An Effective Use of Crowding Distance in Multiobjective Particle Swarm Optimization

Carlo R. Raquel
Dept. of Mathematics and Computer Science
University of the Philippines-Baguio
Gov. Pack Road, Baguio City, Philippines
+(63-74) 442-7231
crraquel@up.edu.ph

Prospero C. Naval, Jr.
Dept. of Computer Science
University of the Philippines-Dilliman
Diliman, Quezon City, Philippines
+(63-2) 981-85-00 ext 3030
pcnaval@up.edu.ph

ABSTRACT
In this paper, we present an approach that extends the Particle Swarm Optimization (PSO) algorithm to handle multiobjective optimization problems by incorporating the mechanism of crowding distance computation into the algorithm of PSO, specifically on global best selection and in the deletion method of an external archive of nondominated solutions. The crowding distance mechanism together with a mutation operator maintains the diversity of nondominated solutions in the external archive. The performance of this approach is evaluated on test functions and metrics from literature. The results show that the proposed approach is highly competitive in converging towards the Pareto front and generates a well distributed set of nondominated solutions.

Categories and Subject Descriptors
I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search – *Heuristic Methods*

General Terms: Algorithms

Keywords
Multiobjective Optimization, Particle Swarm Optimization, Crowding Distance

1. INTRODUCTION
Many real-world optimization problems have multiple objectives which are not only interacting but even possibly conflicting. In general, a multiobjective minimization problem with *m* decision variables (parameters) and *n* objectives can be stated as:

$$\text{minimize } y = f(x) = (f_1(x),..., f_n(x))$$

$$\text{where } x = (x_1,..., x_m) \in X$$

$$y = (y_1,..., y_n) \in Y$$

where x is called decision vector, X the parameter space, y the objective vector and Y the objective space. The desired solution is in the form of "trade-off" or compromise among the parameters that would optimize the given objectives. The optimal trade-off solutions among objectives constitute the Pareto front.

Multiobjective optimization deals with generating the Pareto front which is the set of non-dominated solutions for problems having more than one objective. A solution is said to be non-dominated if it is impossible to improve one component of the solution without worsening the value of at least one other component of the solution. The goals of multiobjective optimization are: (1) to guide the search toward the true Pareto front (non-dominated solutions) or approximate the Pareto optimal set, and (2) to generate a well-distributed Pareto front.

Many evolutionary algorithms (EAs) have been developed in solving multiobjective optimization problems such as Micro-GA [2], NSGA-II [4], PAES [9] and SPEA2 [16]. These EAs are population-based algorithms which allow them to explore the different parts of the Pareto front simultaneously.

Several multiobjective optimization algorithms are based on Particle Swarm optimization (PSO) [7] which was originally designed for solving single objective optimization problems. PSO is an algorithm inspired by the social behavior of bird flocking. The initial population of particles is initialized with random solutions. For every generation, each solution moves toward the global Pareto front by updating its velocity, the best solution a particle has achieved so far and follows the best solution achieved among the population of solutions.

Among those algorithms that extend PSO to solve multiobjective optimization problems are Multiobjective Particle Swarm Optimization (MOPSO) [1], Nondominated Sorting Particle Swarm Optimization (NSPSO) [11], the aggregating function for PSO [12], the algorithm of Fieldsend and Singh [5] that uses an unconstrained archive and a data structure called a "dominated tree" and the use of multilevel sieve for constraint handling [13].

Several techniques have been incorporated into multiobjective optimization algorithms especially to PSO-based algorithms in order to improve convergence to the true Pareto front as well as produce a well-distributed Pareto front. These techniques are elitism, diversity operators, mutation operators, and constraint handling.

The performance of different multiobjective algorithms that incorporate such optimization techniques was compared in [1] using five test functions. These algorithms are NSGA-II, PAES, Micro-GA and MOPSO. The results show that MOPSO was able

GECCO '05, June 25-29, 2005, Washington DC, USA.

to generate the best set of nondominated solutions close to the true Pareto front in all test functions except in one function where NSGA-II is superior. In terms of diversity of the nondominated solutions, NSGA-II produced the best results in all test functions but was not able to cover the entire Pareto front in all test functions. MOPSO was the only algorithm which was able to cover the entire Pareto front.

