

# Using Crowding Distance to Improve Multi-Objective PSO with Local Search

Ching-Shih Tsou<sup>1</sup>, Shih-Chia Chang<sup>1</sup> and Po-Wu Lai<sup>2</sup>

<sup>1</sup>*Department of Business Administration, National Taipei College of Business*

<sup>2</sup>*Department of Information Management, Shih Hsin University  
Taiwan*

## 1. Introduction

Biology inspired algorithms have been gaining popularity in recent decades and beyond. These methods are based on biological metaphor such as Darwinian evolution and swarm intelligence. One of the most recent algorithms in this category is the Particle Swarm Optimization (PSO). PSO is a population-based approach using a set of candidate solutions, called particles, which move within the search space. The trajectory followed by each particle is guided by its own experience as well as by its interaction with other particles. Specific methods of adjusting the trajectory are motivated by the observations in birds, fishes, or other organisms that move in swarms.

Multi-objective optimization (MOO) is an important field to apply swarm intelligence meta-heuristics because there is not only one solution for MOO in general. The solution of a MOO problem is generally referred as a non-dominated solution, which is different from the optimal solution of single-objective optimization problem. A solution is said to be non-dominated over another only if it has superior, at least no inferior, performance in all objectives. Hence, non-dominance means that the improvement of one objective could only be achieved at the expense of other objectives. This concept always gives not a single solution, but rather a set of solutions called the non-dominated set or non-dominated archive.

Generally speaking, there are two approaches to MOO: classical methods and evolutionary methods. Classical methods first convert separate objective functions into a single objective function by weighted sum method, utility function method, or goal programming method, and then solve them by traditional optimization techniques. Such modelling puts the original problem in an inadequate manner, using a surrogate variable with incomplete information. Subsequent optimization techniques also contradicts our intuition that single-objective optimization is a degenerate case of MOO (Deb, 2001). The result of classical approach is a compromise solution whose non-dominance can not be guaranteed (Liu et al., 2003). Lastly, but not the least, a single optimized solution could only be found in each simulation run of traditional optimization techniques such that it limits the choices available to the decision maker. Therefore, using a population of solutions to evolve towards several non-dominated solutions in each run makes evolutionary algorithms, such as swarm intelligence methods, popular in solving MOO problems.

Source: Swarm Intelligence: Focus on Ant and Particle Swarm Optimization, Book edited by: Felix T. S. Chan and Manoj Kumar Tiwari, ISBN 978-3-902613-09-7, pp. 532, December 2007, Itech Education and Publishing, Vienna, Austria

One of the successful applications of PSO to MOO problems, named Multi-Objective PSO (MOPSO), is the seminal work of Coello-Coello and Lechuga (2002). In a subsequent study done by them, MOPSO is not only a viable alternative to solve MOO problems, but also the only one, compared with the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) (Deb et al., 2002), the Pareto Archive Evolutionary Strategy (PAES) (Knowles and Corne, 2000), and the micro-Genetic Algorithm (microGA) (Coello-Coello and Pulido, 2001) for MOO, can cover the full Pareto-optimal front of all the test functions therein.

The goal of generating the non-dominated front is itself multi-objective. That is, designing a Pareto optimizer usually has two major goals - how to converge to the true Pareto-optimal front while achieving a well-distributed set of solutions (Zitzler et al., 2004). Tsou et al. (2006) proposed an improved MOPSO with local search and clustering procedure trying to attain to these goals. Although the best way to obtain a well-distributed set of solutions would be probably to use some clustering algorithm, this effort is usually computationally expensive (Kukkonen and Deb, 2006). This paper extends the research of Tsou et al. (2006), but the clustering algorithm is dropped out. A local search and flight mechanism based on crowding distance is incorporated into the MOPSO. The local search procedure intends to explore the less-crowded area in the current archive to possibly obtain more non-dominated solutions nearby. Besides this, the non-dominated solutions in the less-crowded area are used to guide the population fly over sparse area of the current archive. Such that a more uniform and diverse front might be formed by the optimizer. In a short, mechanisms based on the crowding distance not only implicitly maintain the diversity of the external archive, but also facilitate the convergence of MOPSO to the true Pareto-optimal front. Our approach seeks to reach a reasonable compromise between the computational simplicity and efficiency. Several test problems are employed to verify the performance of our approach.

