Constructing the Pareto Front using Limited Information: A Case of the Opposition based Solution Generation Scheme

Author-1, Author-2 and Author-3, MSU

COIN Report Nnumber

Abstract In this paper we investigate a curious example of opposition based solution generation applied to an evolutionary multi-objective optimization (EMO) algorithm, namely on NSGA-II . . .

1. INTRODUCTION (AND RELATED WORK)

Talk about some introductory stuffs and do some literature reviews to set the scenario, also discuss basic things like "what is MOP" etc.

2. AN ALTERNATIVE INTERPRETATION OF OPPOSITION

As we have already seen in the previous section, mostly the idea of *opposition* is employed as the incorporation of new solutions with a certain kind of *opposite traits* into the existing population [**?**]. Such *traits* could be interpreted in terms different perspectives. For example, an *opposite* solution could be – i) the one with an opposite representation (with respect to the current best point) [**?**], ii) the ones with the opposite values from the other spectrum of the variable bounds (i.e. in the case of real valued optimization) [**?**]. However, upon injecting the opposite solutions could cause a re-route from the continuing search trajectory and thus could be a misleading step – in a sense that the opposite solutions could only be useful if 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 fact 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 assumes that the search space also needs to be 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 [**?**] into the current population. Therefore, the standard opposite injection scheme could only be effective given the two main assumptions on the underlying search space are valid – multi-modality and the solution symmetry. For this reason, such scheme is quite unwieldy to incorporate into a large scale numerical optimization problems, e.g. multi-objective optimization problems (MOPs) [**?**].

Moreover, we have also found some examples of opposition based algorithm for Q-learning/TD-learning like scenarios [**?**]. Similar argument can be made, as the reinforcement-learning algorithms are inherently greedy [**?**] (as they rely on the Bellman's optimality principle). And the opposite actions during the learning phase introduces a noise; so that the search can branch out to alternative choices

[**?**]. In that sense, we can say that the injection of opposite solutions can be considered as a different form of *variation operator* in population based stochastic search.

However, for the case of MOPs, existing opposition based solution generation scheme may not be effective just because of the sheer complexity of the search space. Moreover, we hardly have any room to make any assumption about the multi-modality and/or symmetry of the solution representation (i.e. such assumption could be made about N-queen problem). Therefore, in this paper, we have revised the notion of opposition in terms of the algorithm's behaviour. For example, most Evolutionary Multi-objective Optimization (EMO) algorithms aim to maximize two principal properties – i) the convergence and ii) the diversity, as the quality of a MOP solution depends on these two factors [**?**]. Therefore, this change of notion will now allows us to re-consider the opposite point generation/injection in a different perspective –

—Convergence: A solution *far* from the true Pareto-front is *opposite* to any solution that is *closer* to the true Pareto-front.

—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 solutions during the search. Obviously, the deterministic point generation scheme will only consider the *opposite trait* that is *good*. In the next section we will see, how the existing EMO algorithms shows the limitations maintaining this two *opposite traits* during the search (i.e. solution generation) process.

3. LIMITATIONS WITH THE CANONICAL MOP ALGORITHMS: THE SEARCH TRAJECTORY BIAS

Most of the standard EMO algorithms (e.g. NSGA-II, SPEA-II 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 point. They also shows a preference over a certain objective function which needs less exploration than the other. We can see such biasness in the search when we try to solve the ZDT4 problem using NSGA-II. In such 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 algorithm. Therefore, it would be helpful if we could selectively inject points during the search where the solution distribution is more sparse. In addition, we also think 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 a lot 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 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 [1].

[1]However, in the section **??** we will demonstrate this fact that just mutating the sparse solutions does not help much.

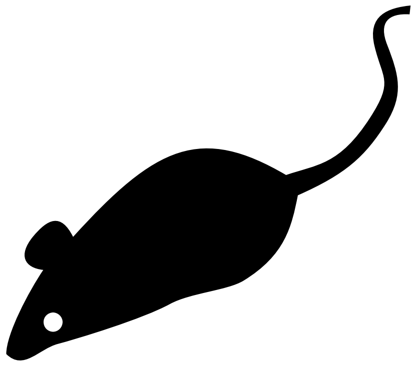


Fig. 1. The effect of the *search trajectory bias* could be seen when we try to solve ZDT4 problem with NSGA-II. Here we can see a long streak of crowded solution near the objective f_1, where the distribution of solutions near the objective f_2 is extremely sparse.

Therefore, in this paper we are going to demonstrate a very effective approach to address this 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 converging fast to the true PF as well.

4. THE DETERMINISTIC OPPOSITE POINT GENERATION SCHEME

As we have discussed in the section 2, we will utilize the so-called notion of *Opposition* to deterministically generate points on the strategically useful place 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 approximation to achieve our goal effectively [2].

