024218719	0.00000000

By considering the average values, we can see that the proposed MOBSO algorithm has better convergence in most of the test functions. In addition, the best values for the most test function were obtained by MOBSO algorithm. Also, MOBSO results better solutions at the worst case compared to MOPSO and NSGAII algorithms. The results show that the MOBSO is the best strategy in terms of Generational Distance metric. However, NSGAII has relatively better performance on the ZDT1 test function. MOPSO produces poor results on this test function whereas MOBSO has competitive performance.

Except to ZDT1 test function, NSGAII is the least effective strategy on the Generational Distance metric. For the most test functions, NSGAII has the zero standard deviation.

Spacing metric used here to compare distribution of the results obtained by the algorithms. Table 1 contains the results for this metric. From the results, MOBSO outperforms MOPSO and NSGAII in all test functions. The performances of NSGAII and MOPSO are comparative with MOBSO in some cases such as Fonseca and Binh2. In general, they are not dominating MOBSO.

In order to provide better comparison, the Pareto fronts obtained by the algorithms have been plotted in Figures 3 and 4 with respect to the optimal Pareto optimal. These figures clearly show the convergence of the algorithms and distribution of the obtained solutions over the Pareto front. The Pareto fronts obtained by the algorithms for the Fonseca and Fleming's, Kursawe's, Deb1 and Deb2 functions are shown in Figure 3. The continuous line shows the Pareto optimal of the problem, and the circle signs show the obtained results by an algorithm. For the Fonseca's test function with non-convex Pareto front, MOBSO outperformed MOPSO an NSGAII with respect to the Generational Distance and Spacing metrics. It can be seen from Figure 3(a) that the Pareto fronts produced by MOPSO and NSGAII are not evenly distributed as compared to MOBSO.

Kursawe's function has a non-convex and discontinuous Pareto front. It is evident from the Figure 3(b) that NSGAII unable to find the complete Pareto front. Although, both MOPSO and NSGAII provided competitive results, they have some solutions that are not converged to the optimal Pareto front. It can be seen that MOBSO have the best performance among the algorithms. Also, MOBSO has the ability to evolve better distribution over the Pareto front.

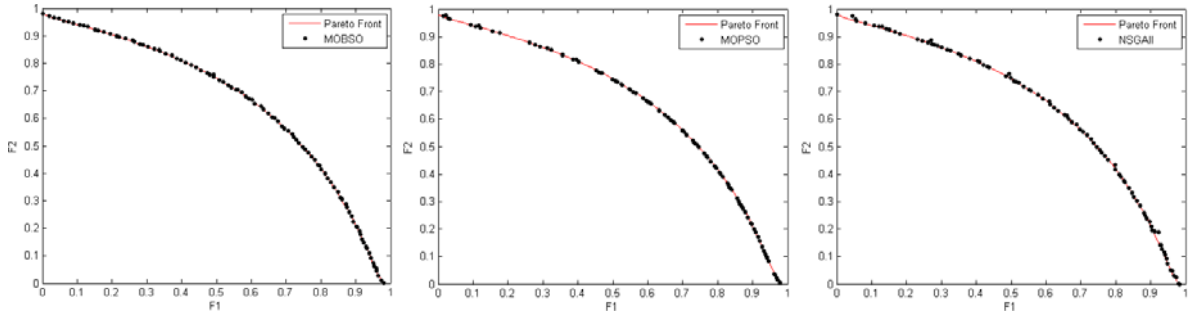
For both of the MOPSO and NSGAII, the solutions are not evenly distributed along optimal Pareto front of Deb1 test function. It can be seen from Figure 3(c) that MOBSO is able to evolve a diverse and well distributed nearly optimal Pareto front for this function. Although competitive results obtained by the algorithms over the Deb2 test function, the Pareto front obtained by MOBSO is found to be more uniformly distributed.

The Pareto fronts obtained by the algorithms for the ZDT1, ZDT6, Kita and Binh2 functions are shown in Figure 4. ZDT1 test function has convex Pareto front. Although, the Pareto front obtained by MOPSO has good distribution over ZDT1, the convergence to the optimal Pareto front was found comparatively inferior to MOBSO and NSGAII. Although NSGAII has better convergence to the optimal Pareto front, the Pareto front obtained by the MOBSO is found to be more uniformly distributed.

