Attempt to Reduce the Computational Complexity in Multi-objective Differential Evolution Algorithms

Martin Drozdik, Hernan Aguirre and Kiyoshi Tanaka
Faculty of Engineering, Shinshu University
4-17-1 Wakasato, Nagano 380-8553, Japan
martin@iplab.shinshu-u.ac.jp, ahernan@shinshu-u.ac.jp, ktanaka@shinshu-u.ac.jp

ABSTRACT

Nondominated sorting and *diversity estimation* procedures are an essential part of many multiobjective optimization algorithms. In many cases these procedures are the computational bottleneck of the entire algorithm. We present the methods to decrease the cost of these procedures for multiobjective differential evolution (DE) algorithms. Our approach is to compute domination ranks and crowding distances for the population at the beginning of the algorithm and use a combination of well known data structures to efficiently update these attributes. Experiments show that the cost of improved nondominated sorting is sub-quadratic in the number of individuals. In practice using our methods the overall DE algorithm can run 2 to 100 times faster.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*; G.1.6 [**Numerical Analysis**]: Optimization—*Global optimization*

General Terms

Algorithms, Performance

Keywords

differential evolution, nondominated sorting, crowding distance, many-objective optimization, K-d tree, skip list

1. INTRODUCTION

Many multi-objective evolutionary optimization algorithms (MOEAs) rank individuals based on *Pareto optimality* and *diversity estimation* procedures. These two procedures are computationally expensive, especially for large populations and high dimensional objective spaces.

For example, we profiled the computational cost of GDE3[9], a differential evolution[10] MOEA, using a population size of 1000 individuals, optimizing a DTLZ4 problem[4] with 5 objectives and 20 variables. We found that 98% of the total computational cost is attributed to nondominated sorting and crowding distance, less than 1% to the objective function, and the rest were recombination and system activities.

The high computational cost of Pareto based methods has motivated research to find ways to reduce it.

The first improvement of the naive nondominated sorting method is due to Deb et al.[3] The fast nondominated sorting method improves the naive method by a factor roughly equal to the number of nondominated fronts in the population.

A sophisticated computation complexity reduction of the nondominated sorting has been achieved by Jensen[6]. Jensen also pointed out that the usage of more elaborate data structures to hold the population can be used to reduce the complexity of both nondominated sorting and diversity estimation. Other improvements in the computational complexity of diversity estimation have been achieved by Kukkonen and Deb, first for bi-objective problems [7], and then for any number of objectives [8].

All these complexity reductions can be applied to any evolutionary algorithm which uses nondominated sorting or diversity estimation. However, to our knowledge, there have not been published any results exploiting characteristics exclusive to differential evolution (DE) algorithms.

In this work, we improve the computational complexity of differential evolution MOEAs. We propose a strategy which exploits the survival selection properties of DE to decrease the complexity of nondominated sorting and diversity estimation. To this end we use a special data structure to manage the nondominated individuals. We test the proposed strategy on GDE3[9], showing that the improved DE algorithm can run 2 to 100 times faster.

2. MULTIOBJECTIVE DIFFERENTIAL EVOLUTION

2.1 Basic concept

DE is an evolutionary algorithm developed by Storn et al.[10] Although originally developed for single objective optimization, it can be easily extended to multiple number of objectives. Notable examples are by Kukkonen[9] and Robič and Filipič[12].

DE tries to improve the population one individual at a time. It does so by looping through the population in what we call a *generational loop* and creating a new *trial* individ-

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

ual[1] for each *target* individual in the population. Instead of waiting until all the trials are generated, their objective value is evaluated immediately and they are compared for domination with the target. The detailed procedure is described in Algorithm 1.

Algorithm 1: multiobjective DE

```
initialize population P = {X_1, ..., X_N};
for g := 1 to G_max do  Evolutionary loop
    for i := 1 to N do  Generational loop
        target := X_i ;
        trial := generate_trial(i) ;
        if target dominates trial then
            discard trial ;
        end
        if trial dominates target then
            replace target with trial ;
        end
        if mutually nondominated then
            add trial to the end of the population ;
        end
    end
    Trim the population to size N using nondominated
    sorting and diversity estimation ;
end
```

At the end of the *generational loop* the population size is between N and $2N$. The size is reduced back to N before proceeding to the next generation. This is done by the computationally expensive nondominated sorting and diversity estimation. We will build methods to dissipate the cost of these procedures throughout the *generational loop*.

2.2 Paradigm and notation

We improve DE algorithms conforming to the Algorithm 1. Also, for concepts pertaining to DE, we use the same notation as in [10] in the rest of the text. Furthermore we assume *minimization* at all times. N stands for the initial population size, M denotes the number of objectives, the objective functions will be called f_i, and X_i will denote a vector in $\mathbb{R}^n$ also called an *individual*.