Our basic approach 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 the *opposite* traits over the existing solutions. Also note that we are not going to deterministically define which portion of the search space is less easy (or hard) and so forth – we will try to devise a technique that will automatically address and solve such issues on the way.

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 function 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 –

4.1 Finding the Extreme Points

Extreme points on the Pareto-front could be found using global search as well [], however our goal was to save the extra computational cost as much as possible. Therefore, we resort to classical single-

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

objective optimization methods to solve this problem. Our choice of such algorithms were limited to, namely, the Interior Point Method[3] and Mesh Adaptive Direct Search[4]. Depending on the difficulty of the problems, appropriate routine parameters were empirically found out and they are summarized as follows:

—If the number of variables > 10, use `fmincon()` with default options.

—Otherwise use `patternsearch()` with default options, however if the number of objective > 2, then use these settings:
  —`InitialMeshSize:` population size
  —`TolX:` $1e^{-7}$, `TolBind:` $1e^{-6}$
  —`Search Method: @MADSPositiveBasis2N` starts searching with $2N$ random directions, where $N =$ number of variables.
  —`CompletePoll: on` and `CompleteSearch: on`

In general, for problems with less than 10 variables were solved using MADS and the larger problems were solved using IPM. The readers should be aware that the algorithm parameters that we have found is not universal setting, they are subjected to empirical investigations and problem domain-knowledge.

ALGORITHM 1: Find Extreme Points ()

```
$k \leftarrow$ no. of objectives for a particular problem
$T \leftarrow$ (population size $\times$ ($\frac{1}{4}$ $\times$ maximum generation)) $\times$ $\frac{1}{k}$
$E^* \leftarrow \{\}$, an empty solution set
**for** *all $i$ from $1$ to $k$* **do**
    $f_i \leftarrow$ $i$-th objective function
    $x \leftarrow$ random initial vector
    **while** *maximum of $\frac{T}{2}$ function evaluation reached* **do**
        **if** $|x| > 10$ **then**
            $x \leftarrow$ solve $f_i$ with fmincon()
        **end**
        **else**
            $x \leftarrow$ solve $f_i$ with patternsearch()
        **end**
    **end**
    $x \leftarrow$ solution found from the above optimization loop
    $f_{\text{aasf}} \leftarrow$ construct AASF function from $f_i$
    **while** *maximum of $\frac{T}{2}$ function evaluation reached* **do**
        **if** $|x| > 10$ **then**
            $x \leftarrow$ solve $f_{\text{aasf}}$ with fmincon()
        **end**
        **else**
            $x \leftarrow$ solve $f_{\text{aasf}}$ with patternsearch()
        **end**
    **end**
    $E^* \leftarrow E^* \cup \{x\}$
    return $E^*$
**end**
```

The actual extreme point location algorithm was conducted in two steps – given a particular objective function f_i, first we try to solve it directly using either IPM or MADS (depending on the problem type); then after some T_1 iterations, we construct the Augmented Achievement Scalarizing Function

[3]We have used the `fmincon()` routine in MATLAB (v. R2014a) for this purpose.
[4]We have used the `patternsearch()` routine in MATLAB (v. R2014a) for this purpose.

(AASF) function from f_i and solve it again for T_2 iterations. To limit the function evaluations, we kept $T = T_1 + T_2$ to a constant value. For all problems, we have fixed this maximum iteration count to the $\frac{1}{4}$-th of the total generation specified. A basic listing for this routine is presented in algorithm 1. The set of the extreme points $E = \{E^*\}$ generated from this algorithm may not contain all unique solution, and also they might not be the true extreme always. However, our algorithm can still converge to the true PF extremes.

4.2 The Opposite Solution Generation Algorithm

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 randomly select 30% solutions from the current population and deterministically change them to generate opposite solutions to place them in strategically viable place. 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 $E = \{E^*\}$ 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 to keep the true extreme points if ones are found in the later generations; and also if E^* are happen to be the weakly dominated points. After this step, we select the points from E that are on the best front and with ∞ crowding distances[5]. Lets denote these selected points as E'. Now at this point, only two kinds of situations are possible:

—The set E' contains only the solutions E^* if we are in the initial generations, or

—The E' will contain the solutions E_c if we are in the later phase of the generations, where E_c will be the true PF extreme.

