The Effect of Extreme Solutions Injection in Opposition-based Evolutionary Multi-objective Optimization

COIN Report Number 2016010

AKM Khaled Ahsan Talukder
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824, USA
talukde1@msu.edu

Kalyanmoy Deb
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48824, USA
kdeb@egr.msu.edu

Shahryar Rahnamayan
Department of Electrical, Computer and Software Engineering
University of Ontario Institute of Technology
Oshawa, Canada
shahryar.rahnamayan@uoit.ca

ABSTRACT

For more than two decades, stand-alone evolutionary multi-objective optimization (EMO) methods have been adequately demonstrated to find a set of trade-off solutions near Pareto-front for various multi-objective optimization problems. Despite a long-standing existence of classical generative single-objective based methods, a very few EMO studies have combined the two approaches for a better gain. In this paper, we investigate the effect of seeding the initial population of an EMO algorithm with extreme solutions obtained using a single-objective method. Our proposed approach is further aided with an opposition based offspring creation mechanism which strategically places new solutions on the current Pareto frontier by a simple, yet a novel arbitration policy that utilizes the relative distances from the extreme solutions in the current population members. We conduct an extensive simulation of our proposed approach on a wide variety of two and three-objective benchmark MOP test problems. Results are shown to be remarkably better than the original EMO approach in terms of hyper-volume metric. The results are interesting and should motivate EMO researchers to integrate single-objective focused optimization and an opposition-based concept with diversity-preserving EMO procedures for an overall better performance.

Keywords

Single objective Search, Multiobjective Optimization, Opposition Based Learning (OBL)

1. INTRODUCTION

In the recent years, the idea of Opposition Based Learning (OBL) has been enjoying a noticeable attention among the AI and OR practitioners. The idea of OBL is to accelerate the learning rate (or convergence rate) by imposing an opposite estimate of the current solution, and deliberately introducing them to influence the search trajectory. This idea was first introduced in [10] and has been demonstrated its effectiveness in different scenarios – from Reinforcement Learning (RL) [11], Differential Evolution (DE) [8] to robotics [6].

In all of the cases, the OBL comes first into play during the initialization phase of the learning algorithm (or optimization algorithm), the argument behind this strategy is that a completely arbitrary (i.e. blind) initialization is no better than an informed boot-strap. Since a random sampling or selection of solutions from a given population has the possibility of visiting or even revisiting unpromising regions of the search space. In [8], it has also been demonstrated that the chance of such revisits are lower in the case of opposite than it is for purely random initialization. There are also several formal proofs that show that, in general, the opposite estimations are more likely to be closer to the optimal solution than the completely arbitrary ones, a more details of a formal probabilistic analysis of opposition based learning can also be found in [5].

Moreover, especially for the case of optimization algorithms, it has been demonstrated that keeping a small ratio of candidate solutions with the opposite estimate help to converge faster [8] [12] – and most of the recent studies are inspired by this approach. For example, in [7] similar strategy has been used in the case of MOEA/D [16].Another relevant study can be found in [13], where the opposition based initialization has been tested with the Particle Swarm Optimization (PSO) scenario to induce a better convergence. Given all of these studies, still we could not find a good example of this idea applied to the Evolutionary Multi-objective Optimization (EMO) cases – more importantly – in a more meaningful and precise way. In this study we will try to fill this gap in an interesting, and more importantly – in a simpler way.

In our approach, we will address this idea of *opposition* in a different perspective, we will classify[1] the solutions with respect to some *desired traits* that we will like to have. Then we will use this information to deterministically[2] generate new solutions in a most viable locations in the search space – and this operation will be dictated by our special notion of opposition. In fact, our approach is somewhat similar in spirit to the previous studies done in [9] and [17]. Except the fact that our approach is extremely simple and does not assume any special property on the underlying search space, moreover, our approach does not build a computationally expensive models as done in [1].

This paper is organized as follows – first, we will discuss

[1]not to be confused with the term "classification" as in the machine learning domain

[2]Here, our method is not "deterministic" in a general sense, the stochasticity incurred by the candidate generation is not equivalent to the generic mutation operation.

why this idea of OBL needs to be reinterpreted for the Multi-objective Optimization (MOP) setting. Next we discuss one interesting limitation that most EMO algorithms suffer from – the *search trajectory bias*, and also give some argument on why such hindrance becomes inevitable. Then we will discuss how the idea of OBL can come into rescue. In our model, we generate the opposite points in a strictly deterministic way, by carrying out the arbitration of *opposition* in a different (and a more MOP relevant manner). To do this, we utilize the extreme solutions on the true Pareto-front (PF), and the next section discusses how to find them efficiently. After that we will describe our main algorithm in details. Then we will show our experiments with different benchmark MOP problem sets.

2. AN ALTERNATIVE INTERPRETATION OF OPPOSITION

As we have already discussed in the previous section, in most of the studies, the idea of *opposition* is employed as an incorporation of new candidate solutions with a certain kind of *opposite traits* into the existing population. Such traits could be interpreted in terms of different problem domain perspectives. For example, the *opposite* solutions could be – i) the ones with complete opposite representation from the current best individual, ii) the ones with the opposite estimates from the other spectrum of the variable bounds (i.e. in the case of real valued optimization). However, the injection of the opposite solutions could cause a re-route from the continuing search trajectory and thus could be a misleading operation – in a sense that the opposite candidate solutions might be useful given that the search space follows a desired pattern.