This remarkable performance of MOPSO can be attributed to its use of an external repository or archive of nondominated solutions found in previous iterations and to its novel mutation operator that initially covers the entire population (including the range of each design variables) which it then gradually decreases during subsequent iterations. The mutation operator improves the exploratory capabilities of the algorithm and prevents premature convergence. This proves that PSO is a powerful optimization algorithm when extended to handle multiobjective optimization problems by using such techniques.

However, it should be noted that the use of the crowded comparison operator, which basically is a computation of the crowding distance of each solution, as a diversity operator by NSGA-II was able to produce a better distribution of the generated nondominated solutions compared to the results generated by MOPSO that uses an adaptive grid in maintaining diversity of the generated solutions. This shows that while MOPSO is superior in converging to the true Pareto front, its diversity mechanism falls behind that of NSGA-II.

The computation time of the algorithms was also observed and MOPSO has been found to have a vastly superior execution time compared to the other algorithms. This is attributed to the adaptive grid used by MOPSO which has lower computational cost [9] than the crowding distance used by NSGA-II combined with the nondominated sorting as suggested by Goldberg [6].

The proposed algorithm extends PSO in solving multiobjective optimization problems by incorporating the mechanism of crowding distance computation in the global best selection and the deletion method of the external archive of nondominated solutions whenever the archive is full. The crowding distance mechanism together with a mutation operator maintains the diversity of nondominated solutions in the external archive.

The remainder of the paper is organized as follows. In Section 2, the proposed algorithm is discussed followed by a discussion of the implementation, experiments done on the algorithm and their results in Section 3. The impact of the mutation operator and replacement of nondominated solutions in the archive on the performance of the proposed algorithm is analyzed in Section 4. The paper closes with the Summary and Conclusion in Section 5.

2. PROPOSED APPROACH

The proposed algorithm which we shall call MOPSO-CD extends the algorithm of the single-objective PSO to handle multiobjective optimization problems. It incorporates the mechanism of crowding distance computation into the algorithm of PSO specifically on global best selection and in the deletion method of an external archive of nondominated solutions. The crowding distance mechanism together with a mutation operator maintains the diversity of nondominated solutions in the external archive. MOPSO-CD also has a constraint handling mechanism for solving constrained optimization problems.

2.1 MOPSO-CD Algorithm

1. For i = 1 to M (M is the population size)
   a. Initialize $P[i]$ randomly
      (P is the population of particles)
   b. Initialize $V[i] = 0$ (V is the speed of each particle)
   c. Evaluate $P[i]$
   d. Initialize the personal best of each particle
      $PBESTS[i] = P[i]$
   e. $GBEST$ = Best particle found in $P[i]$
2. End For
3. Initialize the iteration counter $t = 0$
4. Store the nondominated vectors found in P into A
   (A is the external archive that stores nondominated solutions found in P)
5. Repeat
   a. Compute the crowding distance values of each nondominated solution in the archive A
   b. Sort the nondominated solutions in A in descending crowding distance values
   c. For i = 1 to M
      i. Randomly select the global best guide for $P[i]$ from a specified top portion (e.g. top 10%) of the sorted archive A and store its position to $GBEST$.
      ii. Compute the new velocity:
      $V[i] = W \times V[i] + R_1 \times (PBESTS[i] - P[i]) + R_2 \times (A[GBEST] - P[i])$
      (W is the inertia weight equal to 0.4)
      (R_1 and R_2 are random numbers in the range [0..1])
      ($PBESTS[i]$ is the best position that the particle i have reached)
      ($A[GBEST]$ is the global best guide for each nondominated solution)
      iii. Calculate the new position of $P[i]$:
      $P[i] = P[i] + V[i]$
      iv. If $P[i]$ goes beyond the boundaries, then it is reintegrated by having the decision variable take the value of its corresponding lower or upper boundary and its velocity is multiplied by -1 so that it searches in the opposite direction.
      v. If (t < ($MAXT$ * $PMUT$),
      then perform mutation on $P[i]$.
      (MAXT is the maximum number of iterations)
      (PMUT is the probability of mutation)
      vi. Evaluate $P[i]$
   d. End For
   e. Insert all new nondominated solution in P into A if they are not dominated by any of the stored solutions. All dominated solutions in the archive by the new solution are removed from the archive. If the archive is full, the solution to be replaced is determined by the following steps:
      i. Compute the crowding distance values of each nondominated solution in the archive A
      ii. Sort the nondominated solutions in A in descending crowding distance values
      iii. Randomly select a particle from a specified bottom portion (e.g. lower 10%) which

comprise the most crowded particles in the archive then replace it with the new solution

f. Update the personal best solution of each particle in *P*. If the current *PBESTS* dominates the position in memory, the particles position is updated using *PBESTS[i] = P[i]*

g. Increment iteration counter *t*

6. Until maximum number of iterations is reached

2.2 Crowding Distance Computation

The crowding distance value of a solution provides an estimate of the density of solutions surrounding that solution [4]. Figure 1 shows the calculation of the crowding distance of point *i* which is an estimate of the size of the largest cuboid enclosing *i* without including any other point.