ALGORITHM 2: Generate Pivot Points (P, E^*)

```
$E \leftarrow$ true PF extremes $E^*$ found from FIND-EXTREME-POINTS()
$E_c \leftarrow$ find the extreme points from the current best front in population $P$
$E \leftarrow \{E^* \cup E_c\}$
$E' \leftarrow$ rank points in $E$ and select the extremes from the best front
for all points $p_i$ in $E - E'$ do
    if $p_i$ weakly dominates any solution $p_j \in E'$ then
        replace $p_j$ by $p_i$
    end
end
update $E^*$, $E^* \leftarrow E'$
$G \leftarrow$ find $k$ intermediary gap points from the current best front in population $P$
$E' \leftarrow \{E' \cup G\}$
return $E'$
```

However, during the intermediary generations, it could also happen that we might include some solutions into E that weakly dominate some points already in E, this inclusion will reduce the expected spread of the pivot points – that may diminish the effect of maintaining the diversity. 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 replace the weakly dominating point from E' with the one from the set $E - E'$. The readers might have already noticed that $|E'| \leq |E|$.

[5]Here, by "crowding distance", we mean the inter-solution distances as computed in NSGA-II.

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 biasness in the search trajectory is introduced. However, the difference in the relative difficulty of the objective functions may not be the only reason for such 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. 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 denote them as G. Clearly, these G solutions are those that reside on the edge of the broken Pareto-front. Now we add the G to the set E', thus we make E' as the final "pivot" solutions to generate the opposite points. This should also be noted that $|E'| > k$.

ALGORITHM 3: NSGA-II with Opposition

```
$t = 1$, $N \leftarrow$ population size $|P_t|$
$E^* \leftarrow$ call FIND-EXTREME-POINTS() to get the true PF extremes
while $t \leq$ some maximum generation do
    $P'_t \leftarrow$ randomly select 30% solution from $P_t$
    $E'_t \leftarrow$ call GENERATE-PIVOT-POINTS($P_t$, $E^*$) to make the pivot set
    $O_t \leftarrow \emptyset$
    for each solution $x_i \in P'_t$ do
        $S \leftarrow$ pick $k$ random solutions from $E'_t$
        $v \leftarrow v \in S$ such that it is the furthest from $x_i$
        $x_c \leftarrow$ generate opposite point from $v$ and $x_i$
        $O_t \leftarrow \{O_t \cup x_c\}$
    end
    $P_t \leftarrow \{P_t \cup E^*\}$, $R_t \leftarrow P_t \cup Q_t$
    $\mathcal{F} \leftarrow$ fast-non-dominated-sort($R_t$)
    $P_{t+1} \leftarrow \emptyset$, $i \leftarrow 1$
    while $|P_{t+1}| + |\mathcal{F}_i| \leq N$ do
        crowding-distance-assignment($\mathcal{F}_i$)
        $P_{t+1} \leftarrow P_t + \mathcal{F}_i$
        $i \leftarrow i + 1$
    end
    sort $\mathcal{F}_i$ in descending order using $\prec_n$
    $P_{t+1} \leftarrow$ the first $N - |P_{t+1}|$ solutions from $\mathcal{F}_i$
    $Q_{t+1} \leftarrow$ crossover-mutate($P_{t+1}$)
    for each solution $x_i \in O_t$ do
        randomly insert $x_i$ into $Q_{t+1}$
    end
    $t \leftarrow t + 1$
end
```

As we have mentioned at the beginning that we randomly select 30% of the current population for opposite point generation. We go through each of them and every time we randomly 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 scaling. Instead of any sophistication, we do this scaling in a more straightforward way – given a pivot vector $\mathbf{v}$ and a parent vector $\mathbf{x_p}$, we generate an *opposite* child $\mathbf{x_c}$ as $\mathbf{x_c} = U(\frac{3}{4}||\mathbf{x_c} - \mathbf{x_p}||, \frac{5}{4}||\mathbf{x_c} - \mathbf{x_p}||)\mathbf{x_p}$, where $U(d, u)$ is a uniform random number within the range $[d, u]$. The overall procedure is presented in algorithm 2. 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 in the

NSGA-II run, the algorithm 3 shows how to call this procedure in NSGA-II like algorithm. Moreover, this algorithm is "pluggable" in a sense that we can integrate it to any other elitist EMO algorithms.

5. EXPERIMENTS WITH THE MULTI-OBJECTIVE PROBLEM SETS

Intro to this section.

6. NSGA-II EQUIPPED WITH EXTREME POINTS

Discuss if NSGA-II is given two extreme points, how it behaves (less robust)

6.1 ZDT Problems

Discuss ZDT problem set results.

6.2 DTLZ Problems

Discuss DTLZ problem set results.

6.3 Constrained and Rotated Problems

Discuss constrained and rotated problem (OSY, DTLZ8) results.

7. NSGA-II COMPENSATED FOR THE EXTRA FUNCTION EVALUATIONS

Discuss what if we run the NSGA-II with the compensated function evaluations.

7.1 ZDT Problems

Discuss ZDT problem set results.