The rest of this paper is organized as follows. Section 2 reviews the basic concept of MOO. MOPSO with a random line search is described in Section 3.1. Extensions based on crowding distance are presented in Section 3.2. Section 4 reports the experimental results against four test problems. Finally, conclusions and future research are drawn out in Section 5.

## 2. Multi-objective Optimization

Without loss of generality, a MOO problem (also known as a vector optimization problem) is the problem of simultaneously minimizing  $K$  objectives  $f_k(\bar{\mathbf{x}})$ ,  $k = 1, 2, \dots, K$ , of a vector  $\bar{\mathbf{x}}$  in the feasible region  $\Omega$ . That is,

$$\text{Vector minimize}_{\bar{\mathbf{x}} \in \Omega} \bar{\mathbf{f}}(\bar{\mathbf{x}}) = [f_1(\bar{\mathbf{x}}), f_2(\bar{\mathbf{x}}), \dots, f_K(\bar{\mathbf{x}})]^T \quad (1)$$

, where  $\bar{\mathbf{x}} = [x_1, x_2, \dots, x_D]^T$  is a  $D$ -dimensional vector and  $f_k(\bar{\mathbf{x}})$  ( $k = 1, 2, \dots, K$ ) are linear or nonlinear functions. A decision vector  $\bar{\mathbf{u}} = (u_1, u_2, \dots, u_D)$  is said to strongly dominate  $\bar{\mathbf{v}} = (v_1, v_2, \dots, v_D)$  (denoted by  $\bar{\mathbf{u}} \prec \bar{\mathbf{v}}$ ) if and only if  $\forall i \in \{1, 2, \dots, K\}$ ,  $f_i(\bar{\mathbf{u}}) < f_i(\bar{\mathbf{v}})$ . Less stringently, a decision vector  $\bar{\mathbf{u}}$  weakly dominates  $\bar{\mathbf{v}}$  (denoted by  $\bar{\mathbf{u}} \preceq \bar{\mathbf{v}}$ ) if and only if  $\forall i \in \{1, 2, \dots, K\}$ ,  $f_i(\bar{\mathbf{u}}) \leq f_i(\bar{\mathbf{v}})$  and  $\exists i \in \{1, 2, \dots, K\}$ ,  $f_i(\bar{\mathbf{u}}) < f_i(\bar{\mathbf{v}})$ .

Certainly, we are not interested in solutions dominated by other solutions. A set of decision vectors is said to be a non-dominated set if no member of the set is dominated by any other member. The true Pareto-optimal front is the non-dominated set of solutions which are not dominated by any feasible solution. One way to solving a MOO problem is to approximate the Pareto-optimal front by the non-dominated solutions generating from the solution algorithm.

### 3. MOPSO with Local Search

To speak of MOPSO, let us start with the PSO. In PSO, a population is initialized with random solutions, called "particles". All particles have fitness values that are evaluated by the function to be optimized. Each particle flies through the problem space with a velocity, which is constantly updated by the particle's own experience and the experience of the particle's neighbors, to search for optima iterations by iterations. Compared to genetic algorithms, the advantages of PSO are that it is easy to implement and there are fewer parameters to adjust.

In every iteration, the velocity of each particle is updated by two best values. The first one is the best solution it has achieved so far. This value is called *pbest*. Another best value tracked by the optimizer is the best value obtained so far by the neighbourhood of each particle. This best value is a local best and is called *lbest*. If the neighbourhood is defined as the whole population, each particle will move towards its best previous position and towards the best position ever been in the whole swarm, this version is called *gbest* model. In this paper, we use the global version of PSO. The velocity and position of each particle are updated by the following equations.

$$v_d^{i(new)} = \omega \cdot v_d^{i(old)} + c_1 \cdot RAND \cdot (p_d^i - x_d^i) + c_2 \cdot RAND \cdot (g_d - x_d^i) \quad (2)$$

$$x_d^{i(new)} = x_d^i + v_d^{i(new)}, \quad (3)$$

where

$v_d^{i(old)}$  is the old velocity of particle  $i$  along dimension  $d$ ,

$v_d^{i(new)}$  is the new velocity of particle  $i$  along dimension  $d$ ,

$\omega$  is the inertia weight which is usually between 0.8 and 1.2,

$c_1$  and  $c_2$  are the learning factors (or acceleration coefficients), usually between 1 and 4,

$x_d^i$  is the current position of particle  $i$  along dimension  $d$ ,

$p_d^i$  is the personal best solution of particle  $i$  along dimension  $d$ ,

$g_d$  is the global best solution the whole population ever been along dimension  $d$ , and

$RAND$  is a random number between 0 and 1.

