Multi-objective Evolutionary Programming without Non-domination Sorting is up to Twenty Times Faster

B. Y. Qu, P. N. Suganthan, *Snr Member, IEEE*

***Abstract*—In this paper, Multi-objective evolutionary programming (MOEP) using fuzzy rank-sum with diversified selection is introduced. The performances of this algorithm as well as MOEP with non-domination sorting on the set of benchmark functions provided for CEC2009 Special Session and competition on Multi-objective Optimization are reported. With this rank-sum sorting and diversified selection, the speed of the algorithm has increased significantly, in particular by about twenty times on five objective problems when compared with the implementation using the non-domination sorting. Beside this, the proposed approach has performed either comparable or better than the MOEP with non-domination sorting.**

I. Introduction

Numerous optimization problems have more than one objective in conflict with each other. Since there is no single solution for these problems, the aim is to find the Pareto optimal trade-off solutions that represent the best possible compromises among the objectives. Various multi-objective evolutionary algorithms (MOEAs) have been developed[1] [2] [3] to solve these problems. Multi-objective evolutionary programming (MOEP) is one of the MOEAs to solve the multi-objective optimization problems.

II. Multi-objective evolutionary programming

Evolutionary programming (EP) was first designed as an evolutionary approach to artificial intelligence [4, 5]. It is also very effective in solving numerical and combinatorial optimization problems [6]. Optimization by classical EP can be summarized into two major steps [7] (1) mutate all the solutions in the current population, and (2) select the next generation from the mutated and the current solutions.

MOEP is an extended version of single objective EP. The MOEP algorithm with non-domination sorting can be described in the following steps [8]:

Step 1: Randomly generate N_P number of initial trial solutions as parent solution and initiate the iteration, i=1.

Step 2: Create N_P number of offspring solutions from the parent solutions.

Step 3: Create $2N_P$ number of solutions in a present population, by combining the parent and offspring solutions.

Step 4: Identify the non-dominated solutions and assign the front number to each of the solutions.

Step 5: Sort the total $2N_P$ solution in the ascending order with respect to the front number.

Step 6: Select the first N_P solutions as parent solution to the next higher iteration.

Step 7: Check, if the stopping criterion (Maximum No. of Iterations) is met, then the present parent solutions are Pareto optimal front solutions. Otherwise, the procedure is repeated from Step 2.

Almost all recent Multi-objective evolutionary algorithms (MOEAs) [9] [10] [11] [12] [13] [14] that does not use weighted combination of objectives, use the concept of domination. A solution X_1 is said to dominate the other solution X_2 if the both conditions specified below are satisfied:

1. The solution X_1 is no worse than X_2 in all objectives.
2. The solution X_1 is strictly better than X_2 in at least on objective.

If both solutions do not satisfy the above conditions, solution X_1 and X_2 are non-dominating each other. However the steps of finding the non-dominated solutions are complicated and time consuming. Beside this, if the selection of parents is only based on the non-dominated front number, the diversity of the final solution is likely to be reduced for some problems. Francesco [15] proposed a preference order ranking scheme to improve the quality of the solution, however, this scheme is applied after the non domination sorting and it will further slow down the program. Motivated by these observations, the authors proposed the fast MOEP using fuzzy rank-sum concept and diversified selection.

III. MOEP with rank-sum sorting and diversified selection

A. Rank-sum Sorting

The proposed algorithm is different from classical MOEP in the sorting and selection steps. The rank-sum is used to divide every objective into 100 ranks and sum the corresponding rank for all objectives. The population is assigned the rank-sum according to its position in the search space. The process can be described in following steps assuming all are minimizing problems (maximizing problems can be converted to minimizing problems by multiplying -1):

Step 1 Select one unranked objective

Step 2 Get the maximum and minimum value of the selected objective to calculate the range for this objective

Step 3 Divide the searching range of this objective into 100 grids (100 fuzzy ranks)

Manuscript received November 30, 2008. This work was supported by the A *Star (Agency for Science, Technology and Research) under the grant # 052 101 0020.

Authors are with Nanyang Technological University, School of Electrical and Electronic Engineering, Singapore, 639798 (phone: 65-67905404; fax: 65-67933318; emails: E070088@ntu.edu.sg , epnsugan@ntu.edu.sg

Step 4 For every point in the search space, identify which grid it belongs to and assign the corresponding rank to the point for the selected objective.

Step 5 If all objectives have been selected go to step 6, otherwise repeat Step 1-4.