7.2 DTLZ Problems

Discuss DTLZ problem set results.

7.3 Constrained and Rotated Problems

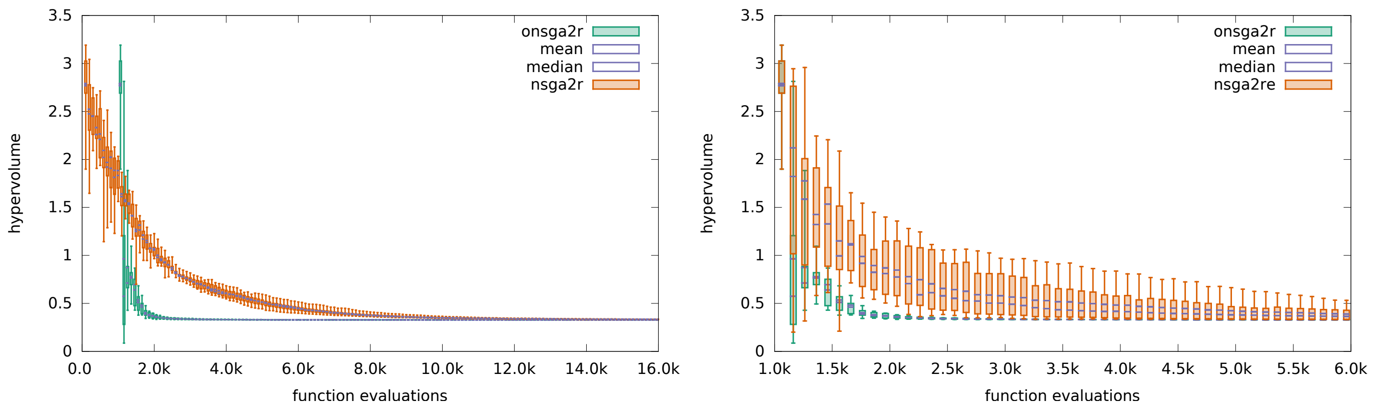
Discuss constrained and rotated problem (OSY, DTLZ8) results.

8. CONCLUSIONS AND FUTURE WORKS

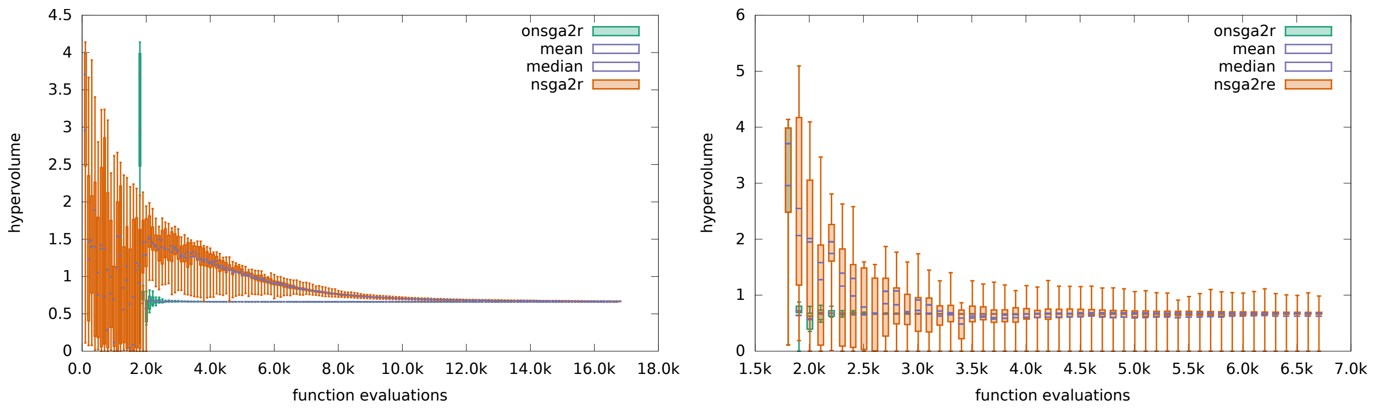
Discuss.