For example, if the task is to solve the N-queen problem, then the opposite representation of the current best can result into another valid global optima. One can easily verify this by computing the reverse assignment of queens from a existing optimal solution; and it will eventually take us to another global optima. This observation also demonstrates that the search space is multi-modal, following from the fact that the reverse representation of the current best solution needs to be an optimum for another peak of the search space. Therefore, most of the standard opposition based algorithms inject the opposite solutions during the optimization start-up; or maintain a constant (generally low) ratio of opposite points throughout the run. Therefore, the standard opposite injection scheme (most likely) to be effective given the two assumptions on the underlying search space are valid – multi-modality and the solution symmetry. For this reason, what we think – such approach is quite unwieldy to directly incorporate into a numerical optimization problems, e.g. multi-objective optimization problems (MOPs).

Moreover, in the case of black-box optimization scenarios, we hardly have a room to make any assumption about the multi-modality of the search space and/or the symmetry of the solution representation (i.e. unlike such assumptions could be made about the N-queen problem). Therefore, in this paper, we have revised the notion of opposition in terms of the preference criteria imposed on a solution. For example, most EMO algorithms aim to maximize two principal properties – i) the convergence and ii) the diversity, since the quality of a MOP solution depends on these two factors. Therefore, let us re-consider the opposite point generation/injection from a different perspective –

- Opposite Convergence: A solution *far* from the true Pareto-front is *opposite* to any solution that is *closer* to the true Pareto-front.
- Opposite Diversity: An *isolated* solution on the true Pareto-front is *opposite* to a *crowded* solution.

By taking the above two principles into account, we will deterministically generate opposite candidate solutions during the search. Obviously, the deterministic candidate generation scheme will only consider an opposite trait that is *good*. In the next section we will see, how the existing EMO algorithms shows the limitations in maintaining this two opposite traits during the search (i.e., solution generation) process.

3. LIMITATIONS OF CANONICAL EMOs: THE SEARCH TRAJECTORY BIAS

Most of the standard EMO algorithms (e.g. NSGA-II, SPEA-II [19] etc.), are elitist by design. They are also "opportunistic" in a sense that the population always try to converge to a particular portion of the Pareto-front (PF) which seems to be easier to solve at that moment. Therefore, they show preferences over a certain objective function which needs less exploration than the other. We can see such bias in the search when we try to solve the ZDT4 problem using NSGA-II. In this case, the first objective is easier to optimize than the second one, the readers can verify this fact from the Figure 1. Therefore, the search trajectory deliberately accumulates more points over the first objective to optimize one particular portion of the Pareto-front. Moreover, while putting more solutions to the vicinity of one particular objective axis, the search trajectory looses the uniformity by forming a crowded streak of points along that axis; on the other hand we can see that there is almost no solution on the other spectrum of the objective space. This kind of non-symmetric search behaviour is, we think, causes a hindrance to the optimization procedure. Therefore, it would be helpful if we could selectively inject points

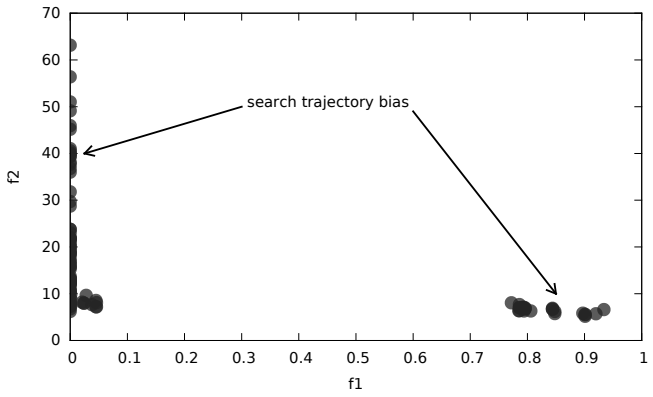


Figure 1: The effect of the *search trajectory bias* could be seen when we try to solve ZDT4 problem with NSGA-II (at generation 46, population size 100). Here we can see a long streak of crowded solution near the objective f_2, where the distribution of solutions near the objective f_1 is extremely sparse.

during the search where the solution distribution is more sparse. In addition, we also believe that this biased nature of the search trajectory could degrade with the addition of more objective functions. Moreover, this could also lead to a stagnation on the local optima, given that the search space has many of them.

Given this specific scenario, now the main problem is to devise a way to deterministically generate points where the distribution of the solution is sparse. Here we assume that the lesser the number of candidate solutions in a vicinity of objective f_i, the harder it is to solve. This would be easy if we know the exact mapping of the design variable to objective values – however such mapping is always unavailable and above all it is very expensive to infer. Another way could be to mutate the points where the solution is more sparse, but we think as the original algorithm already goes through such step, it is not going to be very effective.

In this paper, we are going to demonstrate a very effective approach to address the above mentioned issue, we will show how we can maintain a balanced distribution of solutions that is parallel to the true PF. The proposed approach is also extremely effective in fast converging to the true PF as well.

4. THE DETERMINISTIC OPPOSITE SOLUTION GENERATION SCHEME

As we have discussed in the Section 2, we will utilize the so-called notion of *opposition* to deterministically generate points into strategically useful places on the search space. In principle, we do not assume any exact mapping over the design variables to objective values, and we apply a linear translation to achieve our goal effectively[3]. In essence, instead of considering *sparsity* and *crowd* as two opposite traits, our method can impose any criteria like *being 'x'* and *being 'y'* as two opposites. Being said that, linear translation is a simplest way to deterministically generate *'x' like* solutions from *'y' like* ones.

So, the initial step is to infer the true PF before starting the actual optimization run. To do this, we depend on barely k number (k = number of objectives) of extreme (or near extreme) points on the true PF since we can safely assume that the population will eventually reach to the vicinity of those extreme points in the end. Moreover, the extreme points will be used as a pivot to arbitrate between the opposite traits over the existing solutions. Also note that we are not going to explicitly define which portion of the search space is less easy (or hard) and so forth – we will try to devise a technique that will implicitly address and solve such issues on the fly.