As we can see in Algorithm 1, an important difference between DE and MOEAs such as NSGA-II[3] is that the selection process in DE is gradual and much of the selection happens intermittently with generation of new individuals, during the *generational loop*. We exploit this characteristic.

3. PROPOSED METHOD TO REDUCE THE COST OF NONDOMINATED SORTING

3.1 Motivation

When using evolutionary algorithms to solve problems with 3 and more objectives, there is a tendency for a large proportion of the population to become nondominated[1]. This phenomenon is informally called the *the curse of dimensionality*. This is quite a general notion, so in the rest of this paper we shall call the previously described phenomenon *overnondomination*.

[1]This is analog to the *offspring* in classical MOEA terminology.

Our idea is to *take advantage* of the *overnondomination*. When there are more than N nondominated individuals at the end of the *generational loop* we can pick the N individuals that will survive into the next generation from these individuals. Our method is to determine *right at the start of the algorithm* which individuals are nondominated and *update this information after each change to the population*.

This is a drastic deviation from the standard approach which is to compute the nondominated fronts at the end of each generation. We can do this only because DE alters population one individual at a time, so there are no sudden changes to the population. Thanks to the *overnondomination*, this approach can eliminate the need for nondominated sorting entirely.

In the cases when there are less than N nondominated individuals, that is when the *overnondomination* is not significant enough, we can fall back to fast nondominated sorting.

During the *generational loop* the population changes if and only if we *replace the target with trial* or *insert the trial*. Keeping track of the nondominated individuals in face of such a rapid fluctuation can get computationally expensive. In the next sections, we describe an efficient process to update our knowledge of the nondominated individuals.

3.2 Geometry of nondominated sorting

Let us look at the first of the two possible changes of the population, that is *replacing a target with a dominating trial* in Figure 1. Let us suppose that the target is nondominated. This is a realistic assumption, since almost all the individuals are nondominated throughout the run of the algorithm.

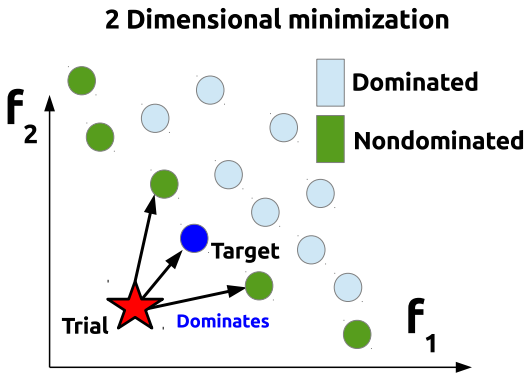

Figure 1: *Replacement* of Target

After the replacement, we need to update our knowledge of the nondominated individuals. Without any computation, we know that the trial is nondominated since it dominates a previously nondominated target. However, we still need to find out if some of the nondominated individuals have become dominated by the newly inserted trial.

In Figure 1 we see that the trial dominates other two individuals besides the target. We need to find these two individuals and update their status to "dominated". Checking with each nondominated individual is out of the question. We need to perform this check up to N times per generation and because of *overnondomination* there are relatively many nondominated individuals to compare with. This naive strategy would lead us back to the $O\left(N^2\right)$ complexity of the fast nondominated sorting.

Instead of doing domination comparisons, let us try to *construct* the set of the individuals dominated by the trial. In Figure 2 we see the area dominated by the trial individual.

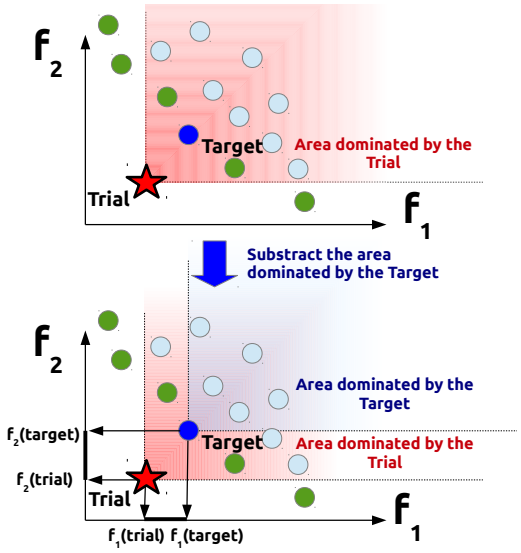

Figure 2: Finding individuals *dominated by* Trial

We need to find the individuals in this area. Since we are interested only in nondominated individuals we know that there are no individuals of interest in the set dominated by the target. Subtracting the portion of $\mathbb{R}^2$ dominated by the target leaves us only two narrow stripes to search. Mathematically, we just need to find the individuals X for which:

$$f_1(X) \in [f_1(trial); f_1(target)] \quad (1)$$

$$\vee$$

$$f_2(X) \in [f_2(trial); f_2(target)] \quad (2)$$

The second of the two possible changes of the population is the *insertion* of a trial individual without removing the target individual such as in Figure 3.