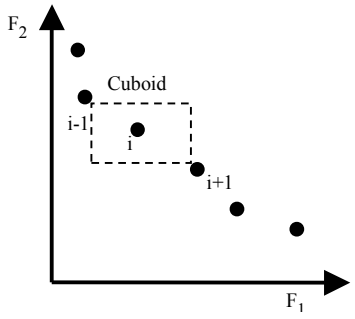

Figure 1. Crowding distance computation.

Crowding distance is calculated by first sorting the set of solutions in ascending objective function values. The crowding distance value of a particular solution is the average distance of its two neighboring solutions. The boundary solutions which have the lowest and highest objective function values are given an infinite crowding distance values so that they are always selected. This process is done for each objective function. The final crowding distance value of a solution is computed by adding the entire individual crowding distance values in each objective function. The pseudocode of crowding distance computation is shown below.

1. Get the number of nondominated solutions in the external repository
   a. n = | S |
2. Initialize distance
   a. FOR i=0 TO MAX
   b. S[i].distance = 0
3. Compute the crowding distance of each solution
   a. For each objective m
   b. Sort using each objective value
      S = sort(S, m)
   c. For i=1 to (n-1)
   d. S[i].distance = S[i].distance + (S[i+1].m – S[i-1].m)
   e. Set the maximum distance to the boundary points so that they are always selected
      S[0].distance = S[n].distance = maximum distance

2.3 Global Best Selection

The selection of the global best guide of the particle swarm is a crucial step in a multiobjective-PSO algorithm. It affects both the convergence capability of the algorithm as well as maintaining a good spread of nondominated solutions. In MOPSO-CD, a bounded external archive stores nondominated solutions found in previous iteration. We note that any of the nondominated solutions in the archive can be used as the global best guide of the particles in the swarm. But we want to ensure that the particles in the population move towards the sparse regions of the search space.

In MOPSO-CD, the global best guide of the particles is selected from among those nondominated solutions with the highest crowding distance values. Selecting different guides for each particle in a specified top part of the sorted repository based on a decreasing crowding distance allows the particles in the primary population to move towards those nondominated solutions in the external repository which are in the least crowded area in the objective space.

Also, whenever the archive is full, crowding distance is again used in selecting which solution to replace in the archive. This promotes diversity among the stored solutions in the archive since those solutions which are in the most crowded areas are most likely to be replaced by a new solution.

2.4 Mutation

The mutation operator of MOPSO was adapted because of the exploratory capability it could give to the algorithm by initially performing mutation on the entire population then rapidly decreasing its coverage over time [1]. This is helpful in terms of preventing premature convergence due to existing local Pareto fronts in some optimization problems.

2.5 Constraint Handling

In order to handle constrained optimization problem, MOPSO-CD adapted the constraint handling mechanism used by NSGA-II due to its simplicity in using feasibility and nondominance of solutions when comparing solutions. A solution i is said to constrained-dominate a solution j if any of the following conditions is true:

1. Solution i is feasible and solution j is not.
2. Both solutions i and j are infeasible, but solution i has a smaller overall constraint violation.
3. Both solutions i and j are feasible and solution i dominates solutions j.

When comparing two feasible particles, the particle which dominates the other particle is considered a better solution. On the other hand, if both particles are infeasible, the particle with a lesser number of constraint violations is a better solution.

2.6 The Time Complexity of MOPSO-CD

The computational complexity of the algorithm is dominated by the objective function computation, crowding distance computation and the nondominated comparison of the particles in the population and in the archive. If there are M objective functions and N number of solutions (particles) in the population, then the objective function computation has O(MN) computational complexity. The costly part of crowding distance computation is sorting the solutions in each objective function. If

there are K solutions in the archive, sorting the solutions in the archive has O(M K log K) computational complexity. If the population and the archive have the same number of solutions, say N, the computational complexity for the nondominated comparison is $O(MN^2)$. Thus, the overall complexity of MOPSO-CD is $O(MN^2)$.