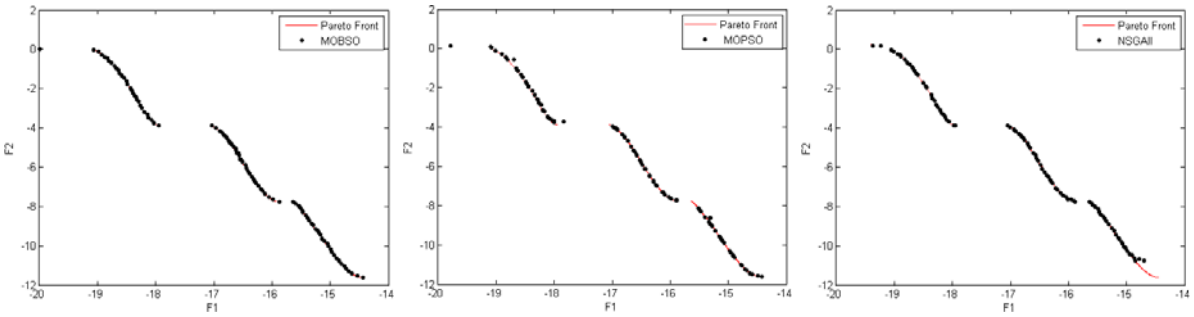
ZDT6 function has non-convex Pareto front. Similar to ZDT1 function, it can be noted that MOPSO has the worst results in terms of both Generational Distance and Spacing metrics. As evident from the figure, MOBSO successfully converged to the optimal Pareto front, while the other algorithms failed to converge to the optimal Pareto front. Pareto front obtained by MOBSO is found to be more uniformly distributed. However, the Pareto front obtained by NSGAII has better distribution with respect to the MOPSO.

The performance of NSGAII was found comparatively inferior to MOBSO and MOPSO over Kita's test problem. Although Pareto front obtained by MOPSO has good distribution, a part of its solution never converged to the Pareto optimal in all the test runs. Unlike, in all the test runs, the MOBSO was found to converge to the optimal Pareto front.

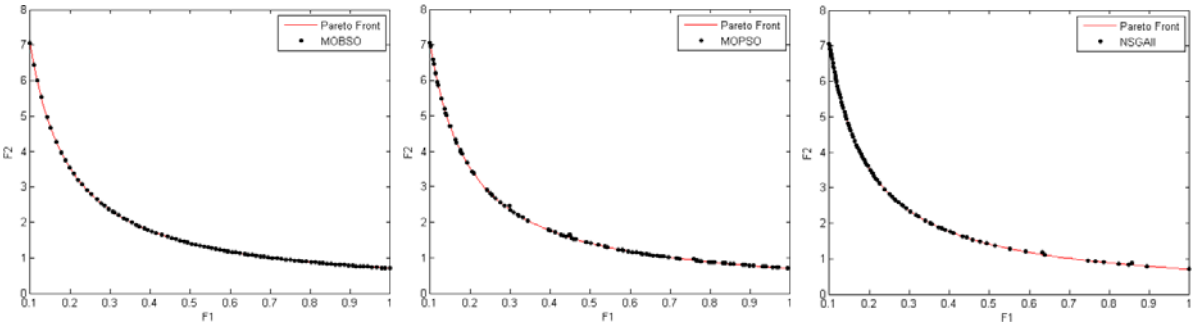
Although three algorithms have competitive solutions over Binh2 problem, the convex Pareto front obtained by MOBSO is found to be more uniformly distributed. The MOBSO have the best performance with respect to the Generational Distance and Spacing metrics.