Another reason to fixate over the k extreme points is that we also wanted to keep the algorithm simple so that it can only utilize the "minimal information" of the true PF. We also think it's valid to assume that any PF could be bounded by at least k-extreme points for any k-objective problem. Although, if we could supply other intermediary points on the true PF, we will be able to see a better performance gain with the existing model, however supply of 1 extra true PF solution comes with an added cost of extra solution evaluations. As the extreme (or near extreme) points are the pivot to define the notion of *opposite* in our case, we will start the next section by discussing how to find them efficiently.

[3]As a matter of fact, we have also noticed that such opposite points are generally $\sim 30\%$ useful in most cases and they are more effective during the initial generations – which is also a very interesting finding.

Algorithm 1 Find Extreme Points

```
k ← no. of objectives
N_p ← population size
N_gen ← maximum generation
T ← (1/k)((1/4) N_p N_gen)
E* ← ∅, an empty solution set
for i from 1 to k do
    f_i ← i-th objective function
    x_i ← random initial vector
    repeat
        x_i ← solve f_i with IP method (or MADS)
    until T/2 solution evaluation reached
    f_aasf ← construct AASF function from f_i
    repeat
        x_i ← solve f_aasf with IP method (or MADS)
    until T/2 solution evaluation reached
    E* ← {E* ∪ x_i}
end for
return E*
```

4.1 Finding the Extreme Solutions

Extreme points on the Pareto-front could be found using global search as well [3], however our goal was to save the extra computational cost as much as possible[4]. Therefore, we resort to the classical single-objective optimization methods to solve this problem. Our choice of such algorithms were limited to, namely, the Interior Point (IP) method and the Mesh Adaptive Direct Search (MADS)[5]. As we also did not want to spend the valuable solution evaluations for this purpose, we have conducted this extreme solution search as a fixed budget operation. Depending on the difficulty of the problems, appropriate routine parameters were empirically found out which are problem dependent. These settings can be summarized as follows:

- If the problem has no local optima then we use IP method (i.e. `fmincon()`), it has been found to be comparatively less expensive even if the variable size is large.
- If the problem has local optima, MADS (i.e. `patternsearch()`) is faster for finding more accurate extreme points. However, for the DTLZ [4] problem set, these settings are found to be more useful:
  - `InitialMeshSize:` 1/population size
  - `Search Method: @MADSPositiveBasis2N` to start searching with $2N$ random directions, where N = number of variables.
  - and keeping both `CompletePoll` and `CompleteSearch` to `on`

The actual extreme point computation algorithm was conducted in two steps – given a particular objective function

[4]The readers might be aware that efficiently finding the extreme points on the true Pareto-front is itself a separate research problem [3].

[5]We have used the `fmincon()` and the `patternsearch()` routine in MATLAB (v. R2014a) for IP method and MADS respectively.

f_i, first we try to solve it directly using either IPM or MADS (depending on the problem type); then after some $\frac{T}{2}$ iterations once we find a reference solution $\mathbf{z}$ that is hopefully close to f_i's optima, then we construct a so-called Augmented Achievement Scalarizing Function (AASF) [15] from f_i as $f_{\text{aasf}} = \max_{j=1}^{k} w_j(f_j(\mathbf{x}) - z_j) + \rho \sum_{j=1}^{k} w_j(f_j(\mathbf{x}) - z_j)$ and solve it again for $\frac{T}{2}$ iterations. Here, we set $w_i = 0.9$, $w_{j\neq i} = \frac{1}{10(k-1)}$ and $\rho = 0.0001$.

To limit the solution evaluations, we kept T to a constant value (as a budget). For all problems, we have fixed this maximum iteration count to the $\frac{1}{4}$-th of the total generation specified. More precisely, $T = \frac{1}{k}(\frac{1}{4}N_p N_{\text{gen}})$, where k = no. of objectives, N_p = population size and N_{gen} = maximum generation. A basic listing for this routine is presented in Algorithm 1. The set of the extreme points E^* generated from this algorithm may not contain all the unique solution, and also they might not be the true extreme always, they can be weakly dominated solutions by the true PF extremes. However, our approach can utilize them efficiently to converge to the true PF extremes.

4.2 Generating the Opposite Solutions

Once the extreme points are discovered, now we utilize them to generate the so-called *opposite* points during the main evolutionary runs. On each generation, we select 25% of the best individuals (front-wise) from the current population and deterministically change them to generate opposite solutions – in such a way that we can address the strategically preferable places. And to conduct this variation, we will utilize the points in the set E^* as pivot points. We call these points as "pivot" since we will selectively try to generate points around these pivots. However, before doing this, we will *refine* our pivot points E^* in a certain way.

The *refinement* starts by finding the current population extreme points E_c and merging them with the set E^* such that $E = \{E_c \cup E^*\}$. Next we apply the non-dominated sort on E to find the Pareto-front within this set. We apply this sorting because we are not still sure if E^* contains true PF extremes. This sorting will keep the true extreme points if ones are found in the later generations. After this step, we select the points from E that are on the best front and with ∞ crowding distances[6]. Lets denote these selected points as E'. Now at this point, two situations are possible:

- The set E' contains only the solutions E^* while we are in the initial generations, or
- The set E' contains the solutions E_c while we are in the later phase of the generations, where E_c are the true PF extreme.

However, during the intermediate generations, it can also happen that we may include some solutions into E that weakly dominate a subset of points already in E, this inclusion will reduce the expected spread of the pivot points – that may diminish the effect of maintaining the diversity. For example, if the actual true PF is a broken Pareto-front, and if E contains the extreme points from one broken edge, then we need to expand the current edges so that the refinement procedure can include points from the further extreme ends. Therefore, if there exist a point in $E - E'$ that is on the best front and also weakly dominated by any point in E', then we replace the weakly dominating point from E' with the one from the set $E - E'$. The readers might have already noticed that $|E'| \leq |E|$.