3. EXPERIMENTS

The performance of the proposed algorithm is compared to MOPSO using three test functions and two performance metrics. The computational time of the two algorithms is also evaluated.

3.1 Test Functions

The first test function proposed by Kita [8] is a multiobjective maximization function which has three constraints. This is the only constrained problem used in the experiments.

Maximize $F = (f_1(x,y), f_2(x,y))$, where

$$f_1(x,y) = -x^2 + y, f_2(x,y) = \frac{1}{2}x + y + 1$$

subject to

$$0 \geq \frac{1}{6}x + y - \frac{13}{2}, 0 \geq \frac{1}{2}x + y - \frac{15}{2}, 0 \geq 5x + y - 30$$

and x, y ≥ 0 with a range of $0 \leq x, y \leq 7$.

The second test function was introduced by Kursawe [11] which has three disconnected Pareto curves. Its solution mapping in the objective space is very convoluted.

Minimize $f_1(\vec{x}) = \sum_{i=1}^{n-1}(-10 \exp(-0.2\sqrt{x_i^2 + x_{i+1}^2}))$

Minimize $f_2(\vec{x}) = \sum_{i=1}^{n}(|x_i|^{0.8} + 5\sin(x_i)^3)$

where $-5 \leq x_1, x_2, x_3 \leq 5$.

The third test function used was proposed by Deb [3] has a bimodal function $g(x_2)$ which has local and global minimum values.

Minimize $f_1(x_1, x_2) = x_1$

Minimize $f_2(x_1, x_2) = \frac{g(x_2)}{x_1}$

$$g(x_2) = 2.0 - \exp\left\{-\left(\frac{x_2 - 0.2}{0.004}\right)^2\right\} - 0.8\exp\left\{-\left(\frac{x_2 - 0.6}{0.04}\right)^2\right\}$$

and $0.1 \leq x_1, x_2 \leq 1.0$.

3.2 Performance Metrics

The two performance metrics used for the experiments are the two set coverage and spacing metric. The two set coverage metric (C) proposed by Zitzler et al. maps the ordered pair (A, B) to the interval [0,1] using the following equation:

$$C(A,B) = \frac{\left|\{b \in B; \exists a \in A : a \succeq b\}\right|}{\left|B\right|}$$

The value C(A,B) =1 means that all solutions in B are weakly dominated by A while C(A,B) = 0 represents the situation when none of the solutions in B are weakly dominated by A. Note that always both directions have to be considered, since C(A,B) is not necessarily equal to 1 – C(B,A). In the case that $0 < C(A,B) < 1$ and $0 < C(A,B) < 1$, then we say that neither A weakly dominates B nor B weakly dominates A. We can say that sets A and B are incomparable or that A is not worse than B.

The spacing metric [14] aims at assessing the spread (distribution) of vectors throughout the set of nondominated solutions. This metric is defined as

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

where $d_i = \min(|f_1^i(\vec{x}) - f_1^j(\vec{x})| + |f_1^i(\vec{x}) - f_1^j(\vec{x})|)$, i ,j = 1, …, n, $\bar{d}$ is the mean of all d_i , and n is the number of nondominated vectors found. The desired value for this metric is zero which means that the elements of the set of nondominated solutions are equidistantly spaced.

The proposed algorithm was compared to MOPSO. In the experiments conducted, both MOPSO and MOPSO-CD used 100 particles, a repository size of 100 particles and a mutation rate of 0.5. MOPSO used 30 divisions for its adaptive grid while the proposed algorithm selects the global best from the top 10%[1] sorted repository and replaces one of the nondominated solutions in the bottom 10% of the repository. To restrict the influence of random effects, the experiments were repeated thirty times for each test function. Each experiment uses a different randomly generated initial population.

3.3 Results and Discussion

The following are the results of the 30 independent runs of both algorithms. In Figures 2 to 7, the graphical results show the median result with respect to the coverage metric value produced by each algorithm. It can be seen that both MOPSO-CD and MOPSO were able to converge to the true Pareto front and were able to cover the entire Pareto fronts of the three test functions.

Tables I to III show the numerical results produced by both algorithms in terms of the two performance metrics considered and their computational time.