APPENDIX

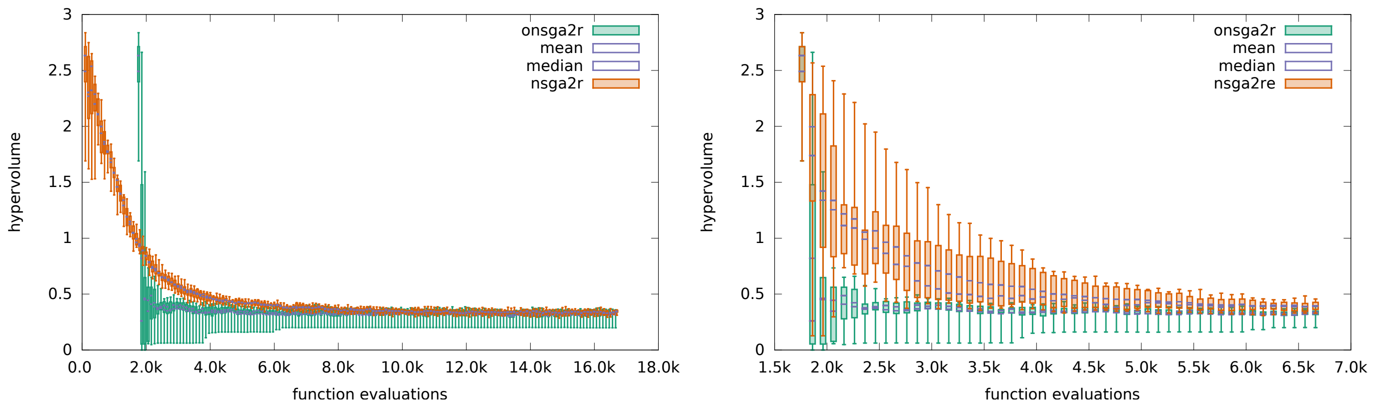
REFERENCES



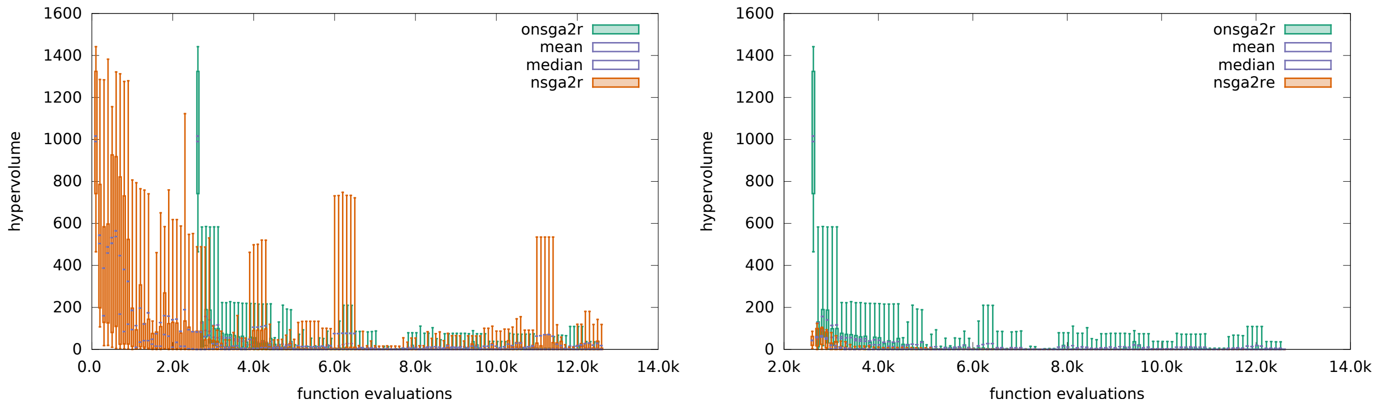
(a) zdt1



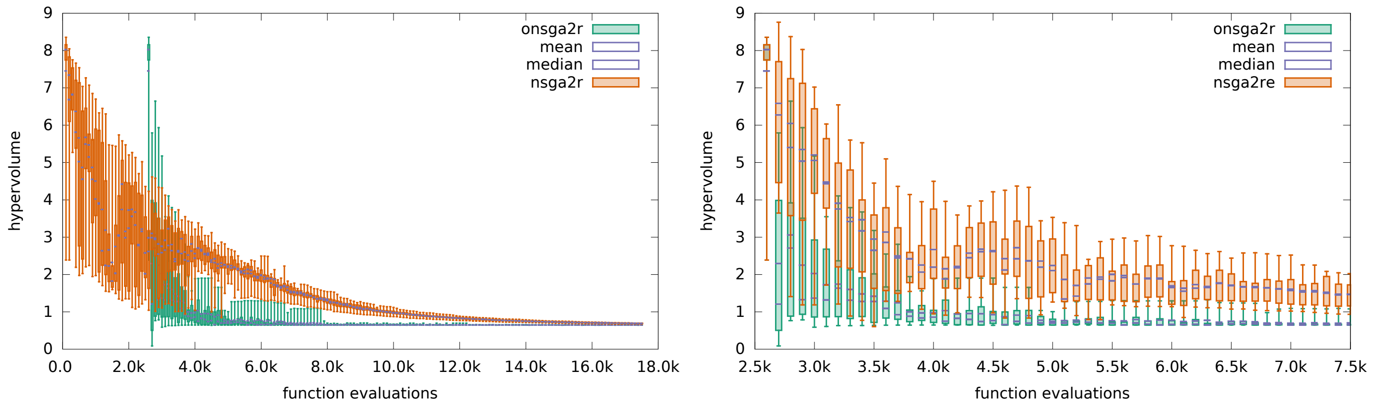
(b) zdt2



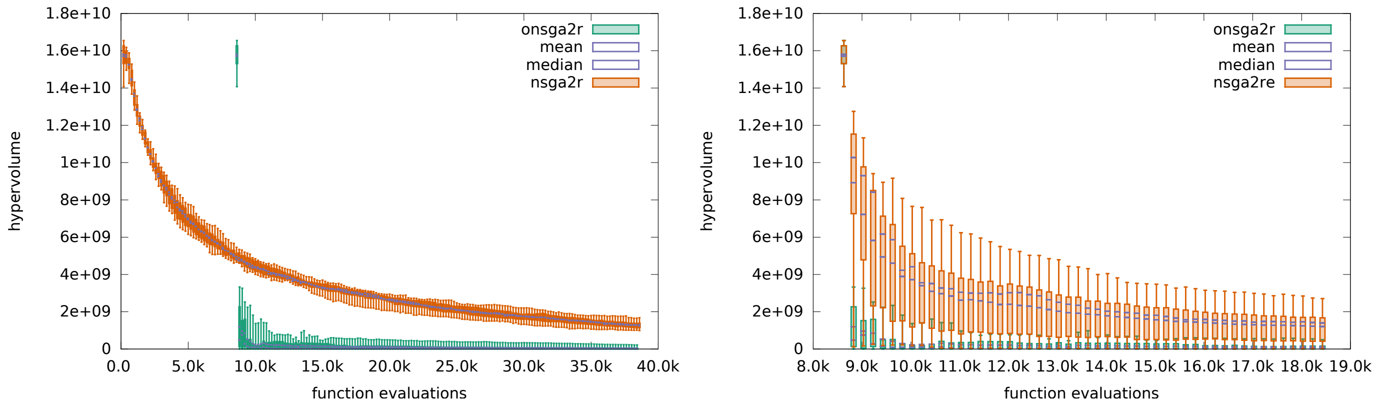