Step 6 Sum the ranks of every objectives of each solution to obtain the rank-sum of the solution. Also obtain the rank-sum for all solutions in population.

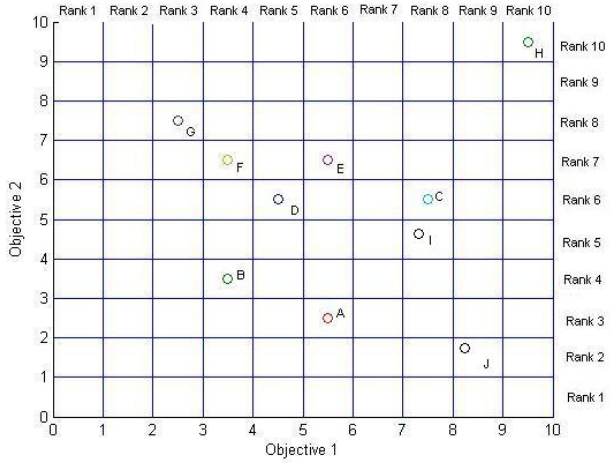

Fig. 1. Illustration of rank-sum

Table I Rank-sum of all 10 points

Point	Objective 1 rank	Objective 2 rank	Rank-sum
A	6	3	9
B	4	4	8
C	8	6	14
D	5	6	11
E	6	7	13
F	4	7	11
G	3	8	11
H	10	10	20
I	8	5	13
J	9	2	11

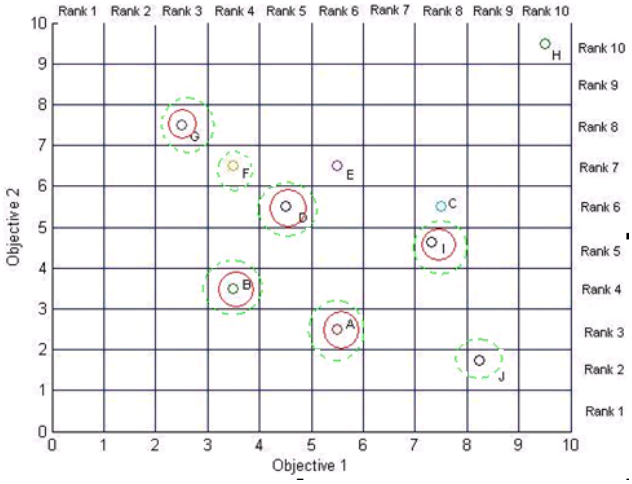

Fig. 2. Illustration of choosing preferential set

The Fig. 1 illustrates how the rank-sum is calculated for a two objective problem. The each objective range is divided into 10ranks. The ranks of all 10 points (A-J) for each objective as well as the rank-sum are shown in Table I.

B. Complexity Calculation

For rank-sum sorting, Step 2 requires O(N) comparisons to find the maximum and minimum value. Step 4 requires O(N) comparisons to identify the corresponding rank. Step 5 recursively call step 1-4, thereby requiring O(MN) computations of the above procedure. For non-domination sorting, the complexity to obtain the overall non-dominated set has the complexity of $O(MN^2)$ [16].

It is clear that the complexity of rank-sum sorting is in linear form while non-domination sorting is non-linear. Therefore, the rank-sort sorting requires shorter CPU time compare to non-domination sorting.

C. Diversified Selection

One of the major problems faced by all the evolutionary algorithms is the loss of diversity along front 1. In order to keep the diversity of the population during the selection process, the population will be divided into 2 sets, preferential set and backup set. As the names indicate, the solutions inside the preferential set will be selected first for generating offspring and backup set will only be selected to evolve if the solutions in the preferential set are insufficient. The steps of getting the preferential set are as follow:

Step 1 Select one unselected objective

Step 2 For the selected objective, scan *P* percentage of the total rank (i.e. from rank 1 to rank *P*, *P* can be set to 80 or 90). For each rank (if this rank is not empty, otherwise just continue to the next rank) of the selected objective, the solutions with the lowest rank-sum will be chosen to enter preferential set.

Step 3 If all objectives have been selected go to step 4. Otherwise repeat Step 1-2.

Step 4 Collect the solutions not inside the preferential set and put them in the backup set.