The results for the coverage metric show that C(MOPSO-CD,MOPSO) < 1 and C(MOPSO,MOPSO-CD) < 1 which mean that the average performance of the two algorithms are incomparable.

In terms of solution diversity, MOPSO-CD has a better distribution of the generated nondominated solutions than MOPSO in all three test functions. The performance of the proposed algorithm is almost twice better than that of MOPSO.

[1] This percentage value was determined after performing experiments on the proposed algorithm.

This shows that using crowding distance not only provides a well-distributed set of nondominated solutions, it also helps in the convergence of the algorithm to the true Pareto front.

As expected, MOPSO is significantly faster computational time than MOPSO-CD. In the first and second test functions, MOPSO is almost twice faster than MOPSO-CD while MOPSO is seven times faster than MOPSO-CD in the third test function. This is attributed to the adaptive grid used by MOPSO which can be computed faster than crowding distance where the relative distances of each solution in the archive is computed whenever the global best solution is selected and when selecting a solution to be replaced by new solutions. Despite this, the proposed algorithm is still competitive because aside from being able to converge to the Pareto front, it also produced a well distributed set of the nondominated solutions.

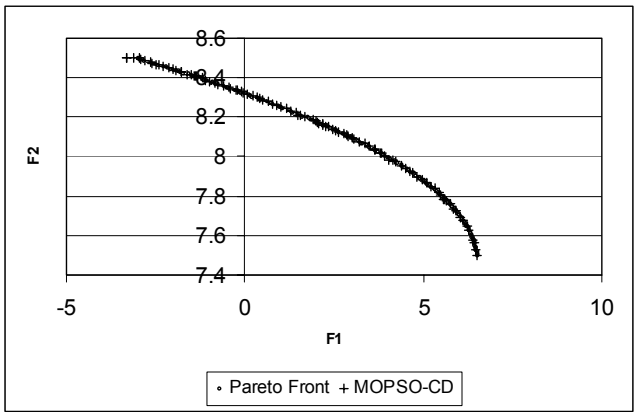

Figure 2. Pareto front produced by MOPSO-CD for the first test function.

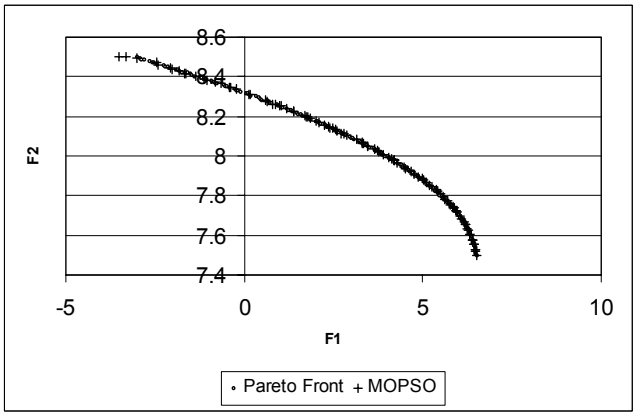

Figure 3. Pareto front produced by MOPSO for the first test function.

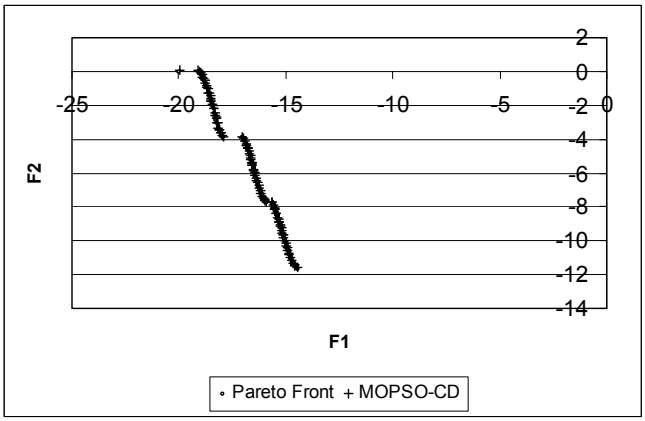

Figure 4. Pareto front produced by MOPSO-CD for the second test function.

