A Fast Algorithm on Finding the Non-dominated Set in Multi-objective Optimization

Lixin Ding
State Key Laboratory of Software Engineering,Wuhan University, Wuhan 430072, P.R. China
lx_ding@263.net
Sanyou Zeng
Dept. of Computer Science, China University of GeoSciences, Wuhan 430074, P.R. China
Dept. of Computer Science, Zhuzhou Institute of Technology,Zhuzhou 412008, P.R. China
sanyou-zeng@263.net
Lishan Kang
Dept. of Computer Science, China University of GeoSciences,Wuhan 430074 , P.R.China
kang@whu.edu.cn

Abstract- A fast algorithm is proposed to find the non-dominated set for multi-objective optimization problems in this paper. Two accelerated techniques are adopted in the algorithm. One is that the algorithm can yield an integer rank set after it indexes the the search space. Based on this, the goal is changed into determination of the non-dominated set of the integer rank set. The other is that the non-dominated check sequence follows that the likely non-dominated members are first checked, and that the check process is stopped when the remaining members in the non-dominated check sequence are dominated. The computational complexity of the new algorithm is analyzed theoretically. Experimental results show that the new method performs much better than KLP(a famous effective algorithm) when the search space contains a large non-dominated set. Moreover, the two new techniques introduced in this paper are very useful for multi-objective evolutionary algorithms (MOEAs) to improve the computational speed.
Keywords: non-dominated set, partial ordered relation, multi-objective optimization.

1 Introduction

Almost every real-world problem involves simultaneous optimization of several incommensurable and often competitive objectives. And in wide sense, to solve a multi-objective problem is to find the set of Pareto-optimal solutions, mathematically, the non-dominated set of the search space. Classical optimization methods can at best find one solution in one simulation run, while evolutionary algorithms (EAs) can find multiple optimal solutions in one single simulation run due to their search based on population. Thus, EAs are ideal candidates for solving multi-objective optimization problems. Some typical MOEAs include Vector Evaluated Genetic Algorithm (VEGA)([Schaffer85]), Pareto-based Ranking Procedure (FFGA)([Fonseca93]), Niched Pareto Genetic Algorithm (NPGA)([Horn93], [Horn94]), Non-dominated Sorting Genetic Algorithm (NSGA)([Srinivas94), Strength Pareto Evolutionary Algorithm (SPEA)([Zitzler99]) and Generalised Regression GA (GRGA)([Tiwari02). Recently, there are some new approaches or modified versions of proposed methods yet, for instance, NSGAII([Deb00]), SPEA2([Zitzler01]), rMOGAxs([Purshouse01]) and Pareto-Archived Evolution Strategy (PAES)([Kwnowles00]) and so on. It is well known that almost all MOEAs can't avoid searching a non-dominated set, thus, in some sense, the pitfall of MOEAs is time-consuming. Upon the above consideration, it is important for us to develop a fast procedure to find the non-dominated set of an objective vector population.

In this paper, a fast method is proposed to find the non-dominated set of a vector set. Enlightened by the effective method KLP ([Kung et al75]), in which sorting technique was adopted, the new algorithm indexes the original set in another way just like the indexing operation in database system and yields an integer rank set corresponding to the original set. Therefore, the goal is converted to find the non-dominate set of the integer rank set, instead of finding the non-dominated set of the original set. Since computers deal with integers very fast, finding an integer non-dominated set will be very fast. Based on the indices, the check sequence statistically follows that the non-dominated members are first checked, and when the remaining members are dominated, the check process is stopped. In this way, the computational expenditure is reduced effectively.

The remaining part of this paper is organized as follows. In Section 2, we briefly mention the Non-dominated Set and an efficient algorithm for finding non-dominated set. Then, the new fast method is presented in Section 3. Thereafter, Section 4 gives some numerical experiments and the related discussion. Finally, we outline the conclusions of this paper in Section 5.

2 Preliminaries

Let $\mathbf{X}$ be a set $\mathbf{X} = \{\mathbf{x}^{(i)} | i = 1, 2, ..., N\}$, where $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_M^{(i)})$ is a vector. We can also regard $\mathbf{X}$ as a matrix $\mathbf{X} = [x_j^{(i)}]_{N \times M}$, where the ith row $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_M^{(i)})$, $i = 1, 2, ..., N$, the jth column $\mathbf{x}_j = (x_j^{(1)}, x_j^{(2)}, ..., x_j^{(N)})^T$, $j = 1, 2, ..., M$.