Fig. 2 illustrates how the preferential set is obtained. According to objective 1, 80 percent of rank is from rank 1 to rank 8. Since ranks 1, 2 and 7 are empty, they are not considered. For rank 3, the point with lowest rank-sum is G and it will be put into preferential set. For ranks 4, 5, 6, 8, solutions B, D, A and I will be selected and put in the preferential set respectively. The solutions chosen to be in the preferential set according to objective 1 are G, B, D, A, and I which are shown by solid circles in Fig. 2. Similarly, G, F, D, I, B, A and J are chosen according to objective 2. These solutions are shown by dotted circles in Fig. 2. Hence, the final preferential has solutions G, B, D, A, I, F and J, while the backup set has E and H. During the selection process, the points inside the preferential set will be selected first. Backup set will only be selected when there is no more points in preferential set for selection to maintain the specified fixed population size in the archive. Otherwise, the points in backup set will be removed. Fig. 3 shows the objective range covered by the preferential set with *P=80* (left) and *P=90*

(right).

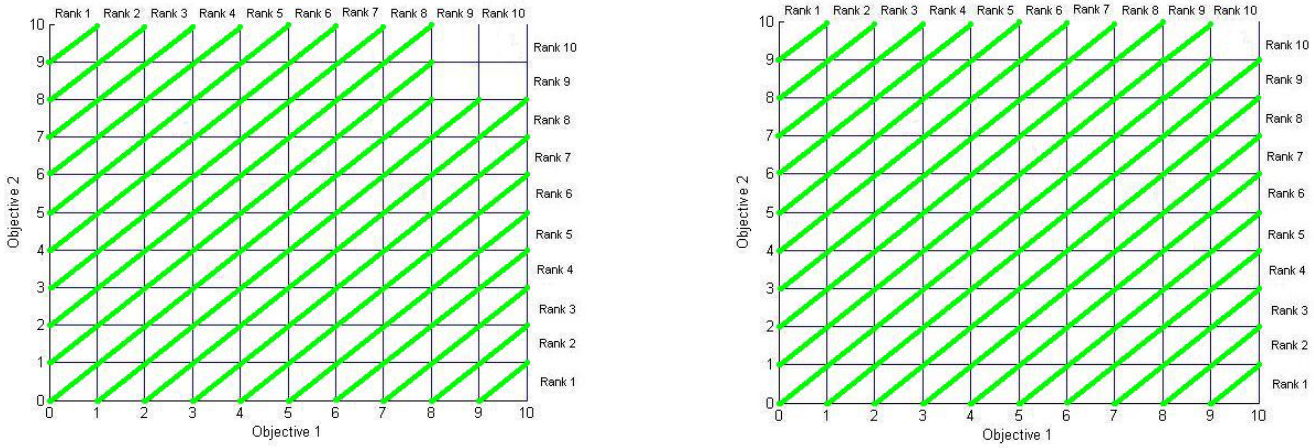

Fig. 3 Preferential set covered area with *P* set to 80 and 90

IV. EXPERIMENTS AND RESULTS

A. *Experiments Setup*

The test problems used in this paper are taken from CEC2009 special session and competition [17]. The algorithm was implemented in matlab 7.1 and the experiment parameters are listed below:

PC configuration:

System: Microsoft Windows XP
RAM: 1.00 GB
CPU: Pentium 4, 2.99 GHz
Computer Language: English

Parameter setting:

The maximal number of function evaluations:300000
Archive size of the MOEP with rank-sum sorting is:
200 for two objective problems
300 for three objective problems
1600 for five objective problems
Population size:
100 for two objective problems
150 for three objective problems
800 for five objective problems

The maximal number of solutions for computing the performance metric and archive size of the MOEP with non-domination sorting is:
100 for two objective problems
150 for three objective problems
800 for five objective problems

B. *Results*

Each of the test problems has been run for 30 times. Fig. 4 – 8 show the approximate solution sets obtained with the rank-sum method for the first 10 test instances. The performance metric (IGD [17]), CPU time of the MOEP with rank-sum sorting and the MOEP with non-domination sorting are presented in table II:

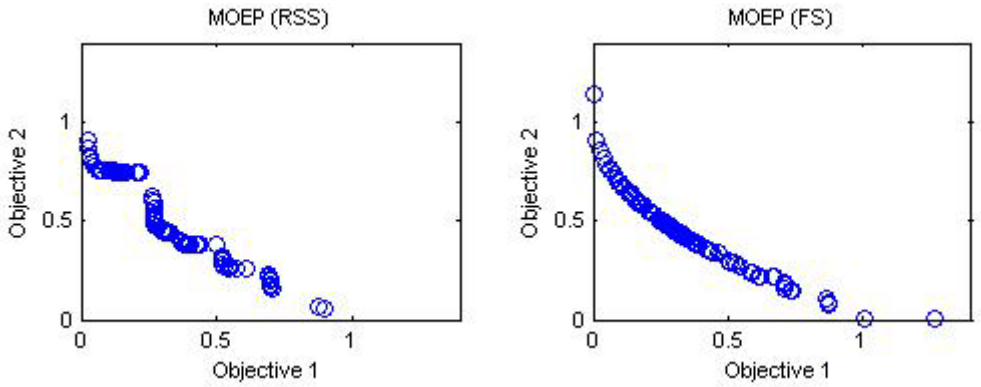

Fig. 4 Unconstrained function 1 (left) and 2 (right)