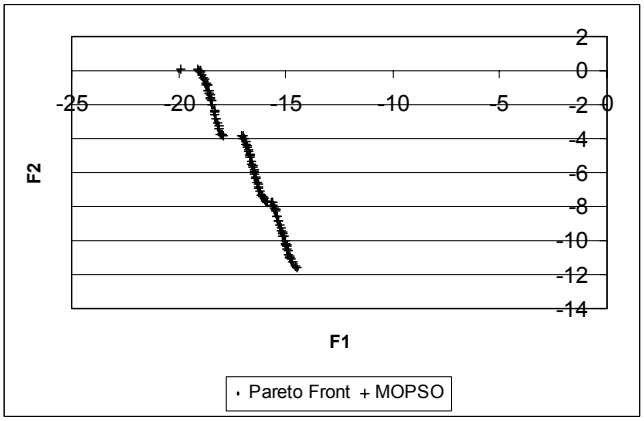

Figure 5. Pareto front produced by MOPSO for the second test function.

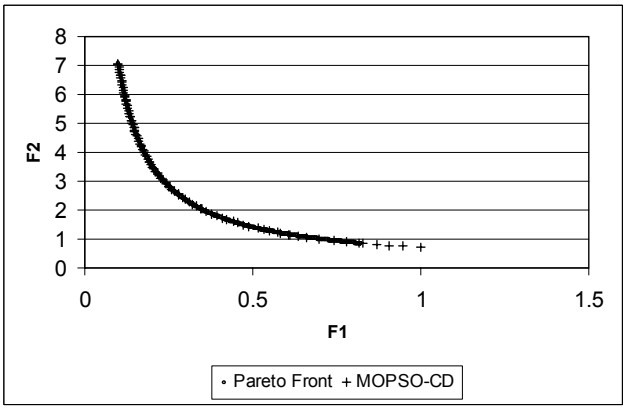

Figure 6. Pareto front produced by MOPSO-CD for the third test function.

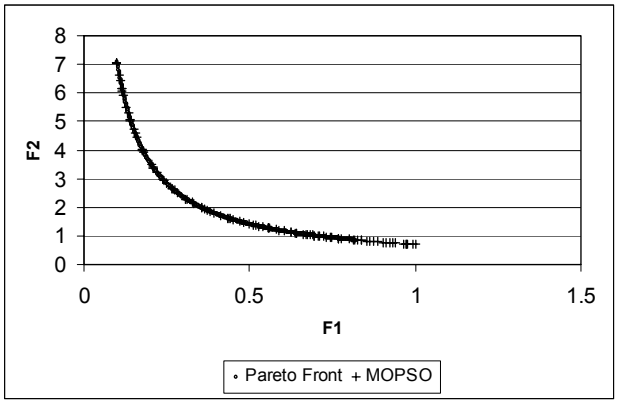

Figure 7. Pareto front produced by MOPSO for the third test function.

Table I. Results of the coverage (C) metric for the three test functions where A is the result of MOPSO-CD and B is the result of MOPSO.

Test Function	C	C(A,B)	C(B,A)
1	Average	0.114333333	0.191
	Median	0.105	0.19
	Std. Dev.	0.040486978	0.043735096
2	Average	0.195333333	0.179333333
	Median	0.2	0.18
	Std. Dev.	0.043607444	0.045480828
3	Average	0.036	0.033333333
	Median	0.04	0.03
	Std. Dev.	0.012484473	0.010933445

Table II. Results of the spacing (SP) metric for the three test functions.

Test Function	SP	MOPSO-CD	MOPSO
1	Average	**0.060737708**	0.117535722
	Median	0.038596099	0.064703038
	Std. Dev.	0.070809635	0.170367787
2	Average	**0.090331351**	0.0917268
	Median	0.094115425	0.099084673
	Std. Dev.	0.013925502	0.021873167
3	Average	**0.025085152**	0.049055135
	Median	0.025050008	0.048681657
	Std. Dev.	0.001324037	0.007868645

Table III. Results of the computational time (in seconds) for the three test functions.

Test Function	Time	MOPSO-CD	MOPSO
1	Average	0.5481	**0.3099**
	Median	0.5545	0.312
	Std. Dev.	0.030488381	0.0202184
2	Average	0.7787	**0.4776**
	Median	0.7735	0.469
	Std. Dev.	0.045782959	0.0243998
3	Average	8.331733333	**1.1239**
	Median	8.3595	1.125
	Std. Dev.	0.165887983	0.0204034

4. ALGORITHM PARAMETERS

Experiments were performed to analyze the effect of mutation in the algorithm as well as the effect or replacing nondominated solutions whenever the archive is full.