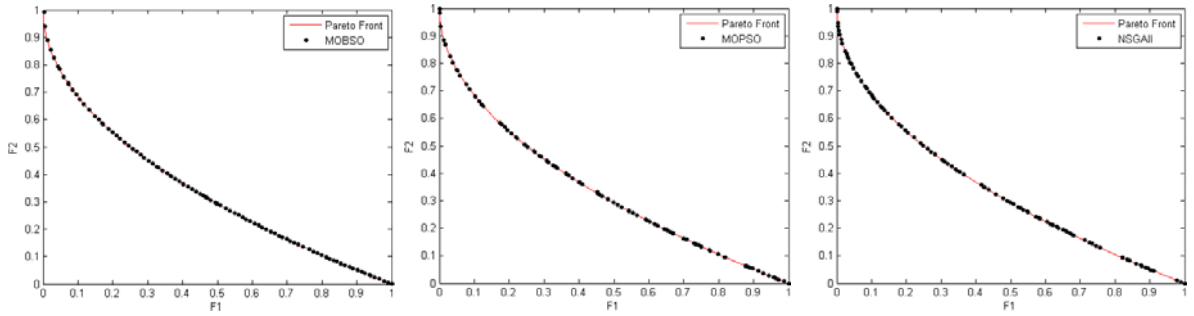
(a) Fonseca



(b) Kursawe

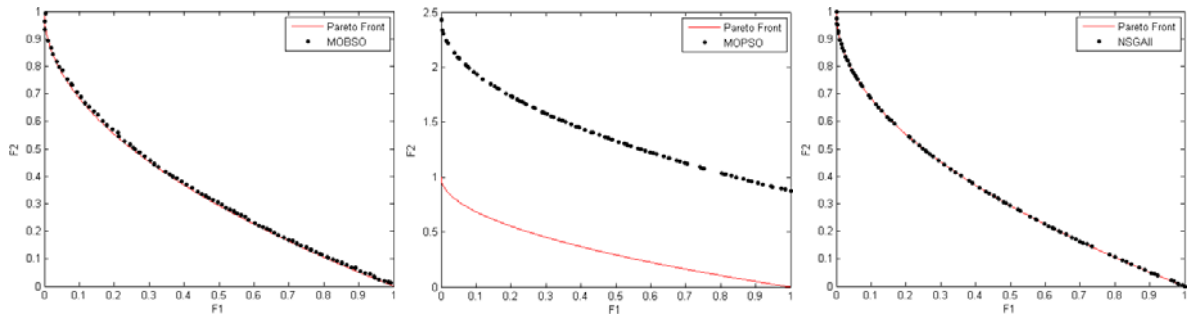


(c) Deb1

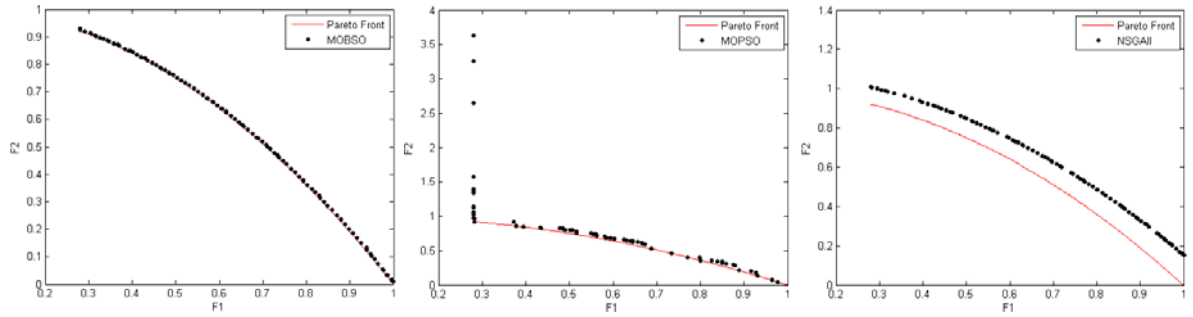


(d) Deb2

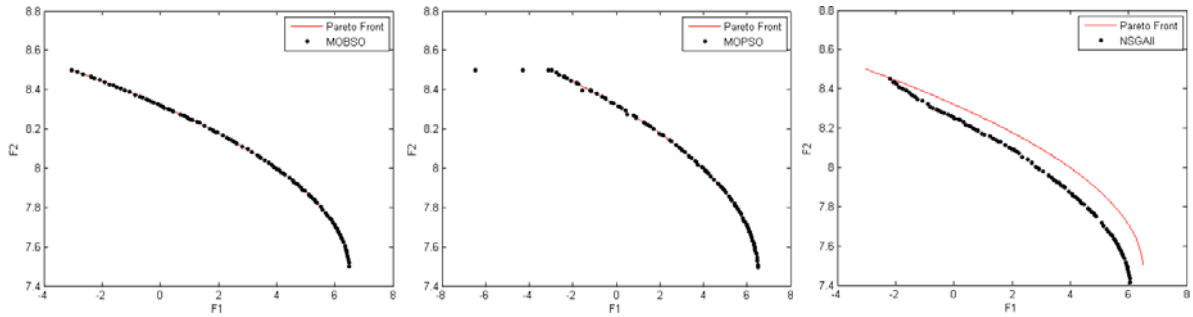
FIGURE 3. Pareto fronts produced for Fonseca, Kursawe, Deb1 and Deb2 test functions



