Constrained Multi-Objective Optimization Algorithm with Ensemble of Constraint Handling Methods

B. Y. Qu and P. N. Suganthan

School of Electrical and Electronic Engineering

Nanyang Technological University, Singapore 639798

E070088@ntu.du.sg, epnsugan@ntu.edu.sg

Abstract: Different constraint handling techniques have been used with multiobjective evolutionary algorithms (MOEA) to solve constrained multiobjective optimization problems. It is impossible for a single constraint handling technique to outperform all other constraint handling techniques always on every problem irrespective of the exhaustiveness of parameter tuning. To overcome this selection problem, we use an ensemble of constraint handling methods (ECHM) to tackle constrained multiobjective optimization problems. The ECHM is integrated with multi-objective differential evolution (MODE) algorithm. The performance is compared between the ECHM and the same single constraint handling methods using the same MODE. The results show that ECHM overall outperforms the single constraint handling methods.

Index Terms - Multiobjective evolutionary algorithms, constrained multiobjective optimization, ensemble of constraint handling methods, multi-objective differential evolution.

I Introduction

Multiobjective evolutionary algorithms (MOEA) have been successfully applied to solve optimization problems with multiple conflicting objectives in the fields of science and engineering [1], [2]. Although MOEAs were originally designed for solving unconstrained and bound constrained multiobjective optimization problems, most real-world optimization problems

have general constraints too. Generally, constrained multiobjective problems are difficult to solve, as finding a feasible solution may require substantial computational resources [3]. Typically, a constrained multiobjective optimization problem can be defined as follows [4]:

$$\text{Minimize/Maximize } f_m(\boldsymbol{x}) = f_m(x_1, x_2, \ldots, x_n), \qquad m = 1, 2, \ldots, M;$$

$$\text{Subject to } g_j(\boldsymbol{x}) = g_j(x_1, x_2, \ldots, x_n) \geq 0, \quad j = 1, 2, \ldots, J;$$

$$h_k(\boldsymbol{x}) = h_k(x_1, x_2, \ldots, x_n) = 0, \qquad k = 1, 2, \ldots, K;$$

$$x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1, 2, \ldots, n.$$

where $f_m(\boldsymbol{x})$ is the M-dimensional vector objective function, $g_j(\boldsymbol{x})$ and $h_k(\boldsymbol{x})$ are inequality constraint and equality constraint functions, respectively. $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ are n-dimensional decision variables. When dealing with constrained multiobjective optimization problems, individuals that satisfy all of the constraints are called feasible individuals while individuals that do not satisfy at least one of the constraints are called infeasible individuals.

One of the major issues of constrained multiobjective optimization is how to deal with the infeasible individuals throughout the search process commonly known as the constraint handling method (CHM). One way to handle infeasible individuals is to completely disregard them and continue the search process with feasible individuals only. However, as MOEAs are probabilistic search methods, any useful information in the infeasible individuals could be wasted [3]. If the infeasible solutions are completely disregarded, the diversity may be lost during the early stages of the search process leading to a locally optimal or partial Pareto-front. Different constraint handling techniques have been proposed to be used with MOEAs [5]. However, according to the no free lunch theorem [6], no single state-of-the-art constraint handling technique can outperform all others on every problem. Therefore, instead of attempting

to choose one method by extensive trial-and-error procedure, we employ an ensemble of constraint handling method (ECHM) incorporating three different constraint handling methods. The results show that the ECHM outperforms all the single constraint handling techniques when integrated with the multiobjective differential evolution (MODE) algorithm.

The remainder of this paper is structured as follows. Section II provides a brief overview of multiobjective optimization. In Section III, various constraints handling methods are presented. In Section IV, the multiobjective differential evolution algorithm is introduced. Next, in Section V, the proposed ensemble of constraint handling methods is described. Finally, we present the results of the experiments and conclude the paper in Sections VI and VII respectively.

II Multiobjective Optimization

Multiobjective optimization (MOO) occurs frequently in real-world scenarios. There is no single optimal solution for multiobjective optimization problems. Instead, different solutions produce trade-offs among different objectives. An approximation to the Pareto optimality is required for selecting a final preferred solution. Most multiobjective evolutionary algorithms [7], [8], [9], [10], [11] use the concept of dominance in their search. During the search process, two solutions are compared on the basis of whether one dominates the other solution. A solution $\boldsymbol{x}_1$ is said to dominate another solution $\boldsymbol{x}_2$ if the both conditions specified below are satisfied:

1. The solution $\boldsymbol{x}_1$ is no worse than $\boldsymbol{x}_2$ in all objectives.
2. The solution $\boldsymbol{x}_1$ is strictly better than $\boldsymbol{x}_2$ in at least one objective.

If $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ do not dominate each other, then they are known as non-dominated solutions. Generally, the current set of non-dominated solutions is stored in an external archive. The size of the archive will be restricted to a pre-specified size which is normally the same as the finally required approximation solution set size. In every generation, the newly generated offspring are combined with the solutions in the current archive to obtain the mixed-temporary population.

Then the non-domination sorting is applied to this mixed-temporary population and the best solutions are retained to fill the capacity of the external archive in the next generation.

III Constraint Handling Methods

Many different constraint handling techniques have been proposed in the literature since the work of Goldberg and Samtani [41]. Michalewicz and Schoenauer [12] grouped the methods for handling constraints within EAs into four categories: preserving feasibility of solutions [13], penalty functions, make a separation between feasible and infeasible solutions, and hybrid methods.

Based on the number of feasible solutions present in the current solutions, the search process of a constrained problem can be divided into three phases [14] considering the combined parent-offspring population: 1) No feasible solution, 2) At least one feasible solution, and 3) Combined offspring-parent population has more feasible solutions than the size of next generation parent population. Various constraint-handling techniques have been proposed over the years to handle diverse constraints in EAs. The differences between these techniques are how to deal with the infeasible individuals throughout the three search phases.

Although solving single objective constrained optimization problems has been studied for several decades, very few works have been done in solving constrained multiobjective optimization problems. Deb et al. [15] proposed a constrained multiobjective algorithm based on the concept of constrained domination, which is also known as superiority of the feasible solution. Woldesenbet *et al.* [16] introduced a constraint handling technique based on adaptive penalty functions and distance measures by extending the corresponding version for the single objective constrained optimization [17].

In this section, three different constraint handling methods namely self adaptive penalty, superiority of feasible solution, and ε-constraint are extended from single objective constrained optimization to multiobjective optimization. For all three CHM, the overall constraint violation $v(\boldsymbol{x})$ is calculated using the following steps:

Step 1: Transform equality constraints into inequality form, and then we can combine all the constraints as:

$$G_j(X) = \begin{cases} \max\{g_j(X),0\} & j = 1,...J. \\ \max\{|h_j(X)| - \delta,0\} & j = J+1,...,J+K \end{cases}$$

where δ is a tolerance parameter for the equality constraints.