4.1 Impact of the Mutation Operator

The impact of using the mutation operator on the proposed algorithm is analyzed by comparing the result of the running MOPSO-CD with and without the mutation operator using the same set of parameters used in Section 3. The results are shown in Tables IV and V. The result show that running MOPSO-CD with and without mutation are incomparable. In terms of diversity, there is a significant improvement in the first test function, a constrained optimization problem, with MOPSO-CD with mutation. The rest have only marginal differences in their performance. This is also true with regard to the computational time. The use of mutation operator is recommended.

Table IV. Results of the coverage (C) metric for the three test functions in determining the impact of the mutation operator on the proposed algorithm where A is MOPSO-CD with mutation and B is MOPSO-CD without mutation.

Test Function	C	C(A,B)	C(B,A)
1	Average	0.186333333	0.175333333
	Median	0.19	0.17
	Std. Dev.	0.04429395	0.043844029
2	Average	0.178333333	0.190666667
	Median	0.17	0.19
	Std. Dev.	0.042024897	0.041848111
3	Average	0.452666667	0.054
	Median	0.05	0.03
	Std. Dev.	0.486896812	0.179954017

Table V. Results of the spacing (SP) metric for the three test functions in determining the impact of the mutation operator on the proposed algorithm.

Test Function	SP	MOPSO-CD With Mutation	MOPSO-CD No Mutation
1	Average	**0.060737708**	0.126501245
	Median	0.038596099	0.046363484
	Std. Dev.	0.070809635	0.221058691
2	Average	**0.090331351**	0.09098054
	Median	0.094115425	0.094548829
	Std. Dev.	0.013925502	0.01348027
3	Average	**0.025085152**	0.032361478
	Median	0.025050008	0.027762782
	Std. Dev.	0.001324037	0.008215267

Table VI. Results of the computational time (in seconds) for the three test functions in running MOPSO-CD with and without mutation.

Test Function	Time	MOPSO-CD With Mutation	MOPSO-CD No Mutation
1	Average	**0.5481**	0.6172333
	Median	0.5545	0.625
	Std. Dev.	0.030488381	0.082852
2	Average	**0.7787**	0.9389667
	Median	0.7735	0.9375
	Std. Dev.	0.045782959	0.055951
3	Average	**8.331733333**	12.5938
	Median	8.3595	12.516
	Std. Dev.	0.165887983	1.1052102

4.2 Impact of Replacement of Nondominated Solutions in the Archive

Tables VII to IX show the results of the 30 independent runs of MOPSO-CD with and without replacing a nondominated solution with a new nondominated solution whenever the archive is full. With regard to the coverage metric, the results show that both MOPSO-CD with and without replacement are incomparable. There is a significant improvement in the diversity of the first test function while second and third test function have only marginal improvement when running MOPSO-CD with replacement. There is a significant decrease in computational time when running MOPSO-CD without replacement. This is attributed to the crowding distance computation of the algorithm when replacing nondominated solutions in the archive.

Table VII. Results of the generational distance (GD) metric for the three test functions in determining the impact of the replacement of nondminated solutions in the archive where A is the MOPSO-CD with replacement on and B is MOPSO-CD without replacement.

Test Function	C	C(A,B)	C(B,A)
1	Average	0.076	0.226666667
	Median	0.08	0.225
	Std. Dev.	0.029077779	0.044515269
2	Average	0.093666667	0.238333333
	Median	0.09	0.245
	Std. Dev.	0.040299169	0.054461839
3	Average	0.029333333	0.033666667
	Median	0.03	0.03
	Std. Dev.	0.014125871	0.015196037

Table VIII. Results of the spacing (SP) metric for the three test functions in determining the impact of the of the replacement of nondominated solutions in the archive.

Test Function	SP	MOPSO-CD With replacement	MOPSO-CD Without Replacement
1	Average	0.060737708	0.100363804
	Median	0.038596099	0.051070457
	Std. Dev.	0.070809635	0.09468505
2	Average	0.090331351	0.10886873
	Median	0.094115425	0.108453288
	Std. Dev.	0.013925502	0.007306683
3	Average	0.025085152	0.035545694
	Median	0.025050008	0.034899382
	Std. Dev.	0.001324037	0.005956281

Table IX. Results of the computational time (in seconds) for the three test functions in running MOPSO-CD with and without replacement of nondominated solutions.