(c) zdt3



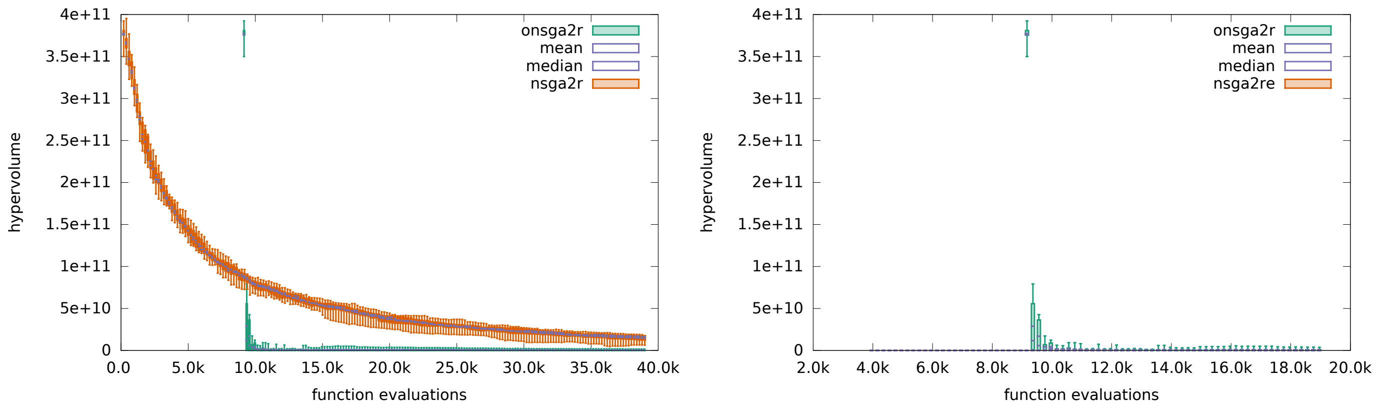
(d) zdt4



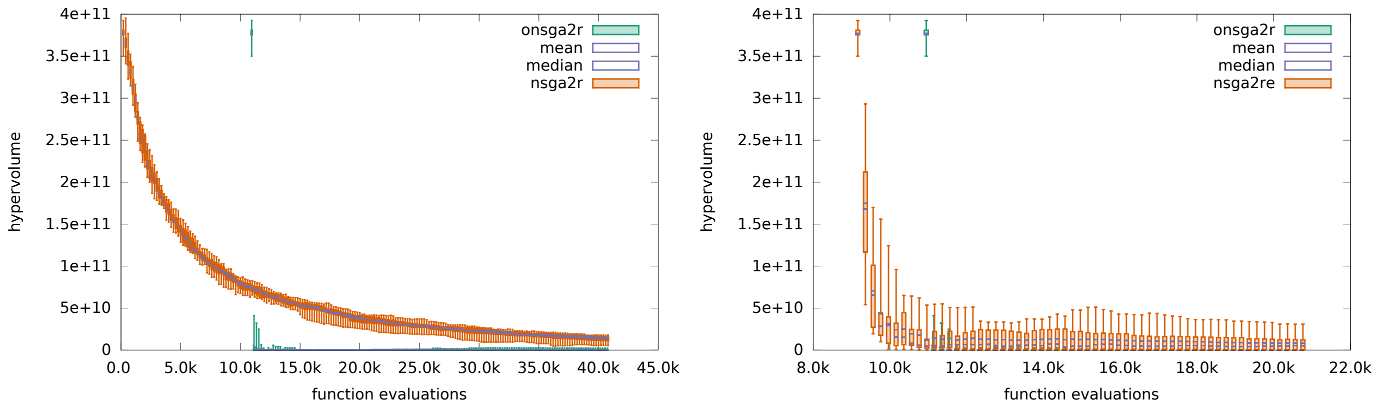
(e) zdt6



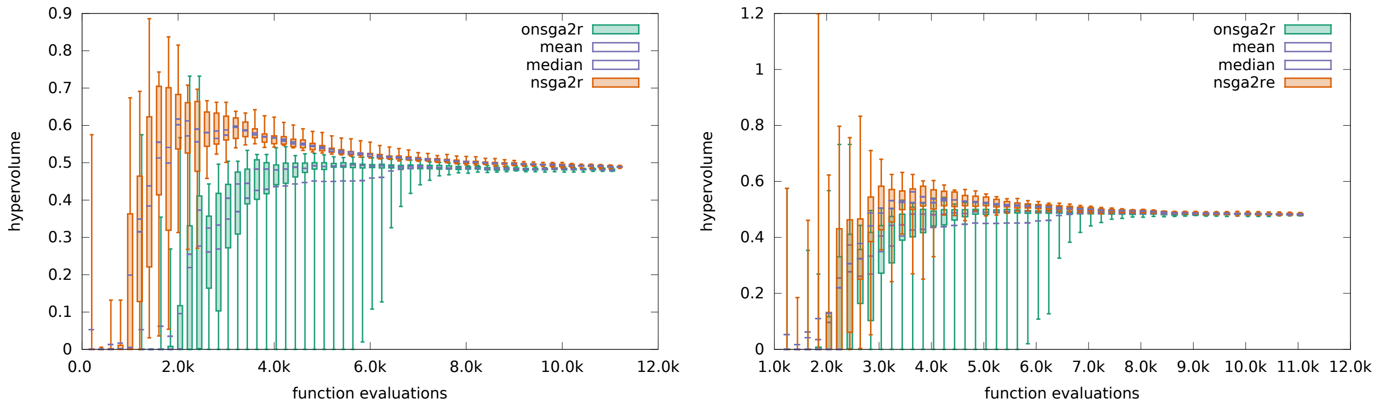
(f) dtlz1



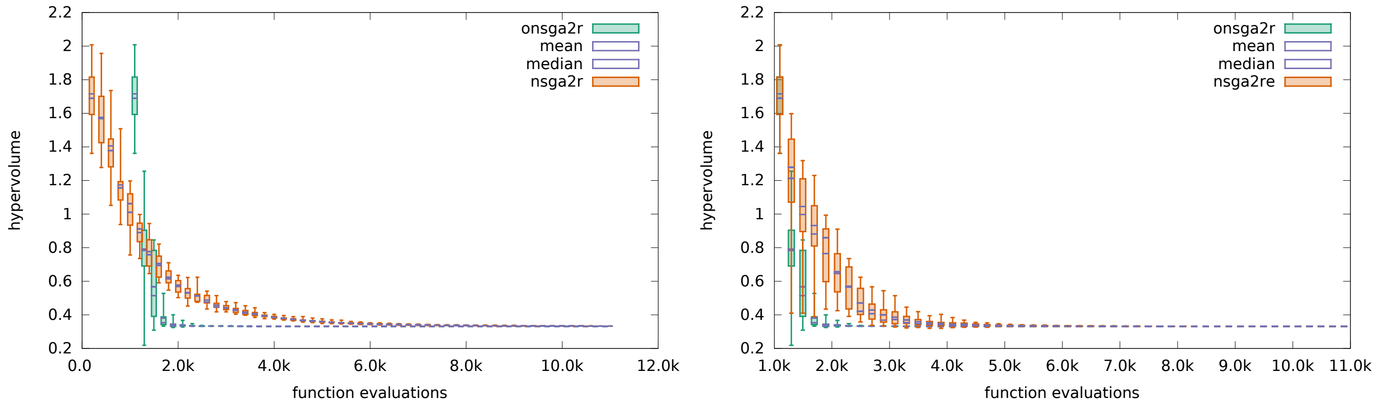