Definition 1 *For any two vectors $x^{(i_1)}, x^{(i_2)} \in X$,*

$$\begin{array}{lll} x^{(i_1)} = x^{(i_2)} & \Longleftrightarrow & \forall j \in 1, 2, ..., M : x_j^{(i_1)} = x_j^{(i_2)} \\ x^{(i_1)} \leq x^{(i_2)} & \Longleftrightarrow & \forall j \in 1, 2, ..., M : x_j^{(i_1)} \leq x_j^{(i_2)} \\ x^{(i_1)} < x^{(i_2)} & \Longleftrightarrow & x^{(i_1)} \leq x^{(i_2)} \wedge x^{(i_1)} \neq x^{(i_2)} \end{array} \quad (1)$$

The relation "$\leq$" on $\mathbf{X}$ is a partial order relation, and "$<$" a strict partial order relation. $\mathbf{X}$ with relation "$<$" is a strict partial order set, where we denote it by $(\mathbf{X}, <)$.

Definition 2 *A member $x' \in X$ is said to be a non-dominated member of $(X, <)$ iff*

$$\nexists x \in X : x < x' \quad (2)$$

The set

$$\begin{array}{l} M(X, <) = \{x \in X | x \ is \\ a \ non - dominated \ member \ of | (X, <)\} \end{array} \quad (3)$$

is called the non-dominated set of $(X, <)$.

The Pareto-optimal set in a multi-objective optimization problem is the non-dominated set of the feasible decision space. This paper will focus on the method for finding the non-dominated set of the strict partial order set $(\mathbf{X}, <)$.

Let us introduce an efficient method proposed by Kung, H. T., Luccio, F. and Preparata, F. P.([Kung et al75]), where we denote it by KLP.

Algorithm 1 *Identifying the Non-dominated Set:KLP*

1. *Sort the set X according to the descending order of importance of the first column component.*
2. *Nondominance(X): If $|X| = 1$, then return X as the output of Nondominance(X).*
   Otherwise, $T = Nondominance(X^{(1)} - X^{(\frac{|X|}{2})})$ and $B = Nondominance(X^{(\frac{|X|}{2}+1)} - X^{(|X|)})$, where $X^{(1)} - X^{(\frac{|X|}{2})}$ is the top half of X, $X^{(|\frac{|X|}{2}+1)} - X^{(|X|)}$ the bottom half. For any member $x^{(i)} \in B$, if $x^{(i)}$ is not dominated by all members in T, create a merged set $M = T \bigcup \{x^{(i)}\}$. Return M as the output of the output of Nondominance(X).

The complexity of this method is $\mathbf{O}(N(\log N)^{M-2})$ for $M \geq 4$ and $\mathbf{O}(N(\log N))$ for $M = 2, 3$. For more existing algorithms about finding non-dominated set, See Deb (Deb01]).

3 A Fast Algorithm

The significant idea of our algorithm is mainly that the most likely non-dominated members are first checked. For convenience, denote the new method by **MLNFC**

3.1 Indexing and the Rank Set

Many multi-objective evolutionary algorithms sort population according to the objectives in order to improve convergence rate. One of the reason why the KLP is efficient lies in the use of sort technique. Enlightened by those works, the new approach indexes the $\mathbf{X}$ in the same way as indexing a table in a database system: the members $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}$ are like records and the columns $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_M$ are like attributes. An index contains pointers to all the members in $\mathbf{X}$ and is sorted by the search key values. We index the $\mathbf{X}$ in M different ways. The first index $\mathbf{I}_1$ is based on the sorting order of the first column called the first search key, for members with equal values in the first column, the order is determined by that of the second column called the second search key, do in such way until the last column called the Mth search key; the first search key of the second index $\mathbf{I}_2$ is the second column, the second search key the third column, ..., the $(M-1)$th search key the Mth column, the Mth search key the first column; .....; the first search key of the Mth index $\mathbf{I}_M$ is the Mth column, the second search key the first column, ..., the Mth search key the $(M-1)$th column. Let's see an example

Example 1

$$X = \begin{bmatrix} (5.5 & 6.5 & 7.0) \\ (2.5 & 8.5 & 5.0) \\ (4.5 & 7.0 & 7.0) \\ (5.5 & 7.5 & 9.5) \\ (9.5 & 1.5 & 8.5) \end{bmatrix} \quad (4)$$