The difficulty in extending the PSO to MOO problems is how to select a global guide for each particle. Because there is not a single optimum in MOO, the non-dominated solutions found by MOPSO so far are all stored in an archive. Each particle can randomly select a non-dominated solution from the archive as the global guide of its next flight. Although this selection method is simple, it can promote convergence (Alvarez-Benitez et al., 2005). The pseudo-code of MOPSO is shown in Fig. 1.

```
MOPSO()
01: Initialize()
02: iter ← 1
03: while iter < MAXITER do
04:   Flight()
05:   CalculateObjVector()
06:   UpdateNondominatedSet()
07:   iter = iter + 1
08: end while
```

Figure 1. The pseudo-code of MOPSO

### 3.1 Local search

Local search plays a role in adding an exploitative component allows algorithms to make use of local information to guide the search towards better regions in the search space. This feature leads to faster convergence with less computational burden. One of the simplest local search algorithms is the random line search. It starts with calculating the maximum step length according to the parameter  $\delta$ . For a non-dominated solution, improvement is sought coordinate by coordinate. The temporary  $D$ -dimensional vector,  $\bar{z}$ , first holds the initial information of each particle. Next, two random numbers are generated to set moving direction and step length for each coordinate, respectively. If the vector  $\bar{z}$  observes a better non-dominated solution, the non-dominated set is updated and the local search for particle  $i$  ends. The local search procedure (shown in Fig. 2) incorporated into the MOPSO is so called MOPSO-LS.

```

LocalSearch( $\delta$ )
01:  $L = \delta \cdot \left( \max_d \{u^d - l^d\} \right)$ 
02:  $\tilde{S} = \text{random}(10\% \text{ of } \tilde{A})$ 
03: for  $i = 1$  to  $|\tilde{S}|$ 
04:    $\bar{\mathbf{z}} = \bar{\mathbf{x}}^i$ 
05:   for  $d = 1$  to  $D$  do
06:      $\lambda_1 = \text{RAND}(0,1)$ 
07:      $\lambda_2 = \text{RAND}(0,1)$ 
08:     if  $\lambda_1 > 0.5$  then
09:        $z^d = z^d + \lambda_2 L$ 
10:     else
11:        $z^d = z^d - \lambda_2 L$ 
12:     end if
13:   end for
14:   CalculateObjVector( $\bar{\mathbf{z}}$ )
15:   if  $\bar{\mathbf{z}} \prec \bar{\mathbf{x}}^i$  or  $\bar{\mathbf{z}} \prec_{\text{crowd}} \bar{\mathbf{x}}^i$  then
16:     UpdateNondominatedSet( $\bar{\mathbf{z}}$ )
17:      $\bar{\mathbf{x}}^i = \bar{\mathbf{z}}$ 
18:   end if
19: end for

```

Figure 2. The pseudo-code of local search procedure

### 3.2 Enhancements from crowding distance

The crowding distance of a non-dominated solution provides an estimate of the density of solutions surrounding it (Deb et al., 2002). It is calculated by the size of the largest cuboid enclosing each particle without including any other point. After normalizing the crowding distance for each non-dominated solution, we sort them in ascending order (Line 01 in Fig. 4). As mentioned earlier, the selection of global guide is a critical step in MOPSO. It affects both the convergence to the true Pareto-optimal front and a well-distributed front. Instead of randomly choosing a global guide from the whole non-dominated archive, it is randomly selected from the top 10% less crowded area of the archive for each particle that is dominated by any solution located in this area. Global guides of other particles are randomly selected from the whole archive as usual. This is the flight procedure used in MOPSO-CDLS (Line 03 in Fig. 4). Raquel and Naval (2005) were the first ones to incorporate the crowding distance into the global best selection in MOPSO, however, each particle associated with its own global guide solely selected from the top 10% less crowded area of the archive. It is too restrictive for those particles far away from the less crowded area and could possibly perturb their happy flight.

```

Flight()
01: SortArchiveByCrowdingDistance()
02: for  $i = 1$  to  $m$  do
03:   if  $\bar{x}^i$  is dominated by the top 10% less crowded area in  $\tilde{A}$ 
       then
04:      $(\bar{x}^i)^{Gbest} = \text{Random}(\text{top 10\% less crowded area in } \tilde{A})$ 
05:   else
06:      $(\bar{x}^i)^{Gbest} = \text{Random}(\tilde{A})$ 
07:   end if
08:   for  $d = 1$  to  $D$  do
09:      $v_d^i \leftarrow \omega v_d^i + c_1 \cdot \text{RAND} \cdot (p_d^i - x_d^i) + c_2 \cdot \text{RAND} \cdot (g_d^i - x_d^i)$ 
10:      $x_d^i \leftarrow x_d^i + v_d^i$ 
11:   end for
12:   CalculateObjVector( $\bar{x}^i$ )
13:   UpdateNondominatedSet( $\bar{x}^i$ )
14: end for