[6]By "crowding distance", we mean the inter-solution distances computed in NSGA-II.

Algorithm 2 Generate Pivot Points

Require: true PF extreme points E^* from Algorithm 1
1: $E_c \leftarrow$ the extreme points from the current PF
2: $E \leftarrow \{E^* \cup E_c\}$
3: rank points in E, $E \rightarrow \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n\}$
4: take the best front in E', $E' \leftarrow \mathcal{F}_1$
5: **for** all points p_i in $E - E'$ **do**
6: **if** p_i weakly dominates any $p_j \in E'$ **then**
7: replace p_j by p_i
8: **end if**
9: **end for**
10: update E^*, $E^* \leftarrow E'$
11: $G \leftarrow$ find k intermediary gaps from the current PF
12: $E' \leftarrow \{E' \cup G\}$
13: **return** E'

Now, at this point, we can ensure that the set E' contains either true PF extremes or points near them. Now if we can generate new points near E', they will induce both better convergence and diversity. In section 3, we have discussed a scenario where we can see how the bias in the search trajectory is introduced. However, the difference in the relative difficulty of the objective functions may not be the only reason for a bias. The imbalance in the solution distribution could happen for other reasons as well. For example, a disconnected Pareto-front, a local optimal front or a specific portion of the Pareto-front being more difficult to solve than the rest. In such cases, we can see a *gap* forming over the Pareto-front during the search, we can see such a convergence pattern in many problems. As an example, we can see similar effect in solving ZDT4 problem as illustrated in Figure 2. To address these *gaps*, we also find the solutions with k-highest (k = no. of objectives) crowding distance from the best front that are not ∞, and call them as set G. Clearly, the G solutions are those that reside on the edge of the broken front. Now we add the G to the set E', thus we make E' ($|E'| > k$) as the final "pivot" solutions to generate the opposite points (see Algorithm 2).

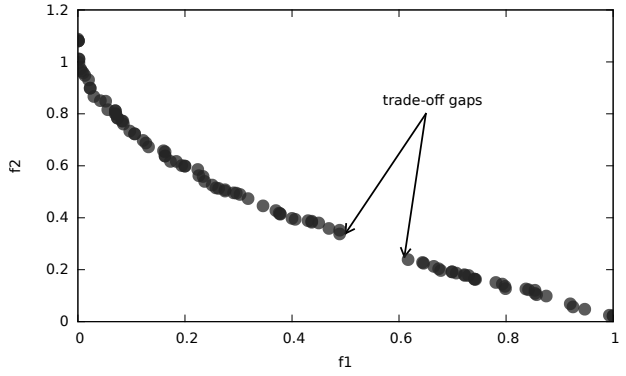


Figure 2: The effect of the *gap* in the trade-off, could be seen when we try to solve ZDT4 problem using NSGA-II.

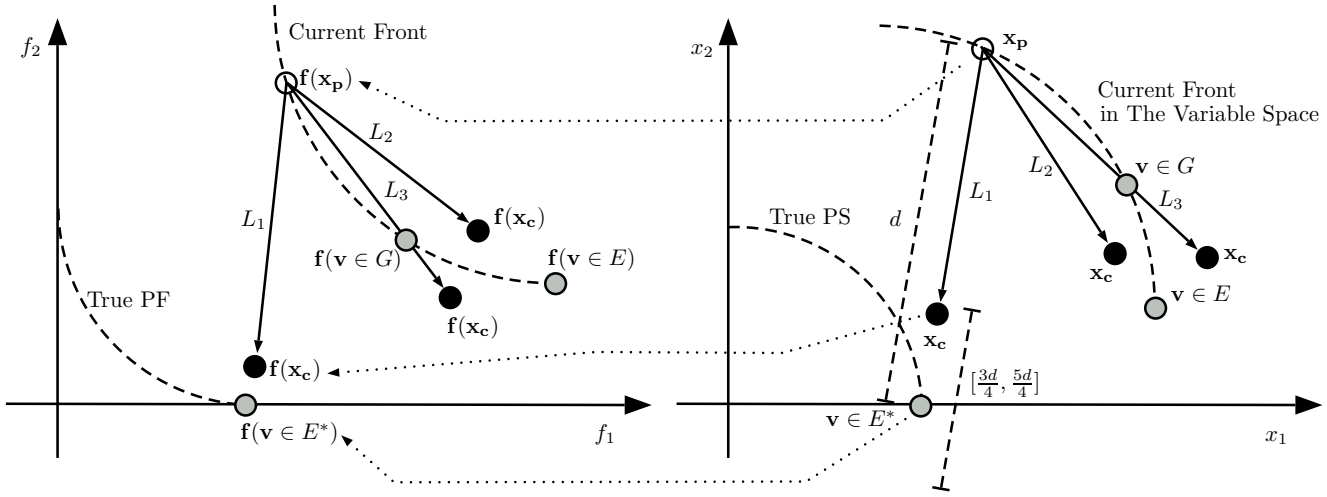


Figure 3: The illustration of lines 9–12 in Algorithm 3. The right axes are the variable space and the left axes are the corresponding objective space. The point $\mathbf{x_c}$ is the child (black circle) and $\mathbf{x_p}$ is the parent (white circle). The point $\mathbf{v}$ are the pivot points (grey circles). The operation will choose one of the directions denoted by L_1, L_2 or L_3. If $\mathbf{x_c}$ violates the variable bound then it is reverted back to the vicinity of the corresponding pivot point $\mathbf{v}$.