Step 2: Get the constraint violation for $v_j(\boldsymbol{x})$, j=1 to J+K (*number_of_constraints*)

Step 3: Normalize the constraint violation using equation:

$$v_{j\_normalized}(\boldsymbol{x}) = v_j(\boldsymbol{x}) / v_{max}$$

where $\boldsymbol{v}_{max}$ is the current maximum violation of constraint j

Step 4: Calculate the overall constraint violation using equation:

$$v(\boldsymbol{x}) = \sum_{j=1}^{J+K} v_{j_{normalized}}(\boldsymbol{x})$$

Self adaptive penalty (SP) [16]

The most common and the earliest approach in the EA community to handle constraints is to use penalty functions. The idea of penalty functions was first introduced by Courant [18]. This method transforms a constrained optimization problem into an unconstrained problem by adding a penalty factor to the fitness value of each infeasible individual so that it is penalized for violating one or more of the constraints. If the penalties added depend only on the degree of

violation, then the penalty is called static penalty. On the other hand, if the penalty function depends on the current generation count also, the penalty function is called dynamic penalty [19]. In adaptive penalty [20], information gathered from the search process will be used to control the amount of penalty added to infeasible individuals. Penalty-based constraint handling techniques for multiobjective is similar to single objective except that the penalty factor is added to all the objectives instead of only one objective.

A self adaptive penalty function is proposed by Woldesenbet [16] to solve constrained multiobjective optimization problems using evolutionary algorithms. The method keeps track of the number of feasible individuals in the population to determine the amount of penalty added to infeasible individuals. If there are a few feasible individuals in the whole population, a larger penalty factor is added to infeasible solutions. Otherwise, a small penalty factor is used. Self adaptive penalty function uses modified objective function values instead of using the original objective values. The modified objective value has two components: distance measure and adaptive penalty.

Distance values are found for each objective function by incorporating the effect of an individual's constraint violation into its each objective value. The distance value of individual $\boldsymbol{x}$ in each objective function $\boldsymbol{m}$ is estimated as follows:

$$d_m^I(x) = \begin{cases} v(x) & \text{if } r_f = 0 \\ \sqrt{f_m''(x) + v(x)^2}, & \text{otherwise} \end{cases} \qquad (1)$$

where $\boldsymbol{v(x)}$ is the overall constraint violation, $r_f = \frac{\text{Number of feasible individuals}}{\text{population size}}$, $f_m''(x)$ is the normalized objective value for m^{th} objective and defined as $f_m''(x) = \frac{f_m(x) - f_{min}}{f_{max} - f_{min}}$. Here, f_{max} and f_{min} are the current maximum and minimum values of the corresponding objective function.

According to equation (1), if there is no feasible solution in the current population, an infeasible solution with a smaller constraint violation will dominate another infeasible individual with a higher constraint violation. On the other hand, if the number of feasible solution is not zero. The distance value will include both objective values and constraint violations. Beside the distance values, two other penalty functions are also used to include additional penalties for infeasible individuals:

$$p_m(x) = (1 - r_f)X_m(x) + r_f Y_m(x), \qquad (2)$$

where

$$X_m(x) = \begin{cases} 0, & \text{if } r_f = 0 \\ v(x) & \text{otherwise} \end{cases}$$

$$Y_m(x) = \begin{cases} 0, & \text{if } x \text{ is a feasible individual} \\ f_m^{\max}, & \text{if } x \text{ is an infeasible individual} \end{cases}$$

The final modified objective value for the m^{th} objective of individual $\boldsymbol{x}$ is formulated as the sum of the distance and penalty values:

$$F_m(x) = d_m(x) + p_m(x) \qquad (3)$$

This modified objective value formulation is flexible and will allow the search to utilize infeasible solutions. Table I shows how the fitness values are modified by the self-adaptive penalty method.

Table I. Fitness modification by using the self-adaptive penalty method

Step 1:	Find feasible solution(s) from the combined population and calculate the feasible percentage of the current population r_f
Step 2:	If $r_f = 0$
	For m = 1 to M (*number_of_objectives)*
	$d_m(x) = v(x)$

$$X_m(x) = 0$$

End For

Else

For m = 1 to M *(number _of_objectives)*

$$d_m(x) = \sqrt{f_m^m(x) + v(x)^2}$$

$$X_m(x) = v(x)$$

End For

End If

Step 3: If $v(x) = 0$

For m = 1 to M *(number _of_objectives)*

$$Y_m(x) = 0$$

End For

Else

For m = 1 to M *(number_of_objectives)*

$$Y_m(x) = f_m^m$$

End For

End If

Step 4: Apply equation (2) to obtain $p_m(x)$.

Step 5: Form the new fitness values for all the objectives of all solutions using equation (3).

Superiority of feasible solution (SF)

This constraint handling method was first introduced by Powell and Skolnick [21] in 1993 and extended by Deb [22] for single objective optimization. The SF as used in multiobjective optimization is expressed as:

$$fitness_m(x) = \begin{cases} f_m(x) & \text{if } x \text{ is feasible} \\ f^m_{worst} + v(x) & \text{otherwise} \end{cases} \quad (4)$$

where f^m_{worst} is the m^{th} objective value of the worst feasible solution with respect to objective m in the population and $v(\boldsymbol{x})$ is the overall constraint violation. If there is no feasible solution in the current population, f^m_{worst} is zero.

In this approach, there is no penalty factor involved. Feasible solutions are always better than the infeasible ones. In this way, the infeasible solutions are expected to evolve towards the feasible region and feasible solutions are expected to evolve towards the global optimal front. Table II shows how the fitness values are modified by the superiority of feasible solution method.

Table II. Fitness modification by using the SF method

Step 1:	Extract feasible solution(s) from the combined population. If there is no feasible solution, the new fitness value is overall constraint violation value. If all solutions are feasible, the fitness values are unchanged.
Step 2:	For m = 1 to M *(number_of_objectives)* Find the maximum objective values $f^m_{m\_max}$ for each objective among the feasible solutions. End For
Step 3:	Form the new fitness values for infeasible solutions using equation (4) and keep the fitness values of feasible solutions unchanged.

ε-constraint (EC)

The ε-constraint handling technique was first proposed by Takahama [23] in 2005 (←check with Rammohan if this is really the first one. Also search for NFT or "nearest feasibility threshold"). The basic idea is similar to the superiority of feasible solutions and the difference is the relaxation of the constraint violation during the early stages of the search process using the ε parameter. As some useful information may be contained in the infeasible solutions with relatively small overall constraint violation, the relaxation of the constraint violations may include more infeasible solutions in the population during the early stages of evolution. The ε value is gradually reduced until the generation counter k reaches the control generations T_c. After T_c generations, the ε value is set to zero and the EC becomes the same as the superiority of feasible method. The ε value is updated according to the following equations:

$$\varepsilon(0) = v(x_\theta) \tag{5}$$

$$\varepsilon(k) = \begin{cases} \varepsilon(0)\left(1 - \frac{k}{T_c}\right)^{cp}, & 0 < k < T_c \\ 0, & k \geq T_c \end{cases}$$

where x_θ is the top θ-th individual at initialization. *cp* parameter is recommended to be [2,10]. Table III shows how the fitness values are modified by the ε-constraint method.

Table III. Fitness modification by using the EC method

Step 1:	Find infeasible solution(s) from the combined population (if all solutions are feasible, ε is set to be zero throughout the evolution)
Step 2:	If the generation count k=1
	Use equation (5) to obtain $\varepsilon(0)$
	Else if the generation count $k < T_c$

$$\varepsilon(k) = \varepsilon(0)\left(1 - \frac{k}{T_c}\right)^{cp},$$

Else

$$\varepsilon(k) = 0$$

End If

Step 3: Find the solutions with constraint violation less than ε and from these solutions find the maximum objective values $f_{m\_max}^{''}$ for each objective.