Test Function	Time	MOPSO-CD With replacement	MOPSO-CD Without Replacement
1	Average	**0.5481**	0.6172333
	Median	0.5545	0.625
	Std. Dev.	0.030488381	0.082852
2	Average	**0.7787**	0.9389667
	Median	0.7735	0.9375
	Std. Dev.	0.045782959	0.055951
3	Average	**8.331733333**	12.5938
	Median	8.3595	12.516
	Std. Dev.	0.165887983	1.1052102

5. SUMMARY AND CONCLUSION

This paper has presented an approach called MOPSO-CD that extends the Particle Swarm Optimization (PSO) algorithm to handle multiobjective optimization problems. The mechanism of crowding distance is incorporated into the algorithm of PSO specifically on global best selection and in the deletion method of an external archive of nondominated solutions. The diversity of nondominated solutions in the external archive is maintained by using the mechanism of crowding distance together with a mutation operator. The performance of this approach is evaluated on test functions and metrics from literature. The results show that MOPSO-CD is highly competitive in converging towards the Pareto front and has generated a well-distributed set of nondominated solutions.

6. REFERENCES

[1] Coello, C., Pulido, G., and Salazar, M. Handling multiobjectives with particle swarm optimization. In *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 256-279, June 2004.

[2] Coello, C. and Pulido, G. Multiobjective optimization using a micro-genetic algorithm. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO 2001)*, L. Spector, E. D. Goodman, A.Wu,W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., San Francisco, CA, pp. 274–282, 2001.

[3] Deb, K. Multi-objective genetic algorithms: problem difficulties and construction of test problems. *Evolutionary Computing*, vol. 7, pp. 205–230, Fall 1999.

[4] Deb, K., Agrawal, S., Pratab, A., and Meyarivan, T. A fast elitist nondominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proc. Parallel Problem Solving from Nature VI Conference*, pp. 849–858, 2000.

[5] Fieldsend, J. and Singh, S. A multi-objective algorithm based upon particle swarm optimization, an efficient data structure and turbulence. In *Proc. 2002 U.K. Workshop on Computational Intelligence*, Birmingham, U.K., pp. 37–44, Sept. 2002.

[6] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[7] Kennedy, J. and Eberhart, R.. Particle Swarm Optimization. In *Proceedings of the Fourth IEEE International Conference on Neural Networks*, Perth, Australia, 1995.

[8] Kita, H., Yabumoto, Y., Mori, N., and Nishikawa, Y. Multi-objective optimization by means of the thermodynamical genetic algorithm. In *Parallel Problem Solving From Nature—PPSN IV*, H.-M. Voigt,W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds. Berlin, Germany: Springer-Verlag, Lecture Notes in Computer Science, pp. 504–512, Sept. 1996.

[9] Knowles, J. and Corne, D. Approximating the nondominated front using the Pareto archived evolution strategy. *Evol. Computing*, vol. 8, pp. 149–172, 2000.

[10] Kursawe, F. A variant of evolution strategies for vector optimization. In *Lecture Notes in Computer Science*, H. P. Schwefel and R. Männer, Eds. Berlin, Germany: Springer-Verlag, vol. 496, Proc. Parallel Problem Solving From Nature, 1st Workshop, PPSN I, pp. 193–197, Oct 1991.

[11] Li, X. *et al*. A nondominated sorting particle swarm optimizer for multiobjective optimization. In *Lecture Notes in Computer Science*, vol. 2723, Proc. Genetic and Evolutionary Computation, GECCO 2003, Part I, E. Cantú-Paz *et al.*, Eds. Berlin, Germany, pp. 37–48, July 2003.

[12] Parsopoulos, K. and Vrahatis, M. Particle swarm optimization method in multiobjective problems. In *Proc. 2002 ACM Symp. Applied Computing (SAC'2002)*, Madrid, pages 603–607, Spain, 2002.

[13] Ray, T. and Liew, K. A swarm metaphor for multiobjective design optimization. *Engineering Opt.*, vol. 34, no. 2, pp. 141–153, March 2002.

[14] Schott, J. *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization.* Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1995.

[15] Van Veldhuizen, D. and Lamont, G. Multiobjective evolutionary algorithm research: A history and analysis. Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst. Technol., Wright-Patterson AFB, OH, Tech. Rep. TR-98-03, 1998.

[16] Zitzler, E., Laumanns, M. and Thiele, L. SPEA2: Improving the strength Pareto Evolutionary algorithm. In *Proc. EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control With Applications to Industrial Problems*,K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, Eds., Athens, Greece, Sept. 2000.