We have $\mathbf{I}_1$, $\mathbf{I}_2$, $\mathbf{I}_3$ (See (5), (6) and (7) bolow)

$$\mathbf{X} = \begin{bmatrix} (5.5 & 6.5 & 7.0) \\ (2.5 & 8.5 & 5.0) \\ (4.5 & 7.0 & 7.0) \\ (5.5 & 7.5 & 9.5) \\ (9.5 & 1.5 & 8.5) \end{bmatrix} \quad \mathbf{I}_1 = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 4 \\ 5 \end{bmatrix} \quad (5)$$

$$\mathbf{X} = \begin{bmatrix} (5.5 & 6.5 & 7.0) \\ (2.5 & 8.5 & 5.0) \\ (4.5 & 7.0 & 7.0) \\ (5.5 & 7.5 & 9.5) \\ (9.5 & 1.5 & 8.5) \end{bmatrix} \quad \mathbf{I}_2 = \begin{bmatrix} 5 \\ 1 \\ 3 \\ 4 \\ 2 \end{bmatrix} \quad (6)$$

$$\mathbf{X} = \begin{bmatrix} (5.5 & 6.5 & 7.0) \\ (2.5 & 8.5 & 5.0) \\ (4.5 & 7.0 & 7.0) \\ (5.5 & 7.5 & 9.5) \\ (9.5 & 1.5 & 8.5) \end{bmatrix} \quad \mathbf{I}_3 = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 5 \\ 4 \end{bmatrix} \tag{7}$$

On the other hand, each member $\mathbf{x}^{(i)}$ $(i = 1, 2, ..., N)$ in $\mathbf{X}$ has its position (or rank) $r_j^{(i)}$ in each index $\mathbf{I}_j$ $(j = 1, 2, ..., M)$ according to the search key order, then $\mathbf{x}^{(i)}$ has a corresponding rank vector $\mathbf{r}^{(i)} = (r_1^{(i)}, r_2^{(i)}, ..., r_M^{(i)})$. Thus, $\mathbf{X}$ determines a set $\mathbf{R} = \{\mathbf{r}^{(1)}, \mathbf{r}^{(2)}, ..., \mathbf{r}^{(N)}\}$ which is called **the rank set** of $\mathbf{X}$. That the indices $\mathbf{I}_1$, $\mathbf{I}_2$, ..., $\mathbf{I}_M$ also index the rank set $\mathbf{R}$ is just like they index $\mathbf{X}$. For the above example, $\mathbf{x}^{(1)}$ has its rank 3, 2, 3 in indices $\mathbf{I}_1$, $\mathbf{I}_2$, and $\mathbf{I}_3$ respectively. Therefore, it determines a rank vector $\mathbf{r}^{(1)} = (3, 2, 3)$. Similarly, $\mathbf{x}^{(2)}$ determines $\mathbf{r}^{(2)} = (1, 5, 1)$, and so on(See (8) below). For the rank set, see (9) below.

$$\begin{array}{l} \mathbf{x}^{(1)} = \begin{pmatrix} 5.5 & 6.5 & 7.0 \end{pmatrix} \longrightarrow \mathbf{r}^{(1)} = \begin{pmatrix} 3 & 2 & 3 \end{pmatrix} \\ \mathbf{x}^{(2)} = \begin{pmatrix} 2.5 & 8.5 & 5.0 \end{pmatrix} \longrightarrow \mathbf{r}^{(2)} = \begin{pmatrix} 1 & 5 & 1 \end{pmatrix} \\ \mathbf{x}^{(3)} = \begin{pmatrix} 4.5 & 7.0 & 7.0 \end{pmatrix} \longrightarrow \mathbf{r}^{(3)} = \begin{pmatrix} 2 & 3 & 2 \end{pmatrix} \\ \mathbf{x}^{(4)} = \begin{pmatrix} 5.5 & 7.5 & 9.5 \end{pmatrix} \longrightarrow \mathbf{r}^{(4)} = \begin{pmatrix} 4 & 4 & 5 \end{pmatrix} \\ \mathbf{x}^{(5)} = \begin{pmatrix} 9.5 & 1.5 & 8.5 \end{pmatrix} \longrightarrow \mathbf{r}^{(5)} = \begin{pmatrix} 5 & 1 & 4 \end{pmatrix} \end{array} \tag{8}$$

$$\mathbf{X} = \begin{bmatrix} (5.5 & 6.5 & 7.0) \\ (2.5 & 8.5 & 5.0) \\ (4.5 & 7.0 & 7.0) \\ (5.5 & 7.5 & 9.5) \\ (9.5 & 1.5 & 8.5) \end{bmatrix} \Longleftrightarrow \mathbf{R} = \begin{bmatrix} (3 & 2 & 3) \\ (1 & 5 & 1) \\ (2 & 3 & 2) \\ (4 & 4 & 5) \\ (5 & 1 & 4) \end{bmatrix} \tag{9}$$

Obviously, the creation of the rank set $\mathbf{R}$ implies an one-one map between $\mathbf{X}$ and $\mathbf{R}$: $\mathbf{x}^{(i)} \longleftrightarrow \mathbf{r}^{(i)}$, $(i = 1, 2, ..., N)$. By this map, it is easy to see that the non-dominated sets of both $\mathbf{X}$ and $\mathbf{R}$ determines each other. Therefore, the goal to find non-dominated set $\mathbf{M}(\mathbf{X}, <)$ may be converted to find non-dominated set $\mathbf{M}(\mathbf{R}, <)$, and the computational efficiency can be obviously improved because of the well-known reasons.