```

Figure 4. The pseudo-code of flight procedure

Besides the flight mechanism based on crowding distance, the local search procedure is also modified to only be executed on the non-dominated solutions in the top 10% less crowded area of the archive. That is, Line 02 in Fig. 2 is modified as  $\tilde{S} = \text{top } 10\% \text{ less crowded area of } \tilde{A}$ . It is expected that better solutions, at least non-dominated, could be found by the random line search around the less crowded area. Dual effects of pushing further towards the true Pareto-optimal front as well as maintaining a diverse and well-distributed archive might be arisen.

#### 4. Experimental Results

The well-known ZDT test problems (Zitzler et al., 2000) were used to validate the MOPSO-CDLS. ZDT1 is an easy bi-objective problem and has a convex and continuous Pareto-optimal front. ZDT2 has a non-convex but still continuous front. The front of ZDT3 is convex, however, it is discontinuous. In other words, it has several disconnected Pareto-optimal front. The last test problem, ZDT4, is convex but has many local fronts.

The population size for MOPSO-LC and MOPSO-CDLC are set to 25 with a step size 25 till 75. The numbers of iterations are set to 30 with a step size 10 till 50. To compare all results in a quantitative way, we use the following performance measures: archive count  $|\tilde{A}|$ , set coverage metric  $C(U, V)$ , spacing ( $S$ ), and maximum spread ( $D$ ) (Okabe et al., 2002).

$$C(U, V) = \frac{|\{b \in V \mid \exists a \in U : a \preceq b\}|}{|V|} \quad (4)$$

, where  $|\cdot|$  means the number of components in the set.

$$S = \sqrt{\frac{1}{|\tilde{A}|} \sum_{i=1}^{|\tilde{A}|} (d_i - \bar{d})^2} \quad (5)$$

, where  $d_i = \min_{j \in \tilde{A}, j \neq i} \sum_{k=1}^K |f_k^i - f_k^j|$  and  $\bar{d}$  is the mean value of the absolute distance measure

$$\bar{d} = \frac{1}{|\tilde{A}|} \sum_{i=1}^{|\tilde{A}|} d_i$$

$$D = \sqrt{\sum_{k=1}^K \left( \max_{i=1}^{|\tilde{A}|} f_k^i - \min_{i=1}^{|\tilde{A}|} f_k^i \right)^2} \quad (6)$$

Tables 1-4 are the results of four test problems for both algorithms. C and L in the parentheses of the first row stand for MOPSO-CDLS and MOPSO-LS, respectively. Some findings are explained in the following.

Iter.	Pop.	C(C,L)	C(L,C)	S(C)	S(L)	D(C)	D(L)	Count(C)	Count(L)	Time(C)	Time(L)
30	25	0.901639	0	0.170416	0.088285	1.417687	1.52305	61	55	0.89	0.843
30	50	0.785714	0	0.029987	0.086507	1.414402	1.519982	70	55	1.078	0.969
30	75	0.071429	0.114286	0.026999	0.033471	1.414214	1.414214	70	68	1.203	1.047
40	25	0.125	0.138889	0.045418	0.044254	1.41424	1.414214	72	73	1.313	1.188
40	50	0.811594	0	0.057449	0.064995	1.414903	1.519977	69	56	1.406	1.313
40	75	0.830986	0	0.059604	0.077409	1.414214	1.519981	71	59	1.672	1.391
50	25	0.876923	0	0.045133	0.08563	1.414563	1.519915	65	57	1.656	1.468
50	50	0.104478	0.044776	0.04486	0.063149	1.414214	1.414214	67	68	1.937	1.671
50	75	0.1	0.1125	0.046662	0.067414	1.414214	1.414214	80	72	2.125	1.797