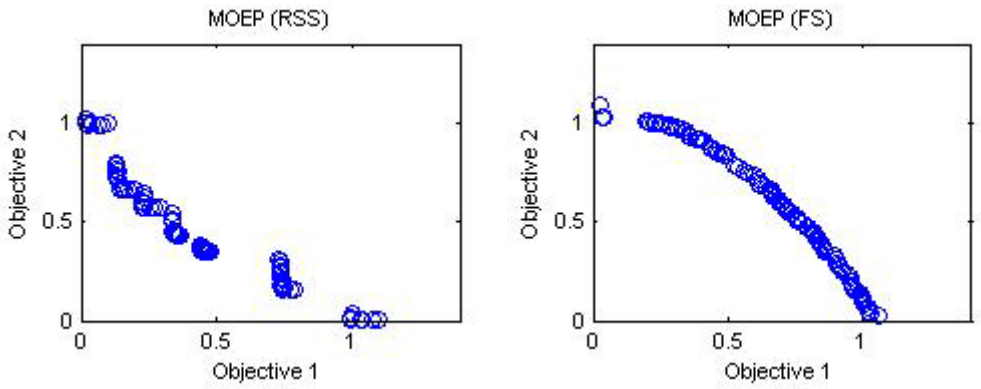

Fig. 5 Unconstrained function 3 (left) and 4 (right)

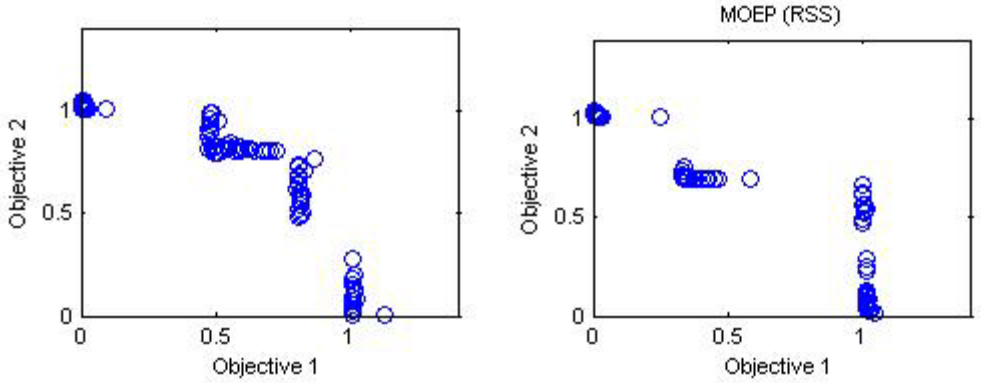

Fig. 6 Unconstrained function 5 (left) and 6 (right)

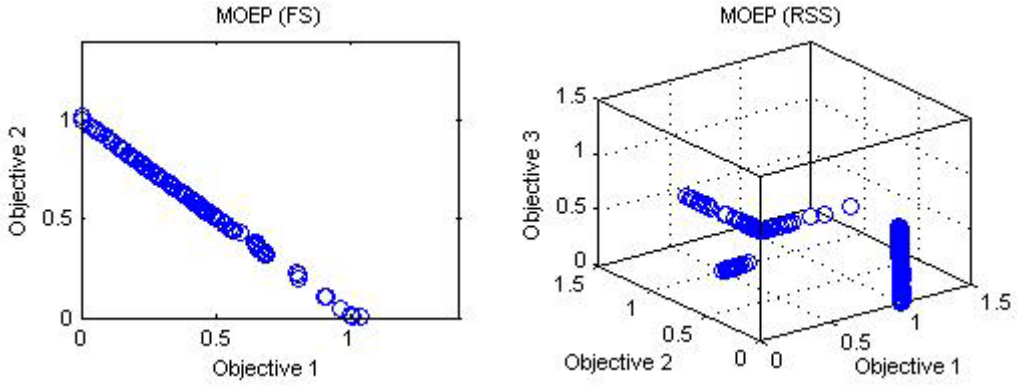

Fig. 7 Unconstrained function 7 (left) and 8 (right)