3.2 Determination of Check Sequences

The order of $\mathbf{R}$ sorted by $\mathbf{I}_j$ is actually a topological order by the strict partial order relation (See definition 1), which means that the non-dominated members stay in front and the dominated ones remain in rear. We hope that the likely non-dominated members are checked first and skip the dominated ones. Therefore, we have the following check order (Denote the member pointed by $\mathbf{I}_j(i)$ by $\mathbf{I}_j(i)'s$): first, check $\mathbf{I}_1(1)'s$, $\mathbf{I}_2(1)'s$, ..., $\mathbf{I}_M(1)'s$; secondly, check $\mathbf{I}_1(2)'s$, $\mathbf{I}_2(2)'s$, ..., $\mathbf{I}_M(2)'s$; and so on. The process is at most a N loop iterations called **check iteration** and yields a sequence $\mathbf{I}_1(1)$, $\mathbf{I}_2(1)$, ..., $\mathbf{I}_M(1)$; $\mathbf{I}_1(2)$, $\mathbf{I}_2(2)$, ..., $\mathbf{I}_M(2)$; .....; $\mathbf{I}_1(N)$, $\mathbf{I}_2(N)$, ..., $\mathbf{I}_M(N)$ called **indices check sequence**. Under the control of the index check sequences, members checked in $\mathbf{R}$ are non-dominated ones or not. To avoid checking duplicate members, we must perform as follows: before checking a member, we shall first see whether it has been checked; if it is not checked, then we check its non-dominance and set it checked, else skip it and deal with the next member. Therefore, a **non-dominance check sequence** yields.

For the example 1, we have the index check sequences: $\mathbf{I}_1(1)$, $\mathbf{I}_2(1)$, $\mathbf{I}_3(1)$; $\mathbf{I}_1(2)$, $\mathbf{I}_2(2)$, $\mathbf{I}_3(2)$; $\mathbf{I}_1(3)$, $\mathbf{I}_2(3)$, $\mathbf{I}_3(3)$; $\mathbf{I}_1(4)$, $\mathbf{I}_2(4)$, $\mathbf{I}_3(4)$; $\mathbf{I}_1(5)$, $\mathbf{I}_2(5)$, $\mathbf{I}_3(5)$ (See (10) below).

$$\begin{array}{ccccc} \mathbf{I}_1 & & \mathbf{I}_2 & & \mathbf{I}_3 \\ \begin{bmatrix} 2 \\ \hookrightarrow 3 \\ \hookrightarrow 1 \\ \hookrightarrow 4 \\ \hookrightarrow 5 \end{bmatrix} & \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} & \begin{bmatrix} 5 \\ 1 \\ 3 \\ 4 \\ 2 \end{bmatrix} & \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} & \begin{bmatrix} 2_{\hookleftarrow} \\ 3_{\hookleftarrow} \\ 1_{\hookleftarrow} \\ 5_{\hookleftarrow} \\ 4 \end{bmatrix} \end{array} \tag{10}$$

The corresponding member order is $\mathbf{r}^{(2)}$, $\mathbf{r}^{(5)}$, $\mathbf{r}^{(2)}$, $\mathbf{r}^{(3)}$, $\mathbf{r}^{(1)}$, $\mathbf{r}^{(3)}$, $\mathbf{r}^{(1)}$, $\mathbf{r}^{(3)}$, $\mathbf{r}^{(1)}$, $\mathbf{r}^{(4)}$, $\mathbf{r}^{(4)}$, $\mathbf{r}^{(5)}$, $\mathbf{r}^{(5)}$, $\mathbf{r}^{(2)}$, $\mathbf{r}^{(4)}$, while the non-dominated check sequence is

$$\mathbf{I}_1(1), \mathbf{I}_2(1), \mathbf{I}_1(2), \mathbf{I}_2(2), \mathbf{I}_1(4) \tag{11}$$

and the actual member check order is

$$\mathbf{r}^{(2)}, \mathbf{r}^{(5)}, \mathbf{r}^{(3)}, \mathbf{r}^{(1)}, \mathbf{r}^{(4)} \tag{12}$$

Since the order determined by each index is that the non-dominated members stay in front and the dominated ones stay in rear. Statistically, the check of the non-dominated members precedes that of the dominated. In order to design a fast algorithm to find non-dominated set, we have done some researches for the check sequence. The following propositions are a few internal rules discovered in the check sequence.