4.3 Integrating into an Elitist EMO Algorithm: NSGA-II

As we have mentioned at the beginning that we select front-wise best 25% of the current population for opposite point generation. We go through each of them randomly and every time we pick k number of random points from E' and pick the pivot point that is the furthest from it, and find the opposite vector using a linear translation. A straight-forward way – given a pivot vector $\mathbf{v}$ and a parent vector $\mathbf{x_p}$, we generate an opposite child $\mathbf{x_c}$ as $\mathbf{x_c} = \mathbf{x_p} + \mathbf{U}[(\frac{3d}{4}, \frac{5d}{4})] \circ (\frac{1}{d}(\mathbf{x_p} - \mathbf{v}))$. Here, $d = ||\mathbf{v} - \mathbf{x_p}||$ and $\mathbf{U}[(d, u)]$ is a uniform random vector where each element is within the range $[d, u]$. The overall procedure is presented in Algorithm 3 in line 9–12 and illustrated in the Figure 3. The lines 9–12 in Algorithm 3 can be recapped as follows: $\mathbf{v}$ is on the true PF extreme and $\mathbf{x_i}$ is *far* from $\mathbf{v}$, therefore, move $\mathbf{x_i}$ closer to $\mathbf{v}$ – *opposite* of *far* is *close*. Similar interpretation can be made when the vector $\mathbf{v}$ is an intermediary *gap*.

Moreover, upon generating the vector $\mathbf{x_c}$, it may happen that one of the variable values go beyond the variable bounds (i.e. $x_j > x_{j_H}$ or $x_j < x_{j_L}$), in that case, we replace the overshot value from the corresponding variable value v_j from the pivot point $\mathbf{v}$. Therefore, if a certain vector $\mathbf{x_c}$ can't make a successful translation, then $\mathbf{x_c}$ is reverted back to the vicinity of $\mathbf{v}$. Thus, we assure a local best estimated translation of the parent vector $\mathbf{x_p}$. This process is done on the line 13 of the Algorithm 3.

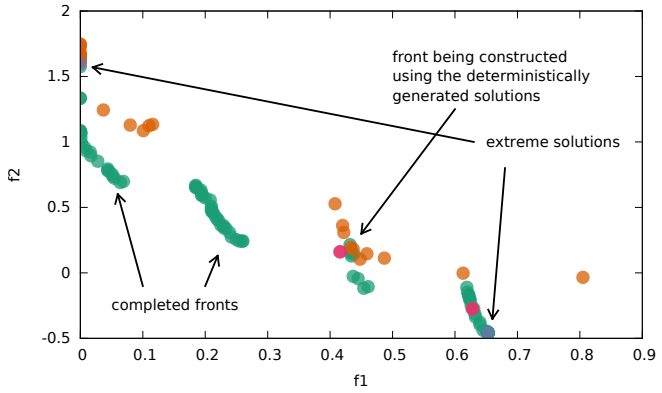
When we apply this algorithm to NSGA-II, we follow the obvious way, the generated opposite population will be inserted into the child population Q_t, the Algorithm 3 also shows how to integrate everything in NSGA-II. Moreover, this algorithm is "pluggable" in a sense that we can integrate it to any other elitist EMO algorithm. In the following section, we are going to see in details, how our opposite generation algorithm drastically improves the convergence rate. The readers should be aware that the 25% allocation was found by empirical experiments, and also to be better for most of the test problems.

5. EXPERIMENTS AND RESULTS

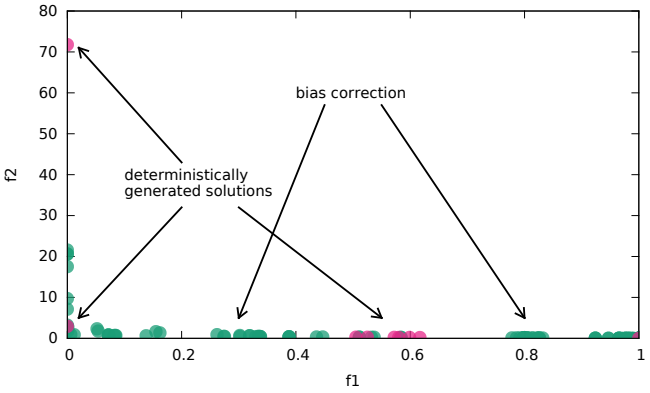
First we have tested the performance of Algorithm 3 on five 2-objective problems [18], namely ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6, and we have set NSGA-II as the control. To maintain a fair comparison, we have compensated the extra solution evaluations by the Algorithm 1 for the NSGA-II runs, and compared NSGA-II and Algorithm 3

Algorithm 3 NSGA-II with Opposition

Require: true PF extreme points E^* from Algorithm 1
1: $N \leftarrow$ population size $|P_t|$
2: $N_{\text{gen}} \leftarrow$ maximum generation
3: $t \leftarrow 1$
4: **while** $t \leq N_{\text{gen}}$ **do**
5: $P'_t \leftarrow$ front-wise best 25% from P_t and shuffle
6: $E'_t \leftarrow$ construct pivot set E' using algorithm 2
7: $O_t \leftarrow \emptyset$
8: **for** each solution $\mathbf{x_i} \in P'_t$ **do**
9: $S \leftarrow$ pick k random solutions from E'_t
10: $\mathbf{v} \in S$ such that $\mathbf{v}$ is the furthest point from $\mathbf{x_i}$
11: $d \leftarrow ||\mathbf{v} - \mathbf{x_i}||$
12: $\mathbf{x_c} \leftarrow \mathbf{x_i} + \mathbf{U}[(\frac{3d}{4}, \frac{5d}{4})] \circ (\frac{1}{d}(\mathbf{x_i} - \mathbf{v}))$
13: $x_j \leftarrow v_j \in \mathbf{v}$ if $x_j \in \mathbf{x_c} > x_{j_H}$ or $x_j \in \mathbf{x_c} < x_{j_L}$
14: $O_t \leftarrow \{O_t \cup \mathbf{x_c}\}$
15: **end for**
16: $P_t \leftarrow \{P_t \cup E^*\}$
17: $R_t \leftarrow \{P_t \cup Q_t\}$
18: rank R_t into fronts, $R_t \rightarrow \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n\}$
19: $P_{t+1} \leftarrow \emptyset$
20: $i \leftarrow 1$
21: **while** $|P_{t+1}| + |\mathcal{F}_i| \leq N$ **do**
22: assign crowding distances on the front $\mathcal{F}_i$
23: $P_{t+1} \leftarrow \{P_t \cup \mathcal{F}_i\}$
24: $i \leftarrow i + 1$
25: **end while**
26: sort $\mathcal{F}_i$ in descending order using $\prec_n$
27: $P_{t+1} \leftarrow$ the first $N - |P_{t+1}|$ solutions from $\mathcal{F}_i$
28: $Q_{t+1} \leftarrow$ select, crossover and mutate P_{t+1}
29: randomly insert all $x_i \in O_t$ into Q_{t+1}
30: $t \leftarrow t + 1$
31: **end while**