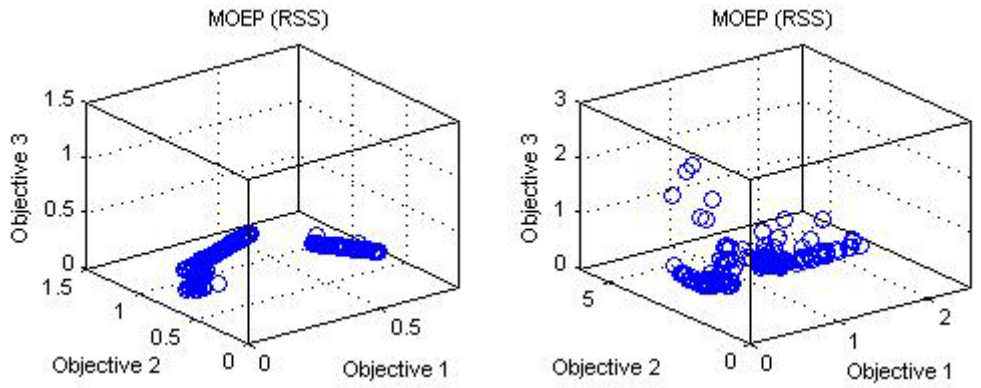

Fig. 8 Unconstrained function 9 (left) and 10 (right)

Table II. Comparsion between 2 algorithms

		moep (rank-sorting)	CPU time	moep (non-domination sorting)	CPU time
UF1	best	0.0354	2 mins 30 secs	0.0301	19 mins 46 secs
	worst	0.074		0.2237	
	mean	**0.0596**		**0.0588**	
	std	0.0128		0.0468	
UF2	best	0.0117	2 mins 25 secs	0.0347	19 mins 25 secs
	worst	0.0283		0.0694	
	mean	**0.0189**		**0.0516**	
	std	0.0038		0.0158	
UF3	best	0.0790	2 mins 35 secs	0.0714	19 mins 46 secs
	worst	0.1408		0.3452	
	mean	**0.0990**		**0.1910**	
	std	0.0132		0.1063	
UF4	best	0.0409	2 mins 39 secs	0.0554	19 mins 10 secs
	worst	0.0447		0.0805	
	mean	**0.0427**		**0.0624**	
	std	8.3463e-004		0.0070	
UF5	best	0.1702	2 mins 40 secs	0.3027	20 mins 10 secs
	worst	0.2670		1.0486	
	mean	**0.2245**		**0.7608**	
	std	0.0344		0.206	
UF6	best	0.0710	2 mins 40 secs	0.0716	21 mins 30 secs
	worst	0.1695		0.4845	
	mean	**0.1031**		**0.3606**	
	std	0.0345		0.1246	
UF7	best	0.0182	2 mins 35 secs	0.0318	19 mins 35 secs
	worst	0.0214		0.0537	
	mean	**0.0197**		**0.0408**	
	std	7.5059e-004		0.0074	
UF8	best	0.2836	2 mins 50 secs	0.4445	27 mins 05 secs
	worst	0.6691		0.9710	
	mean	**0.4230**		**0.6512**	
	std	0.0565		0.2278	
UF9	best	0.1048	2 mins 55 secs	0.1738	27 mins 50 secs
	worst	0.6736		0.4365	
	mean	**0.3420**		**0.2744**	
	std	0.1584		0.1005	
UF10	best	0.2845	2 mins 55 secs	2.1762	29 mins 25 secs
	worst	0.4757		2.7291	
	mean	**0.3621**		**2.4987**	
	std	0.0444		0.2190	
DTLZ2	best	0.3799	6 mins 20 secs	0.9680	125 mins 40 secs
	worst	0.5350		1.1972	
	mean	**0.4337**		**1.0826**	
	std	0.0354		0.0808	
DTLZ3	best	659.2100	6 mins 20 secs	1.0791e+003	125 mins 30 secs
	worst	1.0333e+003		1.2765e+003	
	mean	**885.8932**		**1.1866e+003**	
	std	77.6584		68.0830	

WFG1	best	1.9835	6 mins 15 secs	2.0602	121 mins 55 secs
	worst	2.0366		2.1081	
	mean	**2.0145**		**2.0806**	
	std	0.0137		0.0117	

As evidenced by the table and figures, the performances of the rank-sum sorting method is either comparable or better than the performances of non-domination sorting method. Especially for large number of objectives (5), the rank-sum sorting appears to be superior on the three considered problems. Beside these, the CPU time with rank-sum sorting is much lesser compared to those with the non-domination sorting procedure. For 2 objectives, rank-sum is around 7 times faster than non-domination sorting; while for 3 objectives and 5 objectives are 9 times and 20 times faster, respectively.