Table 1. Computational results of ZDT1 problem

Iter.	Pop.	C(C,L)	C(L,C)	S(C)	S(L)	D(C)	D(L)	Count(C)	Count(L)	Time(C)	Time(L)
30	25	0.609756	0	0.074917	0.123052	1.414265	1.417507	41	34	0.921	0.843
30	50	0.885714	0	0.13882	0.087169	1.414214	1.3933	35	34	0.985	0.906
30	75	0.125	0.025	0.093266	0.116607	1.414214	1.414063	40	35	1.047	0.937
40	25	0.939394	0	0.108248	0.14694	1.413647	1.373224	33	31	1.25	1.109
40	50	0.228571	0.028571	0.123745	0.138773	1.414214	1.414332	35	34	1.344	1.25
40	75	0.222222	0.027778	0.052225	0.10615	1.414212	1.414	36	37	0.688	0.594
50	25	0.894737	0	0.05814	0.104946	1.412791	1.392065	38	36	1.593	1.453
50	50	0.27027	0.027027	0.047095	0.110726	1.414142	1.414255	37	35	1.015	0.797
50	75	0.228571	0	0.123088	0.138386	1.414227	1.414231	35	40	1.781	1.562

Table 2. Computational results of ZDT2 problem

Iter.	Pop.	C(C,L)	C(L,C)	S(C)	S(L)	D(C)	D(L)	Count(C)	Count(L)	Time(C)	Time(L)
30	25	0.473684	0	0.032093	0.291718	0.335081	1.864763	19	18	0.672	0.734
30	50	0.410256	0.025641	0.093437	0.337072	1.95883	1.884636	39	36	0.922	0.812
30	75	0.142857	0.119048	0.053984	0.107845	1.961352	1.927156	42	51	1.047	0.891
40	25	0.5	0.033333	0.122377	0.53802	1.934344	1.947032	30	20	1.094	0.969
40	50	0.2	0.228571	0.030287	0.159101	1.950184	1.94119	35	41	1.203	1.141
40	75	0.163265	0.204082	0.006431	0.008037	1.95511	1.949345	49	47	1.328	1.235
50	25	0.870968	0	0.37742	0.183748	1.953586	2.028031	31	27	1.438	1.313
50	50	0.266667	0.088889	0.097678	0.169007	1.931062	1.955391	45	43	1.609	1.453
50	75	0.148936	0.170213	0.16836	0.135811	1.958623	1.962591	47	53	1.688	1.531

Table 3. Computational results of ZDT3 problem

Iter.	Pop.	C(C,L)	C(L,C)	S(C)	S(L)	D(C)	D(L)	Count(C)	Count(L)	Time(C)	Time(L)
30	25	0.583333	0.027778	0.085428	0.120843	1.415008	1.431134	36	29	0.844	0.782
30	50	0.428571	0.071429	0.070559	0.070263	1.438755	1.429762	42	31	0.875	0.813
30	75	0.513514	0.162162	0.054137	0.11659	1.409016	1.430743	37	36	0.922	0.844
40	25	0.612903	0.032258	0.124403	0.212547	1.417124	1.453333	31	26	1.093	1.078
40	50	0.452381	0.214286	0.083199	0.076452	1.412184	1.410637	42	39	1.172	1.156
40	75	0.589286	0.142857	0.073391	0.101297	1.422002	1.416985	56	54	1.282	1.156
50	25	0.575758	0.212121	0.088226	0.082615	1.440936	1.437883	33	33	1.438	1.328
50	50	0.565217	0.173913	0.079993	0.097561	1.43185	1.429264	46	47	1.562	1.422
50	75	0.433333	0.1	0.037096	0.077393	1.42348	1.417994	60	59	1.656	1.516

Table 4. Computational results of ZDT4 problem

1. In view of the set coverage metric in Tables 1-4, MOPSO-CDLS exhibit better results than MOPSO-LS even in more difficult problem such as ZDT3 and ZDT4. That is, the non-dominated solutions generated by MOPSO-CDLS are closer to the Pareto-optimal front than those by MOPSO-LS.