(a) Exploration of ZDT3 at generation 18



(b) Exploration of ZDT4 at generation 21

Figure 4: This figure illustrates how our algorithm deterministically identifies which front needs to be explored first and gradually discovers the entire PF. The example here demonstrates a case of ZDT3 problem (Figure 4a). The outlier dots (orange) represents the deterministically generated solutions that did not survived because they are weakly dominated. The light green dots are those that are deterministically generated and survived. In the case of ZDT4 (Figure 4b), we can see how the bias and gaps have been corrected by our approach.

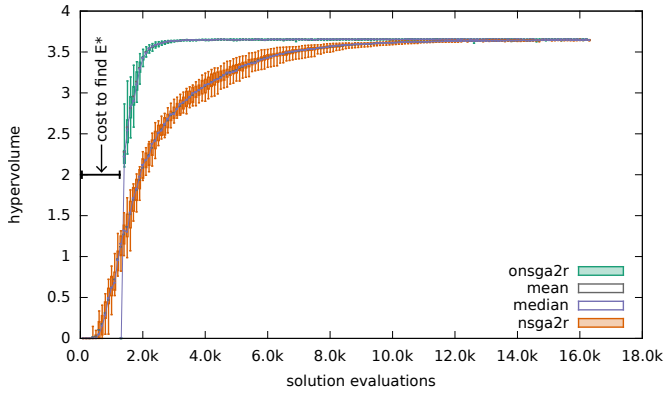


Figure 5: The convergence test of Algorithm 3 (onsga2r) vs. NSGA-II on problem ZDT1. The curves are actually consisted of box-plots, and for the Algorithm 3 (onsga2r), the medians are joined with a line. Here we can see that HV is 0 upto 1.8k SE and then abruptly reaches to 2 (the long vertical line at the beginning of Algorithm 3's curve).

Problem	Ideal HV (IHV)	Reference Points: (f_1, f_2, f_3)	NSGA-II 90%-IHV / SE	Algorithm3 90%-IHV / SE
ZDT1	3.67	(2.0, 2.0)	5100	**1909**
ZDT2	3.34	(2.0, 2.0)	7700	**2164**
ZDT3	4.82	(2.0, 2.0)	4900	8497
ZDT4	3.67	(2.0, 2.0)	14500	**4950**
ZDT6	15.35	(4.0, 4.0)	10500	**1539**
DTLZ1	999.98	(10.0, 10.0, 10.0)	24800	**11602**
DTLZ2	7.48	(2.0, 2.0, 2.0)	1200	10602
DTLZ3	3374.98	(15.0, 15.0, 15.0)	30600	**11802**
DTLZ4	7.48	(2.0, 2.0, 2.0)	1800	2022
DTLZ5	6.1	(2.0, 2.0, 2.0)	1200	1226
DTLZ6	55.6	(4.0, 4.0, 4.0)	29400	**24602**
DTLZ7	134.20	(10.0, 10.0, 10.0)	9200	**3606**

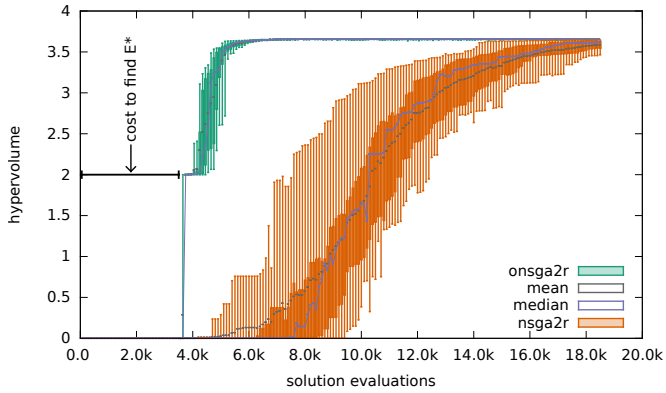
Table 1: The ideal HV, corresponding reference points. The last two colums represent the total number of SEs to attain 90% of the ideal HVs for both algorithms over the benchmark problems. For DTLZ7 problem, the ideal HV is approximated using a Monte-Carlo sampling.

side by side. The performance measure for our test was Hypervolume (HV) [14], and we are interested to see which algorithm can reach to a desired HV within less solution evaluations (SE). For all problems, we have seen our algorithm can demonstrate a very steep convergence to the true PF, even when the extra SE from Algorithm 1 are compensated for NSGA-II.