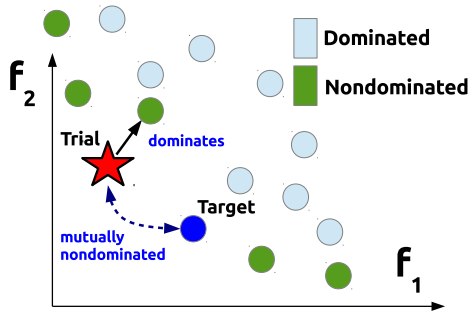

Figure 3: *Insertion* of nondominated Trial

In this case we are not sure if the trial is nondominated. Also, there is a possibility, that the trial dominates some previously nondominated individual. We need to investigate both possibilities.

Again, rather than checking with each nondominated individual, we *construct* the set of all the individuals which are dominated by the trial and the set of individuals which dominate the trial. We do this analogically by subtracting the areas which are *dominated by or dominating the* **target** from the areas which are *dominated by or dominating the* **trial** respectively. This is illustrated in Figure 4.

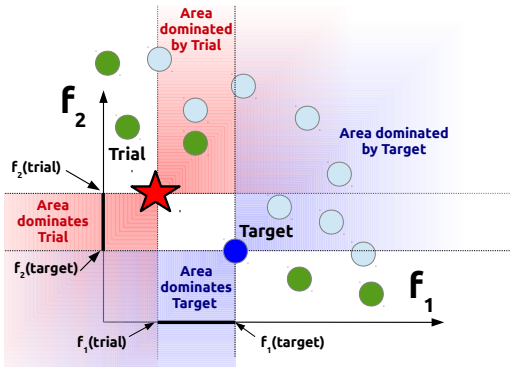

Figure 4: Finding individuals *dominated by* and *dominating* Trial

We can see that the sets of interest are restricted to *relatively small* parts of the objective space:

$$\text{Set of individuals which } \textit{are dominated by trial} \subseteq \{X | f_1(X) \in [f_1(trial); f_1(target)]\} \quad (3)$$

$$\text{Set of individuals which } \textit{dominate trial} \subseteq \{X | f_2(X) \in [f_2(target); f_2(trial)]\} \quad (4)$$

Instead of comparing the trial individual with each nondominated individual, we just compare with the individuals in (3). Then we determine the individuals in (4). If this set is empty, or if there have already been some individuals dominated by the trial, we know that the trial is nondominated. Otherwise we compare with individuals in (4).

3.3 Reference sets and reference individuals

We shall now proceed to generalize the ideas of the previous section. We were examining the changes to a nondominated population P when a new trial individual t has been introduced into the population either as a replacement for a target or merely a new member of the population. We were also using the concept of a target individual. However we did not assume about the target anything besides nondominatedness, so we shall generalize the notion of target individual to that of a reference individual.

We shall generalize our concepts to more than 2 dimensions.

DEFINITION 1. *Let $P = \{X_1, ..., X_N\}$ be a population, whose individuals are all mutually nondominated and let t be a newly created trial individual. Let r be an arbitrary individual from P.*

We define the upper reference set *for individual t induced by individual r, to be the set $R_U(t,r) \subseteq P$ given by:*

$$R_U(t,r) = \bigcup_{i=1}^{M} \{X_j \text{ such that } f_i(t) \geq f_i(X_j) \geq f_i(r)\} \quad (5)$$

Conversely, we define the lower reference set *for individual t induced by individual r, to be the set $R_L(t,r) \subseteq P$ given by:*

$$R_L(t,r) = \bigcup_{i=1}^{M} \{X_j \text{ such that } f_i(t) \leq f_i(X_j) \leq f_i(r)\} \tag{6}$$

This is a direct generalization of (3) and (4). In fact (3) is the lower reference set for the trial individual induced by the target $R_L(trial, target)$ and (4) is the upper reference set $R_U(trial, target)$. In Figure 1 the three individuals dominated by the trial and the target are the lower reference set $R_L(trial, target)$. The upper reference set contains only the target.

We shall call the individual r which induces the reference sets *reference individual*. The next section will pay more attention to the selection of the reference individual.

The next observation proves, that the upper and lower reference sets contain all the individuals that dominate or are dominated by the trial respectively.

OBSERVATION 1. *Let $P = \{X_1, ..., X_N\}$ be a population, whose individuals are all mutually nondominated and let $r \in P$ be an arbitrary individual. Let t be a newly created* trial *individual. Then if t dominates some $X_j \in P$ then $X_j \in R_L(t,r)$.*

In other words, all the individuals dominated by t are in $R_L(t,r)$

PROOF. We shall prove that:

$$t \text{ dominates some } X_j \in P \Rightarrow$$
$$\exists i \in \{1,\ldots,M\} \text{ such that}$$
$$f_i(t) < f_i(r) \wedge f_i(X_j) \in [f_i(t); f_i(r)]$$

which together with (6) implies that $X_j \in R_L(t,r)$.