V. CONCLUSION

This paper presents the multi-objective evolutionary programming with rank-sum sorting method to reduce the complexity of the commonly used non-domination sorting. In order to achieve better diversity and to converge to the Pareto front, we also introduces the preferential set and backup set concept for parents selection. Through the combination of rank-sum and diversity improvement, the proposed algorithms performs better than the one with the non-domination sorting in terms of computation speed and quality of the generated solutions.

REFERENCES

[1] W. F. Leong and G. G. Yen, "Dynamic population size in PSO-based multiobjective optimization," Piscataway, NJ 08855-1331, United States, 2006, pp. 1718-1725.

[2] J. Liu, J. e. Li, K. Liu, and W. Wei, "A hybrid genetic and particle swarm algorithm for service composition," Piscataway, NJ 08855-1331, United States, 2007, pp. 564-567.

[3] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: empirical results," *Evolutionary Computation*, vol. 8, pp. 173-95, 2000.

[4] L. J. Fogel, A. J. Owens, and M. J. Walsh, "Artificial Intelligence Through Simulated Evolution," New York: Wiley, 1996.

[5] D. B. Fogel, "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence," New York: IEEE Press, 1995.

[6] R. Mallipeddi and P. N. Suganthan, "Evaluation of Novel Adaptive Evolutionary Programming on Four Constraint Handling Techniques," in *IEEE Congress on Evolutionary Computation*, Hong Kong, China, 2008, pp. 4045-4052.

[7] Y. Xin and L. Yong, "Fast evolutionary programming," in *Evolutionary Programming V. Proceedings of the Fifth Annual Conference on Evolutionary Programming*, San Diego, CA, USA, 1996, pp. 451-60.

[8] P. Venkatesh and K. Y. Lee, "Multi-Objective Evolutionary Programming for Economic Emission Dispatch problem," in *IEEE Power and Energy Society 2008 General Meeting: Conversion and Delivery of Electrical Energy in the 21st Century, PES*, Pittsburgh, PA, United States, 2008, p. 4596896.

[9] D. W. Corne, J. D. Knowles, and M. J. Oates, "The Pareto envelope-based selection algorithm for multiobjective optimization," in *Parallel Problem Solving from Nature PPSN VI. 6th International Conference. Proceedings (Lecture Notes in Computer Science Vol.1917)*, Paris, France, 2000, pp. 839-48.

[10] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Parallel Problem Solving from Nature PPSN VI. 6th International Conference. Proceedings (Lecture Notes in Computer Science Vol.1917)*, Paris, France, 2000, pp. 849-58.

[11] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched Pareto genetic algorithm for multiobjective optimization," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence (Cat. No.94TH0650-2)*, Orlando, FL, USA, 1994, pp. 82-7.

[12] J. Knowles and D. Corne, "The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Washington, DC, USA, 1999, pp. 98-105.

[13] M. Kim, T. Hiroyasu, M. Miki, and S. Watanabe, "SPEA2+: improving the performance of the strength Pareto evolutionary algorithm 2," in *Parallel Problem Solving from Nature - PPSN VIII. 8th International Conference. Proceedings (Lecture Notes in Comput. Sci. Vol.3242)*, Birmingham, UK, 2004, pp. 742-51.

[14] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 257-71, 1999.

[15] F. di Pierro, K. Soon-Thiam, and D. A. Savic, "An investigation on preference order ranking scheme for multiobjective evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 17-45, 2007.

[16] K. Deb, "Multi-objective Optimization using Evolutionary Algorithms," West Sussex: John Wiley & Sons, 2001.

[17] Qingfu Zhang, Aimin Zhou, S. Z. Zhao, P. N. Suganthan, Wudong Liu, and S. Tiwari, "Multiobjective optimization Test Instances for the CEC 2009 Special Session and Competition," University of Essex, Colchester, UK and Nanyang Technological University, Singapore, Special Session on Performance Assessment of Multi-Objective Optimization Algorithms, Technical Report 2008.