Proposition 1 *Suppose $I_{j_1}(i_1), I_{j_2}(i_2), ..., I_{j_N}(i_N)$ is the non-dominance check sequence, then the index entry sequence $i_1, i_2, ..., i_N$ is ascendent.*

For the example 1, the index entry sequence corresponding to the check sequence is $1, 1, 2, 2, 4$(See (11) above).

Proposition 2 *$I_j(1)'s$ must be non-dominated.*

For the example 1, the members $\mathbf{I}_1(1)'s$ (i.e. $\mathbf{r}^{(2)}$) , $\mathbf{I}_2(1)'s$ (i.e. $\mathbf{r}^{(5)}$) are non-dominated by the proposition 2 (See (11) and (12) above). The algorithm in this paper will make such members be the initial members of the non-dominated set.

Proposition 3 *$I_j(i)'s$ is not dominated by $I_j(i+1)'s, I_j(i+2)'s, ..., I_j(N)'s$*

By propositions 3, in order to determine the non-dominance of $\mathbf{I}_j(i)'s$, comparing $\mathbf{I}_j(i)'s$ with $\mathbf{I}_j(1)'s$, $\mathbf{I}_j(2)'s$, ..., $\mathbf{I}_j(i-1)'s$ will do. For the example 1, although the current non-dominated set is $\{\mathbf{r}^{(2)}, \mathbf{r}^{(5)}, \mathbf{r}^{(3)}\}$, in order to determine the non-dominance of $\mathbf{I}_2(2)'s$ (i.e. $\mathbf{r}^{(1)}$), we only need to compare it with $\mathbf{I}_2(1)'s$ (i.e. $\mathbf{r}^{(5)}$).

Proposition 4 *Suppose $I_j(i)$ stays in the non-dominated check sequence, then $I_j(i)'s$ has its all ranks (components) more than or equal to i.*

For the example 1, in the non-dominance check sequence (See (11) above), $\mathbf{I}_1(4)'s$ is $(4, 4, 5)$ and $\mathbf{I}_1(2)'s$ is $(2, 3, 2)$, which verifies the proposition 4.

Proposition 5 *If a checked member has its all ranks less than or equal to i', then it dominates the remaining members of the non-dominated check sequence, where the corresponding index entries are larger than or equal to i'.*

It is easy to verify proposition 5 by proposition 1 and 4. Therefore, if all ranks of a non-dominated member are less than or equal to i', the check iteration should be stopped before i' by proposition 5. From this point we observe that the computational process is shortened.

3.3 The Procedure of MLNFC

Based on the above indices, rank set and propositions, we have the following procedure which identifies the non-dominated set as soon as possible.

Algorithm 2 *Identifying the Non-dominated Set: MLNFC.*
Input: X.
Output:The non-dominated set —X' of X
(R' store the non-dominate set of rank set R).
Using quick-sort method to create indices $I_1, I_2, ..., I_M$ and rank set R;
Initiate R' with $I_1(1)'s$, $I_2(1)'s$, ..., $I_M(1)'s$, eliminate duplicate ones;
for(i = 1; i≤N; i++) Set $r^{(i)}$ not checked;
stop = N;// Set initial termination position N;
for(i=2;i≤stop;i++){
//Search at the ith entries of the indices;
for(j=1;j≤M;j++){//Search at the ith entry of index I_j;
if($I_j(i)'s$ not checked){
Compare it with the non-dominated ones among $I_j(1)'s, I_j(2)'s, ..., I_j(i-1)'s$ and set it checked;
if($I_j(i)'s$ is not dominated by any of them){
put it in R';
i' =max of its ranks(or components);
stop=min{stop, i'};//Update termination position;
}
}
}
}
Get X' corresponding to R' and output it as the non-dominated set of X;

4 Discussion and Experimental Results

4.1 Computational Complexity

X is created at random. The computation is divided into two parts. The first is the computation of indexing the **X** in M different ways by using quick-sort. It is well known that the number of comparisons in quick-sort is about $O(N \log N)$. Therefore, the number of comparisons on indexing in our algorithm is

$$S_1 = O(MN \log N) \tag{13}$$

The second part is the number of comparisons for finding Non-dominated set based on the indices. For computational convenience, suppose each component of each vector in **R** has its value taken evenly from the set $\{1, 2, ..., N\}$. Regard randomly generating an $\mathbf{X}(\mathbf{R})$ with size $M \times N$ as an experiment, the size of the sample space Ω is $C_{N^M}^N$. Consider the following N events

$$\begin{aligned} &E_1 = \{\mathbf{R} | \exists \mathbf{r}^{(i_0)} \in \mathbf{R}, r_j^{(i_0)} = 1, j = 1, 2, ..., M\} \\ &E_2 = \{\mathbf{R} | \exists \mathbf{r}^{(i_0)} \in \mathbf{R}, r_j^{(i_0)} \le 2, j = 1, 2, ..., M\} \\ &..... \\ &E_N = \{\mathbf{R} | \exists \mathbf{r}^{(i_0)} \in \mathbf{R}, r_j^{(i_0)} \le N, j = 1, 2, ..., M\} \end{aligned} \tag{14}$$