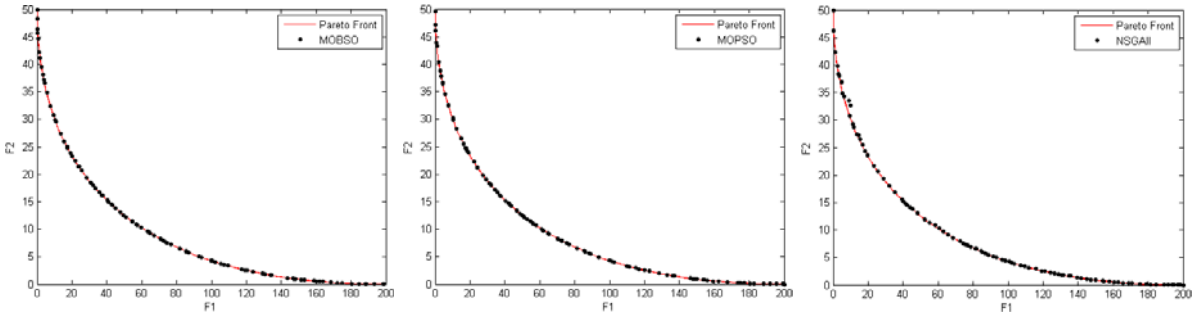
(a) ZDT1



(b) ZDT6



(c) Kita



(d) Binh2

FIGURE 4. Pareto fronts produced for ZDT1, ZDT6, Kita and Binh2 test functions

5. **Conclusion.** In this paper, we presented MOBSO method for optimizing multi-objective problems. The effectiveness of the MOBSO depends on three factors: 1) a swarm of different types of bees, 2) an adaptive windowing mechanism which dynamically adjusts the windows throughout iteration by considering the number of archive members and

the maximum and minimum values of the objective dimensions, and 3) an efficient way for maintaining diversity over the Pareto front. The MOBSO with such properties can be used as an efficient way for optimizing multi-objective problems. The comparative study showed that the proposed strategy produced results that are highly competitive with respect to the Generational Distance and Spacing metrics.

REFERENCES

[1] K. E. Parsopoulos and M. N. Vrahatis, Particle swarm optimization method in multiobjective problems, *Proc. of the 2002 ACM Symposium on Applied Computing*, pp.603-607, 2002.

[2] U. Baumgartner, C. Magele and W. Renhart, Pareto optimality and particle swarm optimization, *IEEE Transactions on Magnetics*, vol.40, no.2, pp.1172-1175, 2004.

[3] C. A. Coello Coello, D. A. Van Veldhuizen and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academic Publishers, New York, 2002.

[4] X. Hu, R. C. Eberhart and Y. Shi, Particle swarm with extended memory for multiobjective optimization, *Proc. of the 2003 IEEE Swarm Intelligence Symposium*, pp.193-197, 2003.

[5] E. Ozcan and C. K. Mohan, Particle swarm optimization: Surfing the waves, *Congress on Evolutionary Computation*, Washington D.C., USA, pp.1939-1944, 1999.

[6] M. Clerc and J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, *IEEE Transactions on Evolutionary Computation*, vol.6, no.1, pp.58-73, 2002.

[7] D. Liu, C. K. Tan, C. K. Goh and W. K. Ho, A multiobjective memetic algorithm based on particle swarm optimization, *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol.37, no.1, pp.42-50, 2007.

[8] C. Liu and Y. Wang, A new dynamic multi-objective optimization evolutionary algorithm, *International Journal of Innovative Computing, Information and Control*, vol.4, no.8, pp.2087-2096, 2008.

[9] K. Li, J. Zheng, M. Li, C. Zhou and H. Lv, A novel slicing based algorithm to calculate hypervolume for multi-objective optimization problems, *ICIC Express Letters*, vol.4, no.4, pp.1113-1120, 2010.

[10] C. A. Coello Coello, G. T. Pulido and M. S. Lechuga, Handling multiple objectives with particle swarm optimization, *IEEE Transactions on Evolutionary Computation*, vol.8, no.3, pp.256-279, 2004.

[11] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation*, vol.6, no.2, pp.182-197, 2002.

[12] C. Liu and Y. Wang, A new evolutionary algorithm for multi-objective optimization problems, *ICIC Express Letters*, vol.1, no.1, pp.93-98, 2007.

[13] C.-C. Lai, C.-H. Wu and M.-C. Tsai, Feature selection using particle swarm optimization with application in spam filtering, *International Journal of Innovative Computing, Information and Control*, vol.5, no.2, pp.423-432, 2009.

[14] X. Zhang, H. Meng and L. Jiao, Intelligent particle swarm optimization in multiobjective optimization, *Congress on Evolutionary Computation*, pp.714-719, 2005.

[15] R. Akbari, M. Mohammadi and K. Ziarati, A novel bee swarm optimization algorithm for numerical function optimization, *Journal of Communications in Nonlinear Science and Numerical Simulation*, vol.15, no.10, pp.3142-3155, 2010.

[16] C. A. Coello Coello, G. B. Lamont and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd Edition, Springer, 2009.