We do a proof by contradiction. First let us prove that there is an i for which $f_i(t) < f_i(r)$ holds. Let us suppose that the negation is true: $\forall i \in \{1,\ldots,M\}\ f_i(t) \geq f_i(r)$. Since t dominates X_j, there is an index I for which $f_I(X_j) > f_I(t)$. That means:

$$\forall i; f_i(X_j) \geq f_i(t) \geq f_i(r) \wedge$$
$$\exists I; f_I(X_j) > f_I(t) \geq f_I(r)$$

This means that r dominates X_j which contradicts the assumption that all the individuals in P are mutually nondominated.

Let us now suppose that for all i for which $f_i(t) < f_i(r)$ holds, the negation of $f_i(X_j) \in [f_i(t); f_i(r)]$ is true. That is $f_i(X_j) < f_i(t) \vee f_i(X_j) > f_i(r)$. We know that $f_i(X_j) < f_i(t)$ cannot hold, since t dominates X_j. That means

$$f_i(t) < f_i(r) \Rightarrow f_i(X_j) > f_i(r) \tag{7}$$

For all the i for which $f_i(t) < f_i(r)$ does not hold, we have $f_i(X_j) \geq f_i(t) \geq f_i(r)$ since t dominates X_j. Combine this with (7) to get that r dominates X_j which is again a contradiction with the assumption of the theorem. □

Analogically we can prove the statement for *upper reference sets.*

OBSERVATION 2. *Let the assumptions of observation 1 hold. Then if t is dominated by some $X_j \in P$ then $X_j \in R_U(t,r)$.*

The purpose of the reference sets is to diminish the number of domination comparisons needed when we add a new individual into the population. Each time the population is changed, the new individual does not need to be compared to all the nondominated individuals for domination. We just compare with individuals in the reference sets.

4. IMPLEMENTATION

4.1 M-list

In this section we describe *how* to efficiently implement the method proposed in the previous section. In order to be able to construct the upper and lower reference sets quickly, we propose to keep the nondominated individuals in a special data structure. If we look at the definition of the lower reference set (6), we see that the reference set is a union of one dimensional *interval queries* of the population. That means we need a structure which supports quick queries of all the values which lie in a given interval.

One of many such structures is the skip list[11]. This structure answers such queries in time $O(\ln(n) + k)$ where n is the number of items in the list and k is the number of items to report. We need to be able to make queries with respect to any objective, so we actually need M skip lists. One for each objective. We call the structure consisting of M skip lists, each sorted by one objective and each holding exactly the nondominated individuals an *M-list.*

Figure 5 is an illustration of a 3-dimensional M-list holding 6 mutually nondominated individuals and a newly inserted trial. We want to know the individuals dominated by the trial. To do this we construct a lower reference set (6). In this case we choose the reference set induced by the target $R_L(trial, target)$

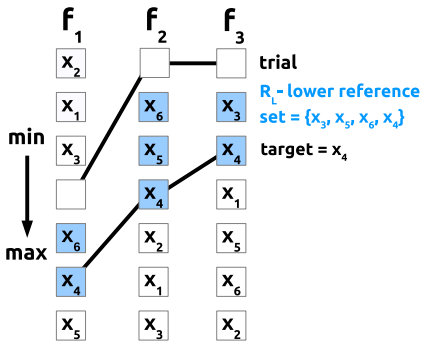

Figure 5: Insertion of a dominating trial into an M-list

To construct $R_L(trial, target)$, we gather all the individuals which lie between the trial and the target including the target. In this case it is the set $R_L = \{X_3, X_5, X_6\} \cup \{X_4\}$. By comparing each individual from R_L to the trial, we find that X_5 and X_6 are dominated by the trial. So we discard them and the target X_4 from the M-list.

When we insert a trial which is nondominated with the target, we need to determine *two* things:

1. Is the trial nondominated?

2. Are some currently nondominated individuals dominated by trial? [2]

We determine this by constructing both the upper and lower reference sets for the trial. The process is illustrated in figure 6. We see that the closer the trial and target are,

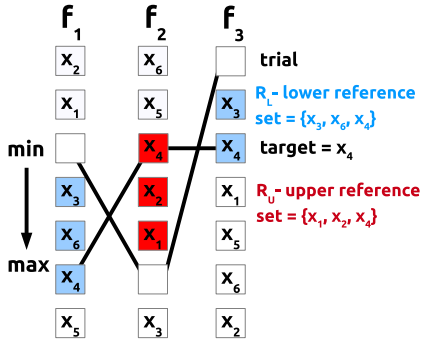

Figure 6: Construction of lower and upper reference sets when inserting a nondominated trial

the less individuals we need to compare. Therefore it is useful to choose the reference individual *as close as possible to the trial*. This leads us to the following topic.