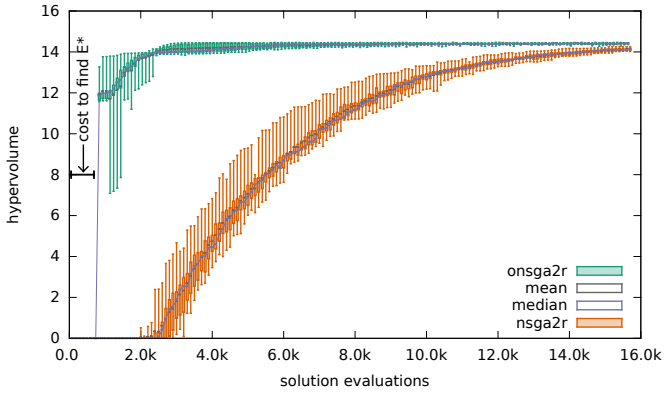
Moreover, as we already know, the standard test problems are designed from some of the well known single-objective optimization functions. For example, in the case of ZDT4, $g(\vec{x})$ is the well known *Rastrigin's function*, and the two objectives are related as $f_2 = 1 - \sqrt{f_1}$. Therefore, each solution on the true PF follows a well-defined pattern – the variables $x_2, x_3, \ldots x_n$ are all the same (solution to the *Rastrigin's problem*) and the first variable x_1 defines the trade-off. As a result, if our algorithm can find any solution on the true-PF at a certain time instance, the overshoot-correction on the line 13 of Algorithm 3 have a good chance of creating a valid neighbouring solution. To mute such an un-warranted exploration, we have changed the problem definition in such a way that all the solutions on the true-PF will have $x_1 = 0.5$, so that the Pareto-set is consisted of a straight line located at $x_1 = 0.5$ instead of $x_1 = 0.0$. Such modifications are done on the problems where appropriate.

All the results are collected from 31 independent runs started with non-identical random seeds. In all plots, *onsga2r* stands for Algorithm 3. The extra cost to find the extreme points are indicated with a "T" arrow on the x-axis. During computation of the HV measure, we have set the reference point to $\{2.0, 2.0\}$ for all problems except ZDT6, where it has been set to $\{4.0, 4.0\}$[7]. The Table 1 summarizes the per-

[7]The code that we have used to compute HV measure was taken from http://www.wfg.csse.uwa.edu.au/hypervolume/index.html#code, where the implementation assumes that all the objective values be on the one side of the reference



(a) Convergence test for ZDT4 problem



(b) Convergence test for ZDT6 problem

Figure 6: These plots illustrates the comparative analysis of the convergence rates for 2-objective problems (ZDT4 and ZDT6), the curves are actually consisted of box-plots. Here onsga2r denotes our algorithm and nsga2r is NSGA-II.

formance measuring parameters, the second column shows the *ideal* hypervolume of each of the PF of the corresponding problems. The second and the third column presents the number of SE have been spent to reach 90% of the HV by each of the algorithms. To ensure a completely fair comparison, for all instances, we have excluded the solutions in E^* during the HV computation.

The experiment with ZDT1 is illustrated in Figure 5, here we can see that the Algorithm 1 takes up to around 2K of solution evaluations. Given that, NSGA-II still lags behind with a multiple factors to reach the desired PF. We have seen similar effect on all the rest of the problems ZDT2, ZDT3, ZDT4 and ZDT6. Except for ZDT3, we have observed some fluctuations due the disconnected nature of the true PF. Due to the space constraints, we present a subset of the results in the Figure 6.

There is another interesting observation we have made, once this opposite point generation scheme is used, the search process becomes more focused and works in a more predictive manner. As an example, in the case of ZDT3 problem (where the true PF consists of 5 disconnected curves), the algorithm first tries to fill up the first partition and gradually moves to the next. The algorithm automatically detects which portion of the PF needs to be addressed first and try to fill the gaps by deliberately injecting points to the vicinity of those gaps. As a result our model can infer which objective is hard to solve and deterministically decides which one needs to be explored more. This scenario is illustrated in Figure 4, where we can see how the point generation algorithm moves from one disconnected front to the next.

Moreover, our approach can also efficiently solve the issue of *search trajectory bias*, if we look at the Figure 4b, we can see that the new solutions are deterministically generated where the explorations are not done thoroughly yet.

In the next experiment, we have carried out the similar tests with the scalable problem sets – DTLZ1, DTLZ2, DTLZ3, DTLZ4, DTLZ5, DTLZ6 and DTLZ7 [4]. For all cases we have considered 3-objectives. The control was NSGA-II results and similarly we compensate the measure with the extra SE to find extremes. All the results are collated from 31 independent runs. The reference point for HV computation has been set to $\{2.0, 2.0, 2.0\}$ for DTLZ2, DTLZ4 and DTLZ5. For DTLZ1 and DTLZ7 it has been set to $\{10.0, 10.0, 10.0\}$, for DTLZ3 it was $\{15.0, 15.0, 15.0\}$ and for DTLZ6, it was $\{4.0, 4.0, 4.0\}$. Due to the space constraints, we present a subset of the results in the Figure 7. In this paper, we only present the results for those problems that are comparatively harder to solve, i.e. problems with local-optima, disconnected fronts, non-uniform densities in the true PF etc.

point. For ZDT6, a closer reference point made the curves in the plots to be showing up very late at the end of x-axis.