It is easy to know that

$$\begin{aligned} &E_1 \subset E_2 \subset ... \subset E_N = \Omega \\ &|E_1| = C_{N^M}^{N-1} \\ &|E_2| = C_{N^M}^N - C_{N^M - 2^M}^N \\ &... \\ &|E_{N-1}| = C_{N^M}^N - C_{N^M-(N-1)^M}^N \\ &|E_N| = C_{N^M}^N \end{aligned} \tag{15}$$

The events we are interested in are

$$E_1' = E_1, E_2' = E_2 - E_1, ..., E_N' = E_N - E_{N-1} \tag{16}$$

Obviously

$$\begin{aligned} &E_i' \bigcap E_j' = \Phi; i, j = 1, 2, ..., N; i \ne j \\ &\bigcup_{i=1}^N E_i' = \Omega \\ &|E_i'| = C_{N^M-(i-1)^M}^N - C_{N^M-i^M}^N, i = 1, 2, ..., N-1 \\ &|E_N'| = C_{N^M-(N-1)^M}^N \end{aligned} \tag{17}$$

In fact, it is easy to assert that the number of iterations in the new algorithm is i for finding nondominated set when the **R** created stays in $E_i', i = 1, 2, ..., N$. Therefore, the

expectation of the number of iterations is

$$\begin{aligned} L &= \sum_{i=1}^{N} \frac{|E_i'|}{C_{NM}^N} \times i \\ &= \frac{C_{NM}^N - C_{NM-1M}^N}{C_{NM}^N} \times 1 + ... \\ &\quad + \frac{C_{NM-(N-2)M}^N - C_{NM-(N-1)M}^N}{C_{NM}^N} \times (N-1) \\ &\quad + \frac{C_{NM-(N-1)M}^N}{C_{NM}^N} \times N \\ &= \frac{C_{NM}^N}{C_{NM}^N} + \frac{C_{NM-1M}^N}{C_{NM}^N} + ... + \frac{C_{NM-(N-1)M}^N}{C_{NM}^N} \\ &= \sum_{i=0}^{N-1} \frac{C_{NM-iM}^N}{C_{NM}^N} \end{aligned} \quad (18)$$

To verify the above analysis, we get an average number of iterations of the new algorithm from 500 repetition experiments, and denote the average by L_0. Experimental results in Table 1 show that L_0 approximates L.

Table 1: Verifying the expectation of the number of iterations L in the new algorithm. L_0 denotes the average number of iterations of the new algorithm from 500 repetition experiments.

Size of **X**	L	L_0
100×3	19.70	18.64
100×5	37.01	35.83
1000×3	89.77	89.74
1000×5	231.11	228.94

Now, under the condition that $\mathbf{I}_1(1)$, $\mathbf{I}_2(1)$, ..., $\mathbf{I}_M(1)$; $\mathbf{I}_1(2)$, $\mathbf{I}_2(2)$, ..., $\mathbf{I}_M(2)$; .....; $\mathbf{I}_1(i)$, $\mathbf{I}_2(i)$, ..., $\mathbf{I}_{j-1}(i)$ have been checked, we consider the probability that $\mathbf{I}_j(i)'s$ is not checked, and denote the probability by $p_{i,j}$. It is easy to know

$$\begin{aligned} &p_{1,1} = 1 \\ &p_{1,2} = 1 - \tfrac{1}{N} \\ &... \\ &p_{1,M} = \left(1 - \tfrac{1}{N}\right)^{M-1} \\ \\ &p_{2,1} = \left(1 - \tfrac{1}{N}\right)^{M-1} \\ &p_{2,2} = \left(1 - \tfrac{1}{N}\right)^{M-2} \left(1 - \tfrac{2}{N}\right) \\ &... \\ &p_{2,M} = \left(1 - \tfrac{2}{N}\right)^{M-1} \\ &..... \\ &p_{N-1,1} = \left(\tfrac{2}{N}\right)^{M-1} \\ &p_{N-1,2} = \left(\tfrac{2}{N}\right)^{M-2} \left(\tfrac{1}{N}\right) \\ &... \\ &p_{N-1,M} = \left(\tfrac{1}{N}\right)^{M-1} \\ \\ &p_{N,1} = \left(\tfrac{1}{N}\right)^{M-1} \\ &p_{N,2} = 0 \\ &... \\ &p_{N,M} = 0 \end{aligned}$$