Step 4: Form the new fitness values for the solutions that have constraint violations greater than ε using the following equation while others remain unchanged:

$$fitness(x) = f_{m\_max}^{''} + v(x)$$

Comparative Discussion on Constraint Handling Methods

When solving constrained problems, one of the following three scenarios can be encountered: (a) no feasible solution, (b) at least one feasible solution, and (c) combined offspring-parent population has more feasible solutions than the size of next generation parent population. When an ensemble of constraint handling method is constructed, it is beneficial to choose competitive CHMs which also perform differently during these three scenarios.

The SF method ranks feasible solutions better than the infeasible ones. Two infeasible solutions are compared based on their overall constraint violations only, while two feasible solutions are compared based on their objective function values only. Therefore, in scenario (a), infeasible solutions with low overall constraint violation are selected from the combined parent and offspring population. In scenario (b), first all feasible ones are selected and then infeasible ones with the lowest overall constraint violations are selected. In scenario (c), only feasible ones with best objective values are selected. The SP method always selects individuals based on a value

determined by the overall constraint violation and objective values. Thus, an individual with lower overall constraint violation and higher fitness can be selected over a feasible individual with lower fitness in scenario (c) too. The ε-constraint (EC) method selects similar to the SF, but in the EC, a solution is regarded as feasible if its overall constraint violation is smaller than $\varepsilon(k)$. Therefore, it is obvious that these competitive methods rank solutions differently.

IV Multiobjective Differential Evolution

Differential evolution is a population-based optimization algorithm that was first introduced by Price and Storn [24] in 1995 for solving single-objective optimization problems. Differential evolution algorithm has also been used to solve multiobjective optimization problems [25], [26], [27], [28], [29], [30], [31], [36], [37], [40] frequently with an external archive to hold the current set of non-dominated solutions. The overall structure of multiobjective differential evolution (MODE) is similar to other multiobjective evolutionary algorithms. MODE starts with a randomly initialized population of size *NP* that is evolved by applying the DE operators such as mutation, recombination and selection to every solution member $\boldsymbol{x}$ until a stopping criterion is satisfied. For each parent x_p at each generation, an associated mutant vector v_p is generated using the following strategy [25] :

DE/rand/2: $v_p = x_{r1} + F \cdot (x_{r2} - x_{r3}) + F \cdot (x_{r4} - x_{r5})$ (6)

where *r1*, *r2*, *r3*, *r4*, *r5* are random and mutually different integers generated in the range [1, *NP* (population size)], which should also be different from the current trial vector. *F* is a scale factor in [0, 2] used to scale the differential vectors.

The “binary” (or uniform) crossover operation is applied to each pair of the generated mutant vector and its corresponding trial vector. The process can be represented as:

$$u_{p,i} = \begin{cases} v_{p,i} & \text{if } rand_i \leq CR \\ x_{p,i} & \text{otherwise} \end{cases} \qquad (7)$$

where u_p is the trial vector. Crossover rate CR determines which dimensions of u_p are copied from v_p or x_p respectively [28] where $rand_i$ is in the range [0, 1].

A new population is formed by combining the current solutions and newly generated solutions. This new population will go through a selection process to choose parents for the next generation. In multiobjective optimization, the process is different from single objective, as in single objective case the comparison between parent and offspring is based on an objective function value, and the vector yielding the smaller result is included in the next generation (minimization problems). In the presence of multiple objectives, the selection process is through non-domination sorting [8]. After the process of non-domination sorting, the population members will be ranked according to the front number. The top solutions will be selected to fill the external archive. The top solutions in the external archive are chosen as the parents for next generation. Constrained multiobjective differential evolution algorithm integrates different constraint handling techniques with the MODE. Table IV shows the CMODE algorithm.

Table IV. Constrained Multiobjective Differential Evolution (CMODE) Algorithm

Step 1 : Random Initialization

Randomly initialize the solutions: for p = 1 to NP,

$parent(p,1:n) = x^{(L)} + (x^{(U)} - x^{(L)}) .* rand(1,n)$

Step 2: Evaluate the fitness values and the constraint violations of the initial solutions.

Initialize the external archive and generation counter.

Step 3: For p = 1 to NP

i) Randomly select 3 mutually different population members from the population also different from the target vector

ii) Generate the mutated vector using equation (6)

iii) The "binary" crossover operation is applied using equation (7)

iv) Maintain new offspring in the search space. (if any dimension exceed the bound, it will be set equal to the bound)

v) Evaluate the fitness values and the constraint violations

End For

Step 4: Combine the parents and newly generated offspring to form a new population

For $p = 1$ to $2NP$

Calculate the overall constraint violations.

End For

Step 5:

Apply the selected constraint handling method to modify the fitness values of infeasible solutions. Details are shown in Tables I-III

Step 6: Sort the population using non-domination sorting according to the new fitness values.

Select the top *NP* solutions as the next generation's parents and update the external archive by selecting the top solutions from the current population to fill the capacity of the archive.

Step 7: Stop if a termination criterion is satisfied. Otherwise, increase the generation count and go to Step 3.

V MODE with Ensemble of constraint handling methods

According to the no free lunch (NFL) theorem [6], it is impossible for a single constraint handling technique to outperform all other constraint handling methods on every problem. Each problem has its own characteristics such as the ratio between feasible search space and the whole search space, linearity/non-linearity of constraint functions and so on. Each constraint

handling method may also work differently during different search phases. Hence, different constraint handling methods may be suitable for solving a problem during different stages of the search process.

The flowchart of the ensemble algorithm is shown in Fig. 1. In our implementation, an ensemble of three constraint handling methods is used. Each constraint handling method has a population associated with it. Each population generates its offspring population. For every offspring in the three offspring populations, objective / constraint function values and overall constraint violation are computed. The three offspring populations are combined and passed to the three fitness modification algorithms in Tables I to III simultaneously to undergo different fitness modification processes according to the three CHMs. After the completion of three fitness modification processes, each offspring population is combined with the respective parent population. Therefore, we perform the objective and constraint function evaluation only once per offspring and perform fitness modification three times according to the CHMs.