2. For the maximum spread in Tables 1-4, there is no significant difference for both algorithms. However, MOPSO-CDLS outperforms MOPSO-LS in the spacing metric. This implies MOPSO-CDLS can generate well-distributed front than MOPSO-LS.
3. It is not surprising that particles flying towards sparse area and gathering local information around it make MOPSO-CDLS find more non-dominated solutions than MOPSO-LS on the average.
4. Certainly, crowding distance calculation need additional time to execute. Although the execution time (in second) of MOPSO-CDLS is a little bit longer than that of MOPSO-LS in all tables, MOPSO-CDLS is still a reasonable simple and efficient algorithm for MOO.

## 5. Conclusions

It is well known that local search, even in its simplest form, prevents search algorithms from premature convergence and, therefore, possibly drives the solution closer to true Pareto-optimal front. A local search procedure and a flight mechanism both based on crowding distance are incorporated into the MOPSO, so called MOPSO-CDLS, in this paper. Computational results against ZDT1-4 problems show that it did improve the MOPSO with random line search in all aspects except the execution time. Local search in less crowded area of the front not only reserves the exploitation capability, but also helps to achieve a well-distributed non-dominated set. Global guides randomly selected from the less crowded area help the particles dominated by the solutions in this area to explore more diverse solutions and in a hope to better approximate the true front.

This study intends to highlight a direction of combining more intelligent local search algorithms into a Pareto optimization scheme. Mechanisms based on crowding distance employed here did not explicitly maintain the diversity of non-dominated solutions which is its original intention, but they indeed facilitate the possibilities of flying towards the Pareto-optimal front and generating a well-distributed non-dominated set. Further researches include comparisons with other multi-objective evolutionary algorithms and accommodating constraints-handling mechanism in the Pareto optimizer.

## 6. References

- Alvarez-Benitez, J.E., Everson, R.M. & Fieldsend, J.E. (2005). A MOPSO algorithm based exclusively on Pareto dominance concepts. *Lecture Notes in Computer Science*, 3410, 459-473.
- Coello Coello, C.A. and Lechuga, M.S. (2002), MOPSO: A proposal for multiple objective particle swarm optimization, *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, pp. 12-17, Honolulu, U.S.A., May 2002, IEEE Press, New Jersey.
- Coello Coello, C.A. & Pulido, G.T. (2001), Multiobjective optimization using a micro-genetic algorithm, *Proceedings of the 2001 Conference on Genetic and Evolutionary Computation Conference*, pp. 274-282, San Francisco, CA, July 2001, IEEE Press, New Jersey.
- Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, New York.
- Deb, K.; Pratap, A.; Agarwal S. & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, Vol. 6, 182-197.

- Knowles, J.D. & Corne, D.W. (2000). Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation Journal*, Vol. 8, 149-172.
- Kukkonen, S. And Deb, K. (2006). Improved pruning of non-dominated solutions based on crowding distance for bi-objective optimization problems, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pp.1179-1186, Vancouver, Canada, July 2006, IEEE Press, New Jersey.
- Liu, G.P.; Yang, J.B. & Whidborne, J.F. (2003). *Multiobjective Optimisation and Control*, Research Studies Press, Hertfordshire.
- Okabe, T.; Jin, Y. & Sendhoff, B. (2003). A critical survey of performance indices for multi-objective optimization, *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, pp. 878-885, December 2003, IEEE Press, New Jersey.
- Raquel, C.R. & Naval, P.C. Jr. (2005). An effective use of crowding distance in multiobjective particle swarm optimization, *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pp. 257-264, Washington D.C., U.S.A., June 2005, ACM Press, New York.
- Tsou, C.-S.; Fang, H.-H.; Chang, H.-H. & Kao, C.-H. (2006). An improved particle swarm Pareto optimizer with local search and clustering. *Lecture Notes in Computer Science*, 4247, 400-406.
- Zitzler, E.; Deb, K. & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms : empirical results. *Evolutionary Computation Journal*, Vol. 8, No. 2, 125-148.
- Zitzler, E.; Laumanns, M. & Bleuler, S. (2004). A tutorial on evolutionary multiobjective optimization, in Gandibleux, X.; Sevaux, M.; Sörensen, K. & T'kindt, V. (Ed.), *Metaheuristics for Multiobjective Optimisation*, Springer, Heidelberg, pp. 3-37.