4.2 Selection of the reference individual

Any nondominated individual can be chosen as the reference individual. But we would like the reference sets to be as small as possible. This happens when the reference individual is relatively close to the trial. Here we propose strategies to select the reference individual:

Target reference: If the crossover probability Cr is low, then the trial inherits almost all the parameters from target and therefore should be close to the target. In case the target is dominated and therefore not in the M-list, we suggest that reference individual is chosen from the M-list randomly.

Base reference: The target vector need not be likely close to the trial. This is especially true when the Cr is high. In this case, there is also an individual which is likely to be close to the trial. It is the *base* individual [10]. The individual to whom the scaled difference of two individuals is added.

Closest to trial: To compute the closest individual in M-list to the trial should result in a relatively small reference set. The downside is the computation cost, but some algorithms such as DEMO[12] already compute the closest individual, so it is worthwhile to use the result of this computation for this purpose too.

The properties of the various strategies are summarized in Table 1.

4.3 K-d tree

Since the update of the population happens N times per generation, the $O(N)$ time to compute the nearest neighbor is unacceptable. However the *nearest neighbor problem* is a heavily researched subject. Using simple data structures, we can solve the nearest neighbor problem in sub linear time

[2]Luckily $2 \Rightarrow 1$ which is equivalent to $\neg 1 \Rightarrow \neg 2$. So if we implement the program to check 1 and then 2, we do not need to check 2 in cases, where 1 is false and conversely, if we check 2 and then 1, each time 2 evaluates to true, we know that 1 is also true.

Table 1: Reference individual strategies summary

Strategy	Cost	Useful when
Target	$O(1)$	Cr small
Base	$O(1)$	Cr large F small
Closest to trial	$O(N)$	always

for dimensions up to approximately 8 and we can compute a very quick and a very accurate approximate nearest neighbor up to about 20 dimensions.

K-d tree[2] is one of many structures designed to manage vector data and offer fast nearest neighbor search. The properties of a K-d tree are summarized in Table 2.

Table 2: K-d tree properties

Operation	Average cost
Construction	$O(n \ln(n))$
Insertion	$O(\ln(n))$
Deletion	$O(\ln(n))$
Exact nearest neighbor	$O(\ln(n))$
Approximate nearest neighbor	$O(\ln(n))$

The problem is that the tree gets unbalanced after a large amount of insertions and deletions. Luckily, the tree is cheap to rebuild and the rebuilding is much faster, if we have the data sorted according to each dimension. That is the case with the M-list. It is therefore a good idea to manage both the K-d tree and M-list.

In our experiments we construct the K-d tree directly from the M-list once the cumulative number of insertions and deletions gets bigger than two times the size of the primary population N.

5. COMPUTATIONAL COMPLEXITY

5.1 Diversity estimation

The M-list is an appropriate structure to compute the crowding distance (CD). Since the population is already ordered with respect to all the objectives, the computation cost of CD equals to that of traversing the M-list which is $O(MN)$. This improves the original $O(MN \ln(N))$. The M-list is also very useful for algorithms which update the CD after they remove one individual[3] since this update is possible in $O(M)$.

5.2 Nondominated sorting

The computational complexity of known methods is summarized in Table 3. The operation, whose cost is measured is the comparison between two real numbers. Therefore the cost of dominance comparison is $O(M)$.

The computational complexity of our method depends very much on the quality of the reference individual. The closer it is to the trial individual, the smaller is the reference set and the smaller is the number of individuals that need to be compared.

First, let us derive the worst case complexity. To do that we just need to realize that our algorithm can make as many comparisons as the fast nondominated sorting. That is $O(MN^2)$. The overhead of updating an M-list does not

[3]Such as GDE3[9]

Table 3: Summary of computational complexities of nondominated sorting methods

	Deb	Jensen	Our method
worst	$O\left(MN^2\right)$	$O\left(N\ln^{M-1}(N)\right)$	$O\left(MN^2\right)$
average	$O\left(MN^2\right)$	$O\left(N\ln^{M-1}(N)\right)$	$O\left(M^2Ng(N)\right)$
best	$O\left(MN^2\right)$	$O\left(N\ln^{M-1}(N)\right)$	$O(MN)$

$M-$ number of objectives
$N-$ population size
$g(N)-$ average number of individuals between the trial and reference individual with respect to one objective. In the next section, g is estimated to be $g(N) \approx \alpha N^{\beta};\ \beta \in [0.6; 1]$

make a difference since insertions and deletions in the M-list are both $O(M\ln(N))$ operations. There can be only $O(N)$ updates which leads to only $O(MN\ln(N))$ cost.