6. CONCLUSIONS AND FUTURE WORKS

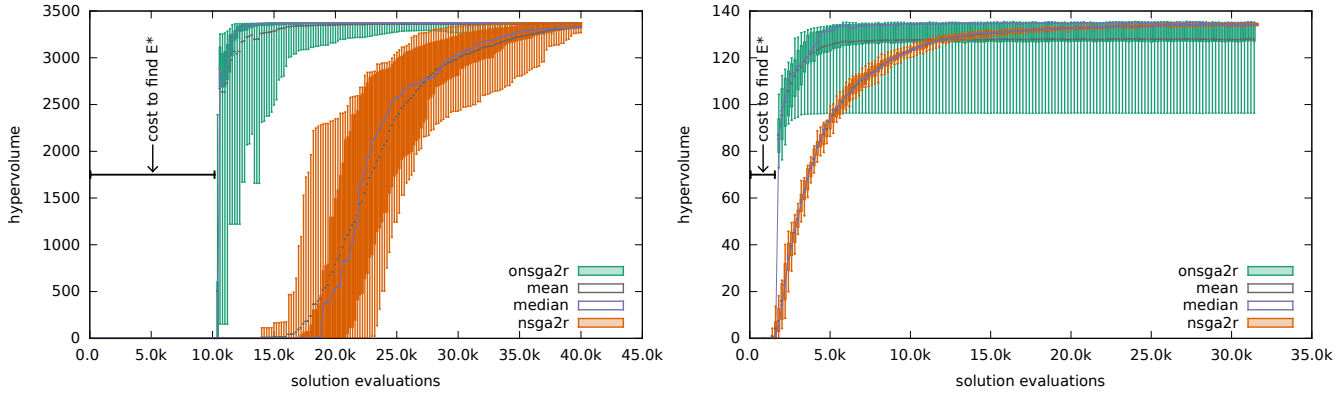
The main contribution of this paper is the incorporation of opposite point generation scheme in a different perspective. Our approach also shows that a simple and a deterministic scheme can aid the EMO optimization algorithm in a very interesting way. Our technique is also easy to implement and offers less overhead to the host algorithm. This approach can also correct the search bias introduced by the problem difficulty in an automated and predictable manner. The original idea of the *Opposition Based Learning (OBL)* is quite interesting; and we can make more of it if this idea is utilized in a more meaningful way – we think this is the main contribution of our study.

However, in the future we want to address some other interesting issues with our current study, especially in the "many-objective" problems. Moreover, we think there are also a possible scope in changing the idea *opposition* for reference point based many-objective algorithms like MOEA/D and NSGA-III [2]. In the future research, we hope to investigate these ideas.

7. REFERENCES

[1] C. K. Chow and S. Y. Yuen. An Evolutionary Algorithm That Makes Decision Based on the Entire Previous Search History. *Evolutionary Computation, IEEE Transactions on*, 15(6):741–769, Dec 2011.

[2] K. Deb and H. Jain. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving



(a) Convergence test for DTLZ3 problem

(b) Convergence test for DTLZ7 problem

Figure 7: These plots illustrates the comparative analysis of the convergence rates for 2 3-objective problems (DTLZ3 and DTLZ7), the curves are actually consisted of box-plots. Here onsga2r denotes our algorithm and nsga2r is NSGA-II. For DTLZ7, our algorithm shows better convergence in terms of mean and median hypervolume.

Problems With Box Constraints. *Evolutionary Computation, IEEE Transactions on*, 18(4):577–601, Aug 2014.

[3] K. Deb, K. Miettinen, and S. Chaudhuri. Toward an Estimation of Nadir Objective Vector Using a Hybrid of Evolutionary and Local Search Approaches. *Evolutionary Computation, IEEE Transactions on*, 14(6):821–841, Dec 2010.

[4] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable Multi-objective Optimization Test Problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 825–830, May 2002.

[5] M. Ergezer and D. Simon. Probabilistic properties of fitness-based quasi-reflection in evolutionary algorithms. *Computers & Operations Research*, 63:114 – 124, 2015.

[6] J. Kulk and J. Welsh. Using redundant fitness functions to improve optimisers for humanoid robot walking. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 312–317, Oct 2011.

[7] X. Ma, F. Liu, Y. Qi, M. Gong, M. Yin, L. Li, L. Jiao, and J. Wu. MOEA/D with opposition-based learning for multiobjective optimization problem. *Neurocomputing*, 146:48 – 64, 2014.

[8] S. Rahnamayan, H. Tizhoosh, and M. Salama. Opposition-Based Differential Evolution. *Evolutionary Computation, IEEE Transactions on*, 12(1):64–79, Feb 2008.

[9] V. Tirronen, F. Neri, and T. Rossi. Enhancing Differential Evolution frameworks by scale factor local search - Part I. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 94–101, May 2009.

[10] H. Tizhoosh. Opposition-Based Learning: A New Scheme for Machine Intelligence. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 1, pages 695–701, Nov 2005.

[11] H. R. Tizhoosh and S. Rahnamayan. Learning Opposites with Evolving Rules. *CoRR*, abs/1504.05619, 2015.

[12] M. Ventresca and H. R. Tizhoosh. A diversity maintaining population-based incremental learning algorithm. *Information Sciences*, 178(21):4038 – 4056, 2008.

[13] H. Wang, Z. Wu, S. Rahnamayan, Y. Liu, and M. Ventresca. Enhancing particle swarm optimization using generalized opposition-based learning. *Information Sciences*, 181(20):4699 – 4714, 2011. Special Issue on Interpretable Fuzzy Systems.

[14] L. While, L. Bradstreet, and L. Barone. A Fast Way of Calculating Exact Hypervolumes. *Evolutionary Computation, IEEE Transactions on*, 16(1):86–95, Feb 2012.

[15] A. Wierzbicki. The Use of Reference Objectives in Multiobjective Optimization. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 468–486. Springer Berlin Heidelberg, 1980.

[16] Q. Zhang and H. Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *Evolutionary Computation, IEEE Transactions on*, 11(6):712–731, Dec 2007.

[17] X. Zhang and S. Y. Yuen. A directional mutation operator for differential evolution algorithms. *Applied Soft Computing*, 30:529 – 548, 2015.

[18] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, 2000.

[19] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100. CIMNE, Barcelona, Spain, 2002.