And it is easy to verify that $\sum_{i=1}^{N} \sum_{j=1}^{M} p_{i,j} = N$

To determine the non-dominance of $\mathbf{I}_j(i)'s$, we only need to compare it with $\mathbf{I}_1(i)'s$, $\mathbf{I}_2(i)'s$, ..., $\mathbf{I}_{j-1}(i)'s$, that is, we only need $i-1$ comparisons. Without loss of generality, suppose that the expectation of the number of the iteration L (See (18) above) is an integer, the average member of comparisons for the new algorithm to find non-dominated set is

$$\begin{aligned} S_2 &= (1 + (1 - \tfrac{1}{N}) + ... + (1 - \tfrac{1}{N})^{M-1}) \times 0 \\ &\quad + ((1 - \tfrac{1}{N})^{M-1} + (1 - \tfrac{1}{N})^{M-2}(1 - \tfrac{2}{N}) + \\ &\quad ... + (1 - \tfrac{2}{N})^{M-1}) \times 1 + \\ &\quad ..... \\ &\quad + ((1 - \tfrac{L-1}{N})^{M-1} + (1 - \tfrac{L-1}{N})^{M-2}(1 - \tfrac{L}{N}) + \\ &\quad ... + (1 - \tfrac{L}{N})^{M-1}) \times (L-1) \\ &= N \times ((1 - \tfrac{1}{N})^{M} - (1 - \tfrac{2}{N})^{M}) \times 1 + \\ &\quad ... + N \times ((1 - \tfrac{L-1}{N})^{M} - (1 - \tfrac{L}{N})^{M}) \times (L-1) \\ &= N \times ((1 - \tfrac{1}{N})^{M} + (1 - \tfrac{2}{N})^{M} + \\ &\quad ... + (1 - \tfrac{L-1}{N})^{M} - (L-1)(1 - \tfrac{L}{N})^{M}) \\ &< N(L-1) \end{aligned} \quad (19)$$

Thus, the sum of comparisons of the new algorithm is

$$S_{MLNFC} = S_1 + S_2 = \mathbf{O}(MN \log N) + \mathbf{O}(NL) \quad (20)$$

where $L = \sum_{i=0}^{N-1} \frac{C_{NM-iM}^N}{C_{NM}^N}$.

4.2 Experimental Results

The search of non-dominated sets was performed in a computer of Pentium 4 with 1.80GHz CPU and 256M memory. The Table 2, 3, 4 and 5 below give the running time comparison of MLNFC with KLP for finding non-dominated set. The results show that when M is small(or original set has small non-dominated set), MLNFC is a little slower than KLP, but when M is relatively larger(or original set has a larger non-dominated set), MLNFC is much faster than KLP.

Table 2: The time comparison of MLNFC with KLP , where M = 3, Q is the size of non-dominated set.

$N \times M$	Running Time(Second)		Q
	KLP	MLNFC	
1000×3	0.0150	0.0160	26
5000×3	0.0780	0.1250	45
10000×3	0.1720	0.2820	49
20000×3	0.3440	0.6250	68
50000×3	0.9530	1.8120	83

Table 3: The time comparison of MLNFC with KLP, where M = 5, Q is the size of non-dominated set.

$N \times M$	Running Time(Second)		Q
	KLP	MLNFC	
1000×5	0.0320	0.0460	172
5000×5	0.2030	0.2500	317
10000×5	0.3910	0.5620	278
20000×5	0.9070	1.1880	513
50000×5	2.5150	3.8750	806

Table 4: The time comparison of MLNFC with KLP, where M = 10, Q is the size of non-dominated set.

$N \times M$	Running Time(Second)		Q
	KLP	MLNFC	
1000×10	0.1560	0.0780	732
5000×10	2.7970	0.8900	2962
10000×10	10.7810	2.7030	5091
20000×10	36.5000	8.7350	8321
50000×10	200.1410	45.7660	16385

Table 5: The time comparison of MLNFC with KLP, where M = 30, Q is the size of non-dominated set.