In the best case, the reference individual is chosen so luckily, that the reference sets generated by it contain only the reference individual. In this case there is only one comparison for each reference set and therefore only $O(N)$ comparisons per *generational loop*. Multiplied by $O(M)$ we get a complexity of $O(MN)$.

To get the average case, we make only a rough "back of the envelope" estimate. Each new individual needs to be compared to the individuals in the upper and lower reference set. Let us suppose that on average there are $g(N)$ individuals between the trial and reference individual with respect to each objective. Further let us suppose, that the individuals that are between the trial and reference individual with respect to one objective are not between the trial and reference individual with respect to any other objective. We see in Figure 6 that the number of individuals in both upper and lower reference sets combined is $Mg(N)$. Multiplying this by $O(N)$ population updates we get $O(MNg(N))$ domination comparisons. That is $O\left(M^2Ng(N)\right)$ real number comparisons.

6. EXPERIMENTAL RESULTS

6.1 Common setup

In the entire section, we will be comparing the improved version of the GDE3[9] algorithm with the original one. We have implemented both algorithms using C++, following the description of the algorithm in [9].

For all experiments we have used exponential crossover[10]. The parameters which are same for all further experiments are summarized in Table 4. This used combination of parameters for GDE3 proved to be most universal after some calibration.

Table 4: Experimental setup

Problems	DTLZ1, WFG9
F mutation factor	0.2
C_r crossover probability	0.2
Number of generations G_{max}	1000
Number of variables	15
Number of runs	10

We were measuring three basic quantities:

Time elapsed for survival selection; We have computed the amount of time *not* spent on survival selection for each run of the original GDE3 algorithm. We named this quantity *baseline* and subtracted it from the total elapsed time of each corresponding run.

Cumulative number of domination comparisons; This is a quite implementation-independent statistic. In this article we described improvement of nondominated sorting, diversity estimation and nearest individual search. This metric will allow us to measure the isolated effect of nondominated sorting improvement.

Number of comparisons in generation 500; Our method exploits *overnondomination*. That is, its full strength is revealed once the majority of individuals is nondominated. At the start of the algorithm there are usually not many nondominated individuals, even for many-dimensional problems. We use this quantity to measure the peak performance of the proposed method.

Furthermore for each experiment for each statistic we shall mark in **bold** the *best* result.

6.2 Reference individual selection

First let us look at various strategies to select the reference individual. In Table 5 we provide the average results of 10 runs with the DTLZ1[4] problem with a population size of 1000 individuals. The **Defeatist** strategy means finding the approximate nearest neighbor according to the K-d tree. The **Euclidean**, **Manhattan** and **Maximum** mean choosing the reference individual to be the nearest neighbor to the trial with respect to the corresponding metric. **Target** means choosing the target vector as the reference individual. **Normal** means the original algorithm using fast nondominated sorting[3] and baseline means the time to execute all *non* selection-related activities.

Table 5: Comparison of various strategies

Time elapsed for survival selection (seconds)

objectives	3	4	5	6	7	8
Defeatist	**12.5**	**19.0**	**35.4**	65.8	101.2	143
Euclidean	18.6	27.8	48.1	79.9	119.7	149
Manhattan	15.4	27.0	53.2	89.3	131.5	161
Maximum	12.6	19.5	36.7	65.4	104.9	143
Target	24.3	35.9	50.5	**57.4**	**69.0**	**82**
Normal	107.3	163.4	259.7	333.8	423.8	444
baseline	3.8	3.6	3.8	3.8	3.7	3.0

Cumulative domination comparisons (millions)

Defeatist	254	194	182	238	333	429
Euclidean	253	189	173	207	273	333
Manhattan	**252**	**187**	**169**	**196**	**256**	**311**
Maximum	252	189	179	219	294	364
Target	333	294	277	292	341	395
Normal	1314	1463	1615	1778	1948	2130

Domination comparisons in generation 500 (thousands)

Defeatist	9	58	137	229	330	407
Euclidean	9	54	126	195	252	304
Manhattan	**8**	**50**	**121**	**184**	**237**	**286**
Maximum	8	52	130	206	273	337
Target	99	163	227	277	325	370
Normal	1309	1457	1602	1758	1921	2115

By looking at the baseline data, we see that selection for survival is indeed the bottleneck for all configurations.

We also see that even though the Manhattan strategy yields the smallest reference sets and hence the lowest number of cumulative domination comparisons, it is slower than the Defeatist strategy or the Target strategy. The cost of computing the exact nearest individual is too high.

Comparing the Defeatist and the Target strategies, we see that the Defeatist produces consistently smaller reference sets, resulting in less domination comparisons up until 7 objectives. However, the cost of maintaining the K-d tree weights the Defeatist strategy down at 6 objectives.