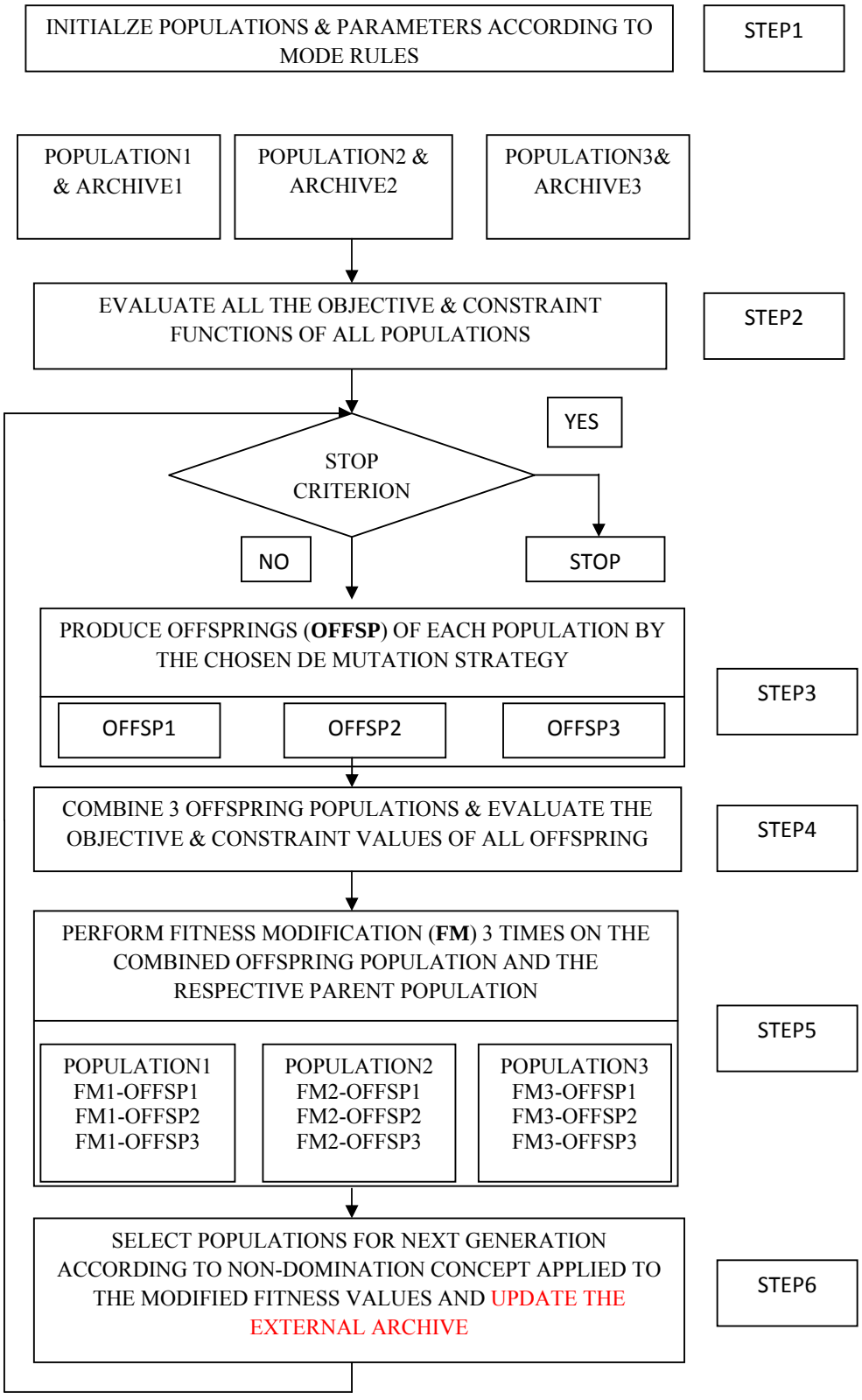

Fig. 1 Flowchart of ensemble of three constraint handling methods

According to the no free lunch (NFL) theorem [6], when we employ single constraint handling methods to solve a particular test problem, we may observe that a particular CHM generates better quality offspring than the other CHMs. Generating better quality solutions implies that these solutions have lower overall constraint violation and/or better fitness values. Such solutions can be detected by all CHMs. Therefore, even if one particular CHM may not be the best choice for generating good quality offspring, this CHM can definitely identify good quality offspring generated by another CHM. Therefore, if a particular constraint handling method is best suited for the search method and the problem during a point in the search process, the offspring population produced by the particular constraint handling technique will dominate the offspring generated by other populations and enter populations corresponding to other constraint handling methods too. In the subsequent generations, these superior offspring will become parents in other populations too. For difficult problems, more constraint handling methods can be included in the ensemble in order to benefit more from each function call.

As the ECHM has three populations, the non-domination sorting operation has to be performed three times whereas the MODE with a single CHM requires only one non-domination sorting operation. In order to make the proposed ECHMs practically beneficial, we can employ the fast sorting method [38] which has a complexity of $O(MN)$, where M is the number of objectives and N is the number of solutions. The commonly used non-domination sorting algorithm has a complexity of $O(MN^2)$ [39].

VI Experimental results

For the experiment, Matlab 7.1 is used as the programming language and the computer is Intel Pentium® 4 CPU 3.00 GHZ, 2 GB of memory. The population size is set at 50 for ensemble

method, 50 and 150 for single constraint handling methods. The maximum external archive size is set at 100. The maximum number of function evaluations (FES) is set as 2e+5. The parameters used in the algorithms are list as below:

DE: $F = 0.3, CR = 0.3$

EC: $cp = 5, T_c = 60000, \theta = 20$

Seven different methods have been compared experimentally. They are:

1. Method A: MODE with an ensemble of three constraint handling methods;
2. Method B: MODE with superiority of feasible solution constraint handling method (population size=50);
3. Method C: MODE with ε-constraint handling method (population size=50);
4. Method D: MODE with self adaptive penalty constraint handling method (population size=50);
5. Method E: MODE with superiority of feasible solution constraint handling method (population size=150);
6. Method F: MODE with ε-constraint handling method (population size=150);
7. Method G: MODE with self adaptive penalty constraint handling method (population size=150);

To test the performances of these algorithms, ten constrained multiobjective benchmark problems available from the literature are used. These problems are presented in Appendix A.

A. Performance measures

In order to compare the performances of different constraint handling methods quantitatively, some performance metrics are needed. There are two goals in a multiobjective optimization: 1) convergence to the Pareto-optimal set and 2) diversity of solutions in the Pareto-optimal set. The

following two indicators are used:

1) R indicator (I_{R2}) [32]: $I_{R2} = \frac{\sum_{\lambda \in \Lambda} u^*(\lambda, A) - u^*(\lambda, R)}{|\Lambda|}$ where R is a reference set, u^* is the maximum value reached by the utility function u with weight vector λ on an approximation set A, *i.e.*, $u^* = \max_{z \in A} u_\lambda(z)$. We choose the augmented Tchebycheff function as the utility function.

2) Hypervolume difference to a reference set ($I_{\bar{H}}$) [32]: The hypervolume indicator I_H measures the hypervolume of the objective space that is weakly dominated by an approximation set A, and is to be maximized. Here we consider the hypervolume difference to a reference set R, and we will refer to this indicator as $I_{\bar{H}}$, which is defined as $I_{\bar{H}} = I_H(R) - I_H(A)$ where smaller values correspond to higher quality as opposed to the original hypervolume I_H.

B. Comparison of ECHM with single constraint handling methods

From the results in Tables V to VIII, it can be observed that the ensemble of constraint handling methods perform either similar to or better than the single constraint handling methods. From the summation of the rankings shown in the last row, it can be observed that ensemble constraint handling method outperforms all the single constraint handling methods with respect to both the *R*-indicator and *H*-indicator. Further, as each single constraint handling method has similar summation of the rankings, it also suggests that different methods perform differently on each problem. Moreover, as the population sizes of the single constraint handling methods vary, their performance on different problems also varies. It suggests that different population sizes are suitable for different test problems with different constraint handling methods.

The superior performance of ensemble over the single constraint handling methods in the given number of function evaluations is due to the efficient use of every function call by all three populations and the search process in the ensemble benefiting from the best performance of different constraint handling methods during different stages of the search process. In every generation only the best offspring from the three populations will survive and have the opportunity to reproduce, which means the offspring not only compete with their parents but also compete with the offspring produced by the populations of other constraint handling methods.

In order to demonstrate the advantage of the ECHMs more clearly, *t*-test is applied to the indicators and the *h* values are shown in Tables V to VIII below the function names in the first column. All the three single constraint handling methods are compared with the proposed ECHMs. The numerical values -1, 0, 1 represent that the proposed ECHMs is statistically superior to, equal to and inferior to the single constraint handling method. From the result, we can observe that the proposed ECHMs performs either better or similar to all three single constraint handling methods.