(g) dtlz2



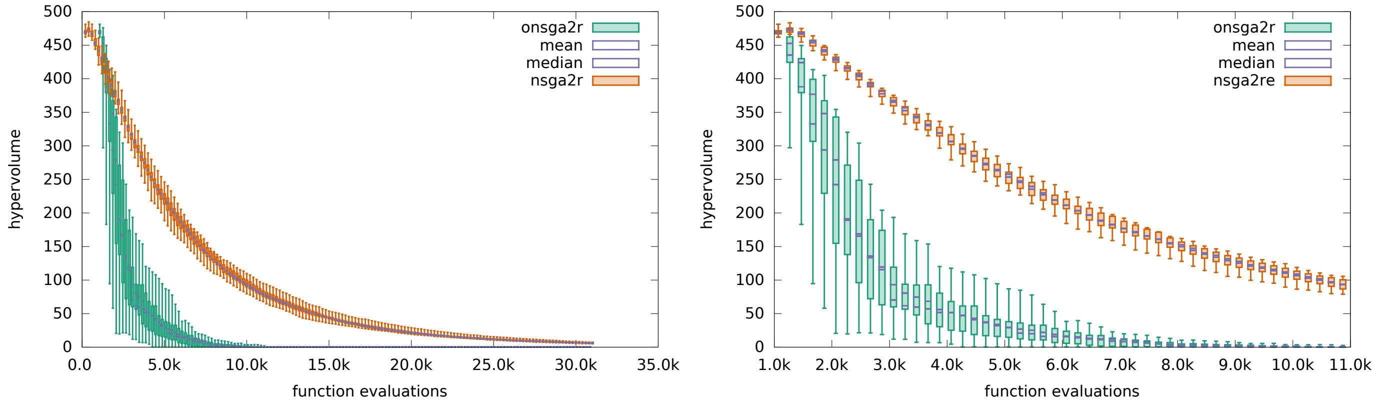
(h) dtlz3



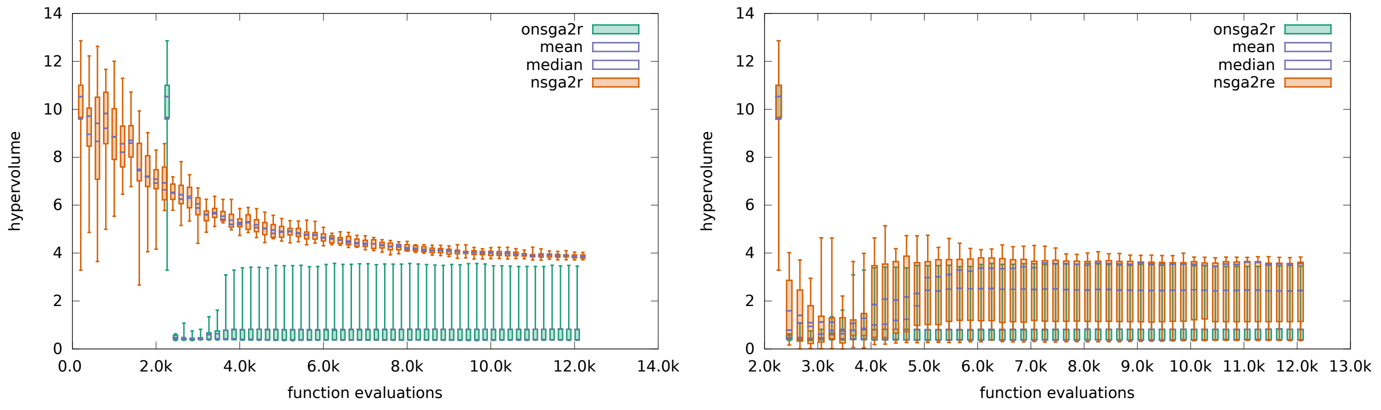
(i) dtlz4



(j) dtlz5



(k) dtlz6



(l) dtlz7

Online Appendix to: Constructing the Pareto Front using Limited Information: A Case of the Opposition based Solution Generation Scheme

Author-1, Author-2 and Author-3, MSU

© 2015 ACM. 1544-3558/2015/08-ART1 $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000