It is surprising that for each strategy, the number of cumulative comparisons *decreases* from 3 until 5 objectives. We found out that this was due to insufficient number of non-dominated individuals in the population. For 3 objectives it took 216 generations until there were more than N nondominated individuals. The algorithm had to fall back to fast nondominated sorting for the dominated part of the population. The *overnondomination* did not strike soon enough.

6.3 Small and medium populations

In Table 6, we compare the times of the most ambitious strategies from the previous experiments, that is Target and Defeatist. We examine population sizes from 100 to 1000. We present the data for the WFG9[5] problem. In order to save space, we present only the ratios of the results for the original and our proposed method.

Table 6: WFG9 Time elapsed for survival selection

Normal / Target

N	M = 3	4	5	6	7
100	**7.551**	**6.044**	**5.737**	**8.483**	**7.188**
200	**3.888**	4.218	**3.854**	**3.313**	**3.851**
300	8.320	5.189	4.159	**4.073**	**3.846**
500	**17.901**	**14.731**	**10.556**	**8.472**	**6.856**
1000	9.224	9.680	7.771	**7.576**	**9.222**

Normal / Defeatist

100	3.032	3.200	2.263	1.865	1.660
200	5.445	**4.823**	3.688	2.987	2.609
300	**10.819**	**6.752**	**4.600**	3.798	3.391
500	15.548	10.909	6.987	4.916	4.146
1000	**15.769**	**11.748**	**7.836**	6.434	5.130
β	1.23	1.50	1.57	1.60	1.73

The elapsed times for the Defeatist strategy can be reliably approximated by a power curve of type $y = \alpha N^{\beta}$ using least squares. The data for the Target strategy does not seem approximable by such a curve. For each number of objectives, we present the exponent β. We do this also for Table 8.

The Defeatist strategy performed well for low dimensionality and high number of individuals. This reflects the properties of the K-d tree. When we look at the cumulative domination comparisons ratios in Table 7, we see that the Defeatist strategy produced smaller reference sets in all configurations up to 6 objectives, but achieved better speed only in some of those configurations. This is attributed to the cost of the K-d tree.

For a fixed number of objectives, the ratio increases with increasing population. Unfortunately, for increasing number of objectives, the ratio decreases.

Table 7: WFG9 Cumulative domination comparisons

Normal / Target

N	M = 3	4	5	6	7
100	12.23	9.05	7.28	6.55	**6.02**
200	13.30	12.07	8.66	7.09	**6.12**
300	15.84	12.20	8.73	7.14	**6.04**
500	16.17	12.47	8.89	7.18	**6.20**
1000	15.62	12.56	9.04	7.25	**6.18**

Normal / Defeatist

100	**44.04**	**16.56**	**9.61**	**6.74**	5.46
200	**53.34**	**21.46**	**10.77**	**7.26**	5.63
300	**60.58**	**23.06**	**11.32**	**7.51**	5.67
500	**63.09**	**25.42**	**12.11**	**7.91**	5.88
1000	**62.16**	**28.91**	**13.59**	**8.55**	6.16

Table 8: WFG9 Domination comparisons in generation 500

Normal / Target

N	M = 3	4	5	6	7
100	12.458	8.866	7.437	6.798	**6.160**
200	13.946	12.312	9.295	7.349	**6.415**
300	17.651	12.433	9.123	7.434	**6.368**
500	17.978	12.731	8.993	7.127	**6.299**
1000	18.132	13.354	9.288	7.365	**6.360**
β	1.83	1.85	1.94	1.99	2.006

Normal / Defeatist

100	**48.695**	**17.195**	**10.077**	**6.436**	5.690
200	**64.091**	**22.810**	**11.336**	**7.585**	5.846
300	**82.248**	**24.656**	**11.704**	**7.616**	5.791
500	**94.016**	**26.712**	**12.303**	**7.899**	6.003
1000	**121.496**	**31.034**	**14.039**	**8.712**	6.345
β	1.62	1.77	1.88	1.90	1.97

The estimates of β in Table 8 shed some light on the unknown function g from section 5.2. αN^{β} is an estimate of the number of domination comparisons in one generation, which was theoretically estimated to be $MNg(N)$. This means that $g(N) \approx \gamma N^{\beta-1}$.

We have run the same configurations with the DTLZ1 problem and the results were very similar to that of WFG9, with the exception that the Target strategy was always sub quadratic.

6.4 10000 individuals

We conclude our experiments section with results for 10000 individuals. Because of the extreme time cost of computing such experiments (see Tables 9 and 10) each run was completed only once. Let us look at the DTLZ1 problem.