Table V: The *R*-indicators for 30 runs and *h*-values on the test problems. *h*-values are shown below the function names in the first column. (Population size=50)

Function		Ensemble	rank	SF	rank	EC	rank	SP	rank
TNK	Mean	2.25E-04	2	2.60E-04	4	2.29E-04	3	2.16E-04	1
	Worst	3.88E-04	2	3.67E-04	1	4.82E-04	4	4.44E-04	3
[0,0,0]	Best	5.45E-05	1	7.18E-04	3	6.31E-05	2	8.97E-05	4
	Std	8.66E-05		9.37E-05		1.46E-04		1.03E-04	
SRN	Mean	8.21E-05	1	3.53E-04	3	3.92E-04	4	2.86E-04	2
	Worst	3.06E-04	1	9.77E-04	4	8.76E-04	3	6.66E-04	2
[-1,-1,-1]	Best	2.75E-06	1	2.78E-05	2	8.82E-05	4	5.45E-05	3
	std	8.04E-05		2.79E-04		2.29E-04		1.80E-04	
CONSTR	Mean	1.56E-05	2	1.94E-05	4	1.55E-05	1	1.66E-05	3
	Worst	2.44E-05	1	3.74E-05	4	3.28E-05	2	3.38E-05	3
[0,0,0]	Best	7.46E-07	1	3.50E-06	3	1.77E-06	2	5.83E-06	4
	std	6.44E-06		1.09E-05		8.42E-06		7.46E-06	
OSY	Mean	7.00E-03	1	1.03E-02	3	3.49E-02	4	7.10E-03	2
	Worst	2.02E-02	1	3.92E-02	3	3.94E-02	4	3.89E-02	2
[-1,-1,0]	Best	1.95E-06	1	3.66E-05	2	1.95E-02	4	3.76E-05	3
	std	9.50E-03		1.16E-02		7.70E-03		1.13E-02	
CTP1	Mean	1.19E-06	1	2.30E-03	4	1.90E-03	3	8.93E-04	2
	Worst	4.46E-06	1	8.00E-03	2	8.50E-03	3	8.50E-03	3
[-1,-1,-1]	Best	3.45E-09	1	3.71E-07	4	2.41E-08	2	4.81E-08	3
	std	1.23E-06		3.10E-03		3.20E-03		2.10E-03	
CTP2	Mean	5.27E-06	1	7.91E-04	4	4.01E-04	3	3.80E-05	2
	Worst	1.36E-05	1	1.74E-02	4	5.50E-03	3	7.14E-05	2
[-1,-1,-1]	Best	3.60E-07	1	3.60E-06	3	3.08E-06	2	9.11E-06	4
	std	4.21E-06		3.30E-03		1.40E-03		2.24E-05	
CTP3	Mean	1.04E-04	1	5.91E-04	4	1.53E-04	2	1.61E-04	3
	Worst	1.87E-04	1	8.50E-04	4	2.71E-04	2	3.39E-04	3
[-1,-1,-1]	Best	4.88E-06	2	5.21E-05	4	1.81E-07	1	4.26E-06	3
	std	5.30E-05		1.80E-03		7.33E-05		7.52E-05	
CTP4	Mean	7.49E-04	1	1.00E-03	3	1.60E-03	4	8.28E-04	2
	Worst	2.00E-03	1	3.90e-03	3	1.22E-02	4	2.10E-03	2
[-1,-1,0]	Best	7.34E-05	1	4.37E-04	4	1.10E-04	2	2.86E-04	3
	std	4.00E-04		8.02E-04		3.10E-03		5.47E-04	
CTP5	Mean	1.00E-03	1	2.90E-03	2	3.90E-03	4	2.90E-03	2
	Worst	1.50E-03	1	1.37E-02	2	1.37E-02	2	1.48E-02	4
[-1,-1,-1]	Best	7.60E-05	2	9.32E-04	4	7.70E-05	2	2.88E-05	1
	std	4.20E-04		3.10E-03		3.30E-03		4.70E-03	
CTP6	Mean	5.87E-08	1	2.80E-03	3	2.80E-03	3	2.50E-03	2
	Worst	2.70E-07	1	1.18E-02	4	6.40E-03	2	9.10E-03	3
[-1,-1,-1]	Best	7.59E-10	1	1.30E-09	2	1.50E-03	4	1.90E-09	3
	std	6.96E-08		2.90E-03		1.60E-03		3.10E-03	
Summation of Ranks:			35		92		85		79

Table VI: The *H*-indicators for 30 runs and *h*-values on the test problems. *h*-values are shown below the function name in the first column. (Population size=50)

Function		Ensemble	rank	SF	rank	EC	rank	SP	rank
TNK	Mean	6.20E-04	2	7.15E-04	4	6.34E-04	3	5.97E-04	1
	Worst	1.20E-03	2	1.00E-03	1	1.30E-03	4	1.20E-03	3
[0,0,0]	Best	1.71E-04	1	2.17E-04	3	1.91E-04	2	2.64E-04	4
	std	2.28E-04		2.47E-04		3.84E-04		2.71E-04	
SRN	Mean	1.60E-03	1	3.80E-03	4	3.70E-03	2	3.70E-03	2
	Worst	1.90E-03	1	4.20E-03	3	4.40E-03	4	4.00E-03	2
[-1,-1,-1]	Best	1.40E-03	1	3.50E-03	3	3.50E-03	3	3.40E-03	2
	std	1.11E-04		1.96E-04		2.41E-04		1.64E-04	
CONSTR	Mean	2.44E-04	2	2.43E-04	1	2.44E-04	2	2.45E-04	4
	Worst	2.60E-04	1	2.69E-04	4	2.66E-04	2	2.67E-04	3
[0,0,0]	Best	1.46E-04	1	2.19E-04	2	2.22E-04	3	2.23E-04	4
	std	3.51E-05		1.53E-05		1.44E-05		1.147-05	
OSY	Mean	1.10E-02	1	1.89E-02	3	7.23E-02	4	1.35E-02	2
	Worst	2.70E-02	1	8.97E-02	4	8.89E-02	3	7.65E-02	2
[0,-1,0]	Best	1.50E-03	1	2.00E-03	2	2.60E-02	4	2.10E-03	3
	std	1.93E-02		2.13E-02		2.43E-02		1.83E-02	
CTP1	Mean	2.34E-05	1	7.90E-03	4	6.40E-03	3	3.40E-03	2
	Worst	3.25E-05	1	2.56E-02	2	2.68E-02	3	2.68E-02	3
[-1,-1,-1]	Best	1.98E-05	1	2.51E-05	4	2.24E-05	2	2.39E-05	3
	std	3.54E-06		9.10E-03		9.80E-03		6.60E-03	
CTP2	Mean	1.14E-05	1	2.40E-03	4	1.30E-03	3	1.21E-04	2
	Worst	2.51E-05	1	5.22E-02	4	1.66E-02	3	5.11E-04	2
[-1,-1,-1]	Best	3.55E-06	2	1.43E-05	1	1.80E-05	3	1.97E-05	4
	std	6.29E-06		9.90E-03		4.30E-02		1.45E-04	
CTP3	Mean	1.80E-04	1	9.59E-04	4	2.37E-04	2	2.53E-04	3
	Worst	3.06E-04	1	1.39E-02	4	4.11E-04	2	5.80E-04	3
[-1,-1,-1]	Best	2.47E-05	2	8.74E-05	4	3.54E-07	1	2.90E-05	3
	std	7.82E-05		2.90E-03		1.10E-04		1.18E-04	
CTP4	Mean	2.20E-03	1	3.30E-03	3	4.80E-03	4	2.50E-03	2
	Worst	6.10E-03	1	9.60E-02	4	3.59E-02	3	6.20E-03	2
[-1,-1,0]	Best	3.84E-04	1	1.30E-03	4	4.28E-04	2	9.31E-04	3
	std	1.10E-03		2.30E-03		9.00E-03		1.50E-03	
CTP5	Mean	1.60E-03	1	4.50E-03	2	6.00E-03	4	4.70E-03	3
	Worst	2.20E-03	1	2.13E-02	2	2.13E-02	2	2.32E-02	4
[-1,-1,-1]	Best	1.48E-04	2	1.50E-03	4	1.83E-04	3	1.58E-04	1
	std	6.21E-04		5.40E-03		6.50E-03		7.40E-03	
CTP6	Mean	9.41E-07	1	4.40E-03	3	4.41E-03	4	3.90E-03	2
	Worst	1.26E-06	1	1.81E-02	4	1.02E-02	2	1.45E-02	3
[-1,-1,-1]	Best	7.91E-07	1	1.02E-06	3	2.20E-03	4	9.89E-07	2
	std	1.14E-07		4.60E-03		2.60E-03		4.90E-03	
Summation of Ranks:			36		94		86		79