$N \times M$	Running Time(Second)		Q
	KLP	MLNFC	
1000×30	0.2340	0.2190	1000
5000×30	6.6410	1.8130	5000
10000×30	30.9850	5.0160	10000
20000×30	137.1720	15.2500	20000
50000×30	938.0460	75.5470	50000

4.3 Availability

The main goal of MLNFC lies in solving multi-objective optimization problems effectively. An MOEA must possess two distinct features: (1) discovering solutions as close to the Pareto-optimal solutions as possible, and (2) finding as diverse solutions as possible in the obtained non-dominated front. The first feature requires a procedure to find non-dominated set. Therefore, many MOEAs employ KLP. However, the performance of KLP is totally poorer than MLNFC according to the above experiments, so MLNFC may be a better choice. The second feature requires a procedure to locate in an uniformly enough distributed Pareto-optimal front. In order to achieve these aims, many MOEAs use some techniques such as sharing function, crowding distance, clustering and so on, which may improve the performance of MOEAs in some extent. But, MOEAs (MOGAs) should be recommended to employ the sorting or indexing technique similar to MLNFC to get a higher computational performance.

5 Conclusion

Two techniques are employed to speed up the search for non-dominated set in MLNFC. One is that determination of non-dominated set of the original set is converted into determination of the non-dominated set of an integer rank set. The other is that the non-dominated check sequence follows that those possible non-dominated members are first checked, and the check process is stopped when the remaining members of the non-dominated check sequence are found to be dominated. Experimental results show that the new method performs much better than KLP when the original set has a large non-dominated set. Actually, these techniques adopted in the paper is very adequate for many multi-objective evolutionary algorithms (MOEAs) to raise the computational speed.

Acknowledgment This work was supported by the National Natural Science Foundation of China (No.s: 60204001, 70071042, 60073043, 60133010), the Scientific Research Foundation of Hunan Provincial Education Department(No. 02C640), Youth Chengguang Project of Science and Technology of Wuhan City(No.:20025001002) and the Opening Research Foundation at State Key Lab of Software Engineering respectively.

Bibliography

[Deb00] Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T. (2000) "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGAII," In M. S. *et al.* (Ed.), Parallel Problem Solving from Nature-PPSN VI, Berlin, pp. 849-858. Springer.

[Deb01] Deb K. (2001) ”Multi-objective Optimization Using Evolutionary Algorithms,” JOHN WILEY&SONS, LTD, 2001 pp.33-43

[Fonseca93] Fonseca, C. M. and P. J. Fleming (1993) ”Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization,” In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, pp. 416C423. Morgan Kaufmann.

[Horn93] Horn, J. and N. Nafpliotis (1993) ”Multiobjective optimization using the niched pareto genetic algorithm,” IlliGAL Report 93005, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, Champaign.

[Horn94] Horn, J., N. Nafpliotis, and D. E. Goldberg(1994) ”A niched pareto genetic algorithm for multiobjective optimization,” In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Computation*, Volume 1, Piscataway, NJ, pp. 82C87. IEEE.

[Kwnowles00] Knowles, J. D. and Corne, D. W.(2000) ”Reducing local optima in single-objective problems by multi-objectivization,” In Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimzation (EMO-2000),pp. 269-283.

[Kung et al75] Kung, H. T., Luccio, F. and Preparata, F. P. (1975) ”On finding the maxima of a set of vectors,” Jornal of the Association for Computing Machinery 22(4) 469-476.

[Purshouse01] Purshouse, R. C., Fleming, P. J. (2001) ”The Multi-objective Genetic Algorithm Applied to Benchmark Problems–an Analysis,” Research Report No. 796. Department of Automatic Control and Systems Engineering University of Sheffield, Sheffield, S1 3JD, UK.

[Schaffer85] Schaffer, J. D. (1985) ”Multiple objective optimization with vector evaluated genetic algorithms,” In J. J. Grefenstette (Ed.),*Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, pp. 93C100. sponsored by Texas Instruments and U.S. Navy Center for Applied Research in Artificial Intelligence (NCARAI).

[Srinivas94] Srinivas, N. and K. Deb (1994) ”Multiobjective optimization using non-dominated sorting in genetic algorithms,” Evolutionary Computation 2(3), 221C248.

[Tiwari02] Tiwari, A., Roy, R.(2002) ”Generalised Regression GA for Handling Inseparable Function Interaction: Algorithm and Applications,” Proceedings of the seventh international conference on parallel problem solving from nature. (PPSN VII). Granada, Spain

[Zitzler99] Zitzler, E.(1999) ”Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications,” Ph. D. thesis, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland. TIK-Schriftenreihe Nr. 30, Diss ETH No. 13398, Shaker Verlag, Aachen, Germany.

[Zitzler01] Zitzler, E., Laumanns, M. and Thiele, L. (2001) ”SPEA2: Improving the Strength Pareto Evolutionary Algorithm,” TIK-Report 103. ETH Zentrum, Gloriastrasse 35, CH-8092 Zurich, Switxerland.