Again we measure time ratios and for illustration in the last row you can find the time it takes to complete the run on a desktop computer. The Manhattan strategy is the fastest for 3 and 4 objectives. The dimensionality is low enough to allow the K-d tree to find the nearest individual

very quickly. For higher dimensions the cost of finding exact nearest neighbor outweighs the benefits gained from finding a quality reference individual.

Table 9: DTLZ1 Normal / Improved

Time elapsed

M	3	4	5	6	7
Maximum	11.63	14.15	10.79	8.93	7.00
Target	3.41	3.87	3.91	4.21	6.08
Manhattan	**21.81**	**20.59**	9.66	7.30	6.00
Defeatist	11.05	14.06	**11.40**	**9.95**	**7.97**
Normal(hours)	12.7	17.9	24.8	34.3	44.3

Domination comparisons in generation 500

Maximum	337.80	44.42	14.39	7.82	5.73
Target	12.75	9.50	6.75	5.35	4.97
Manhattan	**338.17**	**45.92**	**15.21**	**8.43**	**6.28**
Defeatist	294.34	41.33	14.78	8.36	6.16

Using fast nearest neighbor strategies results in abysmally smaller reference sets for small number of objectives.

The results for WFG9 seem to be very similar to those for DTLZ1. The gain from generating small reference sets outweighs the cost of maintaining a K-d tree.

Table 10: WFG9 Normal / Improved

Time elapsed

M	3	4	5	6	7
Target	8.62	9.31	12.56	10.03	6.58
Defeatist	**134.9**	**75.98**	**49.94**	**17.50**	**9.65**
Normal(hours)	37.9	48.5	89	95.7	80.6

Domination comparisons in generation 500

Target	18.17	14.03	10.02	7.74	6.28
Defeatist	**313.1**	**54.4**	**20.8**	**11.2**	**7.7**

7. CONCLUSION

We have provided methods to improve the *nondominated sorting* and *diversity estimation* procedures using the otherwise dreaded *overnondomination* to our advantage.

We used a special data structure to hold the nondominated individuals and rather than performing nondominated sorting, we concentrated on efficiently updating this structure. This led us surprisingly to the nearest neighbor search problem. We handled this using various approaches based on a well known data structure (K-d tree).

Our results show that a speedup of about 3 times is easily achievable in most cases. For big populations a more than 100 fold increase in speed is possible.

The complexity of our nondominated sorting seems to be sub quadratic in the number of individuals and quadratic in the number of objectives. Our algorithm outperforms the fast nondominated sorting even for small populations and high number of objectives, which is not the case for Jensen's method. However it can be used only with DE algorithms or algorithms which change the population one individual at a time. Our method takes advantage of the *overnondomination* and therefore is intended for problems susceptible to this phenomenon (such as problems with 3 or more objectives).

There is space for future work in the exploration of possibilities to generalize our method to algorithms other than DE. Also, there may be data structures for the approximate and exact nearest problem, which are more suitable than the K-d tree for the problem at hand.

8. REFERENCES

[1] M. Farina and P. Amato. On the optimal solution definition for many-criteria optimization problems. In *Fuzzy Information Processing Society, 2002. Proceedings. NAFIPS. 2002 Annual Meeting of the North American*, pages 233–238, 2002.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm NSGA-II. *IEEE Transactions on evolutionary computation*, 6(2), 2002.

[4] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multi-objective optimization. In *Evolutionary Multiobjective Optimization*, pages 105–145. Springer Berlin Heidelberg, 2005.

[5] S. Huband, P. Hingston, L. Barone, and L. While. A review of multiobjective test problems and a scalable test problem toolkit. *Trans. Evol. Comp*, 10(5):477–506, oct 2006.

[6] M. T. Jensen. Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. *IEEE Transactions on Evolutionary Computation*, 7(5):503 – 515, 2003.

[7] S. Kukkonen and K. Deb. Improved pruning of non-dominated solutions based on crowding distance for bi-objectve optimization problems. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 1179–1186. IEEE Press, 2005.

[8] S. Kukkonen and K. Deb. A fast and effective method for pruning of non-dominated solutions in many-objective problems. In *Proc. Parallel Problem Solving from Nature - PPSN IX*, volume 4193, chapter LNCS, pages 553–562. Springer Berlin Heidelberg, 2006.

[9] S. Kukkonen and J. Lampinen. GDE3: the third evolution step of generalized differential evolution. In *IEEE Congress on Evolutionary Computation*, pages 443–450, 2005.

[10] K. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution - A Practical Approach to Global Optimization*. Springer, Berlin, Germany, 2005.

[11] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, jun 1990.

[12] T. Robič and B. Filipič. DEMO: Differential evolution for multiobjective optimization. *Proc. Intl. conf. on Evolutionary Multi-criterion optimization (EMO 2005), Springer, LNCS 3410*, pages 520–533, 2005.