Table VII: The *R*-indicators for 30 runs and *h*-values on the test problems. *h*-values are shown below the function names in the first column. (Population size=150)

Function		Ensemble	rank	SF	rank	EC	rank	SP	rank
TNK	Mean	2.25E-04	1	3.44E-04	4	3.42E-04	3	2.49E-04	2
	Worst	3.88E-04	2	7.09E-04	3	7.75E-04	4	3.87E-04	1
[-1,-1,0]	Best	5.45E-05	1	8.21E-05	3	7.47E-05	2	1.52E-04	4
	Std	8.66E-05		1.91E-04		1.73E-04		9.29E-05	
SRN	Mean	8.21E-05	1	6.05E-04	3	1.40E-03	4	4.72E-04	2
	Worst	3.06E-04	1	1.10E-03	2	5.60E-03	4	1.80E-03	3
[-1,-1,-1]	Best	2.75E-06	1	2.10E-05	3	2.80E-05	4	1.57E-05	2
	std	8.04E-05		3.75E-04		1.30E-03		4.69E-04	
CONSTR	Mean	1.56E-05	3	1.55E-05	2	1.83E-05	4	1.08E-05	1
	Worst	2.44E-05	1	2.56E-05	2	5.23E-05	4	3.38E-05	3
[0,0,0]	Best	7.46E-07	1	1.97E-06	4	8.55E-07	2	1.87E-06	3
	std	6.44E-06		7.52E-06		1.12E-05		8.08E-06	
OSY	Mean	7.00E-03	1	1.68E-02	3	3.17E-02	4	1.55E-02	2
	Worst	2.02E-02	1	3.89E-02	3	3.92E-02	4	3.88E-02	2
[-1,-1,-1]	Best	1.95E-06	1	3.76E-05	3	1.94E-02	4	1.34E-05	2
	std	9.50E-03		1.48E-02		9.40E-03		1.14E-02	
CTP1	Mean	1.19E-06	1	7.78E-05	3	1.56E-04	4	5.13E-05	2
	Worst	4.46E-06	1	3.77E-04	2	4.20E-03	4	3.77E-04	2
[-1,-1,-1]	Best	3.45E-09	1	4.26E-08	4	2.41E-08	3	5.23E-09	2
	std	1.23E-06		1.52E-04		7.76E-04		1.30E-04	
CTP2	Mean	5.27E-06	2	6.96E-06	3	7.57E-06	4	4.92E-06	1
	Worst	1.36E-05	1	2.01E-05	3	2.61E-05	4	1.84E-05	2
[0,-1,0]	Best	3.60E-07	1	6.34E-07	3	1.24E-06	4	5.00E-07	2
	std	4.21E-06		5.66E-06		6.29E-06		4.29E-06	
CTP3	Mean	1.04E-04	1	1.59E-04	4	1.41E-04	2	1.49E-04	3
	Worst	1.87E-04	1	2.83E-04	3	2.50E-04	2	2.86E-04	4
[-1,-1,-1]	Best	4.88E-06	1	3.22E-05	3	4.63E-05	4	2.56E-05	2
	std	5.30E-05		6.18E-05		6.14E-05		5.69E-05	
CTP4	Mean	7.49E-04	2	6.24E-04	1	1.06E-03	4	7.56E-04	3
	Worst	2.00E-03	2	1.40E-03	1	2.50E-03	4	2.10E-03	3
[0,-1,0]	Best	7.34E-05	1	2.92E-04	3	2.28E-04	2	3.10E-04	4
	std	4.00E-04		2.02E-04		6.08E-04		4.58E-04	
CTP5	Mean	1.00E-03	1	1.20E-03	2	1.20E-03	2	1.40E-03	4
	Worst	1.50E-03	1	2.10E-03	2	2.80E-03	3	5.50E-03	4
[0,0,-1]	Best	7.60E-05	2	7.68E-05	3	6.82E-05	1	7.69E-05	4
	std	4.20E-04		4.51E-04		6.36E-04		1.10E-03	
CTP6	Mean	5.87E-08	1	5.36E-04	2	2.90E-03	4	1.10E-03	3
	Worst	2.70E-07	1	7.80E-03	3	3.00E-03	2	7.80E-03	3
[-1,-1,-1]	Best	7.59E-10	1	3.42E-09	3	3.04E-09	2	7.97E-09	4
	std	6.96E-08		2.00E-03		5.49E-04		2.70E-03	
Summation of Ranks:			37		83		98		79

Table VIII: The *H*-indicators for 30 runs and *h*-values on the test problems. *h*-values are shown below the function name in the first column. (Population size=150)

Function		Ensemble	rank	SF	rank	EC	rank	SP	rank
TNK	Mean	6.20E-04	1	9.27E-04	4	9.21E-04	3	6.75E-04	2
	Worst	1.20E-03	1	1.90E-03	3	2.10E-03	4	1.20E-03	1
[-1,-1,0]	Best	1.71E-04	1	2.35E-04	3	2.16E-04	2	4.18E-04	4
	std	2.28E-04		5.02E-04		4.56E-04		2.44E-04	
SRN	Mean	1.60E-03	1	3.60E-03	3	3.50E-03	2	3.70E-03	4
	Worst	1.90E-03	1	4.60E-03	4	4.10E-03	2	4.10E-03	2
[-1,-1,-1]	Best	1.40E-03	1	3.50E-03	2	3.60E-03	4	3.50E-03	2
	std	1.11E-04		2.02E-04		1.20E-04		1.45E-04	
CONSTR	Mean	2.44E-04	2	2.44E-04	2	2.49E-04	4	2.42E-04	1
	Worst	2.60E-04	1	2.76E-04	4	2.68E-04	3	2.66E-04	2
[0,0,0]	Best	1.46E-04	1	2.18E-04	3	2.11E-04	2	2.21E-04	4
	std	3.51E-05		1.49E-05		1.13E-05		1.35-05	
OSY	Mean	1.10E-02	1	2.75E-02	3	5.33E-02	4	2.27E-02	2
	Worst	2.70E-02	1	7.29E-02	3	8.64E-02	4	6.33E-02	2
[-1,-1,-1]	Best	1.50E-03	1	2.10E-03	3	2.43E-02	4	2.03E-03	2
	std	1.93E-02		2.48E-02		2.15E-02		1.73E-02	
CTP1	Mean	2.34E-05	1	1.97E-04	3	2.79E-04	4	1.31E-04	2
	Worst	3.25E-05	1	9.57E-04	2	7.30E-03	4	9.57E-04	2
[-1,-1,-1]	Best	1.98E-05	1	2.03E-05	3	2.05E-05	4	2.01E-05	2
	std	3.54E-06		3.87E-04		1.30E-03		3.29E-04	
CTP2	Mean	1.14E-05	2	1.22E-03	3	1.35E-05	4	1.01E-05	1
	Worst	2.51E-05	1	2.72E-02	3	2.88E-05	4	2.57E-05	2
[0,0,0]	Best	3.55E-06	1	4.33E-06	2	4.91E-06	4	4.38E-06	3
	std	6.29E-06		5.95E-05		4.30E-02		4.13E-06	
CTP3	Mean	1.80E-04	1	2.66E-04	4	2.37E-04	2	2.41E-04	3
	Worst	3.06E-04	1	4.57E-04	4	3.93E-04	2	4.56E-04	3
[-1,-1,-1]	Best	2.47E-05	1	7.00E-05	3	1.02E-04	4	6.61E-05	2
	std	7.82E-05		9.34E-05		9.00E-05		8.41E-05	
CTP4	Mean	2.20E-03	2	1.90E-03	1	2.70E-03	4	2.30E-03	3
	Worst	6.10E-03	2	4.20E-03	1	7.20E-03	4	6.10E-03	2
[0,0,0]	Best	3.84E-04	1	9.66E-04	3	7.88E-04	2	1.00E-03	4
	std	1.10E-03		2.30E-03		1.70E-03		1.30E-03	
CTP5	Mean	1.60E-03	1	1.80E-03	2	1.90E-03	3	2.20E-03	4
	Worst	2.20E-03	1	3.20E-03	2	4.20E-03	3	8.70E-03	4
[0,0,-1]	Best	1.48E-04	2	1.54E-04	3	1.43E-04	1	1.86E-04	4
	std	6.21E-04		6.71E-04		9.50E-04		1.70E-03	
CTP6	Mean	9.41E-07	1	1.50E-03	2	7.60E-03	4	3.10E-03	3
	Worst	1.26E-06	1	2.22E-02	3	7.90E-03	2	2.22E-02	3
[-1,-1,-1]	Best	7.91E-07	1	1.19E-06	2	1.20E-06	3	1.22E-06	4
	std	1.14E-07		5.70E-03		1.50E-03		7.80E-03	
Summation of Ranks:			35		83		96		79

VII Conclusions

In this paper, an ensemble of three different constraint handling methods is employed with multiobjective differential evolution algorithm, since there is no single constraint handling method that can outperform all other constraint handling methods on every problem. The important property of the ECHMs is that every function call is used by every population associated with each constraint handling method. As evolutionary algorithms are stochastic in nature, the search process passes through different phases at different points during the evolution. Different constraint handling methods can be effective during different stages of the search process. The ensemble of constraint handling methods will bring the best solutions generated by the most suitable technique to all the populations. In this way, the search process will continue using the best parents to generate offspring. Hence, the ECHMs has the potential to perform well over diverse problems when compared to a single CHM. We tested the performances of the proposed ECHMs as well as the three individual constraint handling methods. Experimental results showed that the ECHM overall outperforms all three single constraint handling methods.

Acknowledgment: Authors acknowledge the financial support offered by the A*Star (Agency for Science, Technology and Research, Singapore) under the grant #052 101 0020.

REFERENCES

[1] B. Rosenberg, M. Richards, J. T. Langton, S. Tenenbaum, and D. W. Stouch, "Applications of multiobjective evolutionary algorithms to air operations mission planning," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, New York, NY 10036-5701, United States, 2008.

[2] M. G. C. Tapia and C. A. C. Coello, "Applications of multiobjective evolutionary algorithms in economics and finance: A survey," in *IEEE Congress on Evolutionary Computation*, Piscataway, NJ 08855-1331, United States, 2008.

[3] B. Tessema and G. G. Yen, "A self adaptive penalty function based algorithm for constrained optimization," in *IEEE Congress on Evolutionary Computation*, Piscataway, NJ, USA, 2007.

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

[5] C. A. Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering,* vol. 191, pp. 1245-1287, 2002.

[6] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation,* vol. 1, pp. 67-82, 1997.

[7] 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.* , Paris, France, 2000.

[8] 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.* , Paris, France, 2000.

[9] 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.*, Orlando, FL, USA, 1994.

[10] 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*, Washington, DC, USA, 1999.

[11] 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.*, Birmingham, UK, 2004.

[12] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary Computation,* vol. 4, pp. 1-32, 1996.

[13] S. Koziel and Z. Michalewicz, "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization," *Evolutionary Computation,* vol. 7, pp. 19-44, 1999.

[14] Y. Wang, Z. Cai, Y. Zhou, and W. Zeng, "An Adaptive Tradeoff Model for Constrained Evolutionary Optimization" *IEEE Transactions on Evolutionary Computation,* vol. 12, pp. 80-92, Feb. 2008.

[15] 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, pp. 182-97, 2002.

[16] Y. G. Woldesenbet, B. G. Tessema, and G. G. Yen, "Constraint handling in multi-objective evolutionary optimization," in *2007 IEEE Congress on Evolutionary Computation*, Piscataway, NJ, USA, 2007.

[17] B. Tessema and G. G. Yen, "A self adaptive penalty function based algorithm for constrained optimization," in *IEEE Congress on Evolutionary Computation*, 2006.

[18] R. Courant, "Variational methods for the solution of problems of equilibrium and vibrations," *Bulletin of the American Mathematical Society,* vol. 49, pp. 1-23, 1943.

[19] J. Joines and C. Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs," in *First IEEE Congress on Evolutionary Computation*, Orlando, 1994.

[20] R. Farmani and J. A. Wright, "Self-Adaptive Fitness Formulation for Constrained Optimization," *IEEE Transactions on Evolutionary Computation,* vol. 7, pp. 445-455, 2003.

[21] D. Powell and M. M. Skolnick, "Using genetic algorithms in engineering design optimization with non-linear constraints," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA, USA, 1993.

[22] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering,* vol. 186, pp. 311-338, 2000.

[23] T. Takahama and S. Sakai, "Constrained Optimization by Constrained Particle Swarm Optimizer with -level Control," in *Proc. of 4th IEEE International Workshop on Soft Computing as Transdisciplinary Science and Technology* Muroran, Japan, 2005.

[24] R. Storn and K. V. Price, "Differential evolution-Asimple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization,* vol. 11, pp. 341-359, 1995.

[25] X. Q. Chen, Z. X. Hou, and J. X. Liu, "Multi-objective optimization with modified pareto differential evolution," in *International Conference on Intelligent Computation Technology and Automation,*, Piscataway, NJ 08855-1331, United States, 2008.

[26] J. Fan, S. Xiong, J. Wang, and C. Gong, "IMODE: Improving Multi-objective Differential Evolution algorithm," in *4th International Conference on Natural Computation*, Piscataway, NJ 08855-1331, United States, 2008.

[27] J. q. Zhang and A. C. Sanderson, "Self-adaptive multi-objective differential evolution with direction information provided by archived inferior solutions," in *IEEE Congress on Evolutionary Computation*, Piscataway, NJ 08855-1331, United States, 2008.

[28] K. Zielinski and R. Laur, "Differential evolution with adaptive parameter setting for multi-objective optimization," in *IEEE Congress on Evolutionary Computation*, Piscataway, NJ 08855-1331, United States, 2008.

[29] B. V. Babu and M. M. L. Jehan, "Differential evolution for multi-objective optimization," in *Congress on Evolutionary Computation*, Piscataway, NJ, USA, 2003.

[30] F. Hui-Yuan, J. Lampinen, and Y. Levy, "An easy-to-implement differential evolution approach for multi-objective optimizations," *Engineering Computations,* vol. 23, pp. 124-38, 2006.

[31] L. Zhang, C. Zhou, M. Ma, and C. Sun, "Multi-objective differential evolution algorithm based on max-min distance density," *Jisuanji Yanjiu yu Fazhan/Computer Research and Development,* vol. 44, pp. 177-184, 2007.

[32] J. Knowles, L. Thiele, and E. Zitzler, "A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers," Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland Feb 2006.

[33] M. Tanaka, H. Watanabe, Y. Furukawa, and T. Tanino, "GA-based decision support system for multicriteria optimization," in *IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century* New York, NY, USA, 1995.

[34] N. a. D. Srinivas, K., "Multi-objective function optimization using non-dominated sorting genetic algorithms," *Evolutionary Computation,* pp. 221-248, 1994.

[35] A. Osyczka and S. Kundu, "A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm," *Structural Optimization,* vol. 10, pp. 94-9, 1995.

[36] S. Kukkonen and J. Lampinen, "GDE3: The third evolution step of generalized differential evolution," *KanGAL Report* #2005013, http://www.iitk.ac.in/kangal/papers/k2005013.pdf, 2005.

[37] L. V. Santana-Quintero and C. A. C. Coello, "An algorithm based on differential evolution for multi-objective problems," *Int. J. of Computational Intelligence Research* 1(1-2): 151-169, 2005.

[38] B. Y. Qu, P. N. Suganthan, "Multi-objective Evolutionary Programming without Non-domination Sorting is up to Twenty Times Faster," *Proc. of CEC 2009*, Norway.

[39] C. A. Coello Coello, G. B. Lamont, D. A. Van Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, Springer, 2nd Ed., 2007.

[40] V. L. Huang, P. N. Suganthan A. K. Qin and S. Baskar, "Multiobjective Differential Evolution with External Archive and Harmonic Distance-Based Diversity Measure", *Technical Report*, Nanyang Technological University, Singapore, 2005.

[41] D. E. Goldberg, M. Samtani, "Engineering optimization via genetic algorithm", *Proc. of 9th Conf. on Electronic Computation*, pp. 471-482, University of Alabama, 1986.

Appendix A

Test problems

Test problem 1: TNK

The first problem is proposed by Tanaka [33]:

Minimize $f_{1(x)} = x_1,$

Minimize $f_{2(x)} = x_2,$

Subject to $C_{1(x)} = x_1^2 + x_2^2 - 1 - 0.1\cos\left(16\arctan\frac{x_1}{x_2}\right) \geq 0,$

$C_{2(x)} = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5,$

$0 \leq x_1 \leq \pi,$

$0 \leq x_2 \leq \pi.$

Test problem 2: SRN

The second problem is proposed by Srinivas [34] :

Minimize $f_1(x) = 2 + (x_1 - 2)^2 + (x_2 - 1)^2,$

Minimize $f_2(x) = 9x_1 - (x_2 - 1)^2,$

Subject to $C_1(x) = x_1^2 + x_2^2 \leq 225,$

$C_2(x) = x_1 - 3x_2 + 10 \leq 0,$

$-20 \leq x_1 \leq 20,$

$$-20 \leq x_2 \leq 20.$$

Test problem 3: CONSTR

Minimize $f_1(x) = x_1$

Minimize $f_2(x) = (1 + x_2)/x_1$

Subject to $C_1(x) = 9x_1 + x_2 - 6 \geq 0$

$C_2(x) = 9x_2 - x_2 - 1 \geq 0$

$0.1 \leq x_1 \leq 1,$

$0 \leq x \leq 51.$

Test problem 4: OSY

Osyczka and Kundu [35] used the following six-variable test problem:

Minimize $f_1(x) = -[25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2 + (x_4 - 4)^2 + (x_5 - 1)^2]$

,

Minimize $f_2(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2,$

Subject to $C_1(x) = x_1 + x_2 - 2 \geq 0,$

$C_2(x) = 6 - x_1 - x_2 \geq 0,$

$C_3(x) = 2 - x_2 + x_1 \geq 0,$

$C_4(x) = 4 - (x_3 - 3)^2 - x_4 \geq 0,$

$C_5(x) = 2 - x_1 + 3x_2 \geq 0,$

$$C_6(x) = (x_2 - 3)^2 + x_6 - 4 \geq 0,$$

$$x_1 \geq 0,\ x_2, x_6 \leq 10,$$

$$1 \leq x_3, x_4 \leq 5, \qquad 0 \leq x_5 \leq 6.$$

Test problem 4: CTP1 [4]

Minimize $f_1(x) = x_1$

Minimize $f_2(x) = g(x)\exp\left(-\frac{f_1(x)}{g(x)}\right)$

Subject to $f_2(x) - a_j \exp(-b_j f_1(x)) \geq 0,$

$j = 1, \ldots, J$

Where $0 \leq x_1 \leq 1, -5 \leq x_2, x_3, x_4 \leq 5$ *and*

$g(x) = 91 + \sum_{i=2}^{4}(x_i^2 - 10\cos(4\pi x_i))$

$J = 2,\ a_1 = 0.858, a_2 = 0.541, a_2 = 0.728$ *and* $b_2 = 0.295$

Test problems 5-10: CTP2-CTP6

Minimize $f_1(x) = x_1$

Minimize $f_2(x) = g(x)\left(1 - \sqrt{\frac{f_1}{g(x)}}\right)$

Subject to $\cos(\theta)\,(f_2(x) - e) - \sin(\theta) f_1(x) \geq$

$a|\sin(b\pi(\sin(\theta)\,(f_2(x) - e) + \cos(\theta) f_1(x))^c)|^d$

Where $0 \leq x_1 \leq 1, -5 \leq x_2, x_3, x_4 \leq 5$ *and*

$$g(x) = 91 + \sum_{i=2}^{4}(x_i^2 - 10\cos(4\pi x_i))$$

The parameters chosen for the different CTP2 to CTP6 problems are listed in Table VII

Table IX. The parameters for CTP2 to CTP6

	θ	a	b	c	d	e
CTP2	-0.2π	0.2	10	1	6	1
CTP3	-0.2π	0.1	10	1	0.5	1
CTP4	-0.2π	0.75	10	1	0.5	1
CTP5	-0.2π	0.75	10	2	0.5	1
CTP6	-0.05π	40	5	1	6	0