Hypervolume by Slicing Objective Algorithm: An Improved Version

Sumit Mishra[1], Srinibas Swain[1], Sangita Sarmah[1] and Carlos A. Coello Coello[2]

[1]Department of Computer Science & Engineering, IIIT Guwahati, Assam – 781015, India
[2]Departamento de Computación, CINVESTAV-IPN, Mexico City, D.F. 07360, Mexico
Email: sumit@iiitg.ac.in, srinibas@iiitg.ac.in, sangitasarmah77@gmail.com and ccoello@cs.cinvestav.mx

***Abstract*—The hypervolume remains a popular performance indicator in evolutionary multi-objective, mainly because of its nice mathematical properties (*i.e.*, it's the only performance indicator known to be Pareto-compliant). However, its high computational cost (which grows polynomially on the population size but exponentially on the number of objectives) has severely limited its use in many-objective optimization. This has motivated a variety of proposals that attempt to overcome this limitation. One of the most popular proposals currently available is the so-called Hypervolume by Slicing Objectives (HSO) algorithm. Here, we show that the worst-case time complexity of the HSO algorithm, as obtained by its authors, is incorrect. Then, we provide an efficient implementation of the HSO algorithm, which guarantees that unique slices are generated to compute the hypervolume.**

***Index Terms*—Evolutionary computation, Hypervolume, Non-dominated points, Time complexity.**

I. INTRODUCTION

In recent years, the solution of multi-objective optimization problems has received an increasing attention from researchers, particularly regarding problems with more than three objectives (the so-called many-objective optimization problems) [1]. This has motivated the development of a wide variety of Multi-objective Evolutionary Algorithms (MOEAs) [2]–[4]. As more MOEAs are proposed, performance assessment becomes crucial, which raises the importance for challenging test problems [5] and reliable performance indicators [6], [7]. Some of the most commonly adopted performance measures to assess convergence include the Hypervolume [8]–[10], $R2$ [11], the Inverted Generational Distance (IGD) [12], the Inverted Generational Distance plus (IGD$^+$) [13], the ϵ indicator [14], and Δ_p [15]. The high computational cost of the hypervolume (which is also called the $\mathcal{S}$ metric [8] and the Lebesgue measure [16]) in high dimensional spaces, has triggered a significant amount of research during the last few years (see for example [17]–[19].

Let $\mathbb{P} = \{p_1, p_2, \ldots p_N\}$ be a set of N points in $\mathbb{R}^M$ whose hypervolume needs to be calculated. The i^{th} point of $\mathbb{P}$; p_i is denoted as $\mathbb{P}\texttt{[i]}$. A point p in an M-dimensional space is a vector of size M represented as $\langle p_1, p_2, \ldots, p_M\rangle$. A reference point $\mathcal{R} \in \mathbb{R}^M$ is represented as $\langle R_1, R_2, \ldots, R_M\rangle$. In general, $N >> M$. We consider only maximization problem and with this assumption, a point $p = \langle p_1, p_2, \ldots, p_M\rangle$ dominates point $q = \langle q_1, q_2, \ldots, q_M\rangle$ denoted as $p \prec q$, if $\forall m \in \{1, 2, \ldots, M\}$ $p_m \geq q_m$ and $\exists m \in \{1, 2, \ldots, M\}$ $p_m > q_m$. Two points p and q are said to be *non-dominated* if neither p dominates q nor q dominates p. The hypervolume is defined as follows.

Definition 1 (Hypervolume). *The hypervolume of a set of "non-dominated points" measures the size of the portion of objective space that is dominated collectively by these points [9], [20]. The hypervolume of a set is measured relative to a reference point. This reference point is usually an anti-optimal point or the "worst possible" point in objective space [10].*

There have been numerous proposals to improve the computation of the exact hypervolume values [9], [10], [16]–[31]. The Lebesgue Measure Algorithm (LebMeasure) [16] has a time complexity $\mathcal{O}(N^M)$ [21]. The worst-case time complexity of the Hypervolume by Slicing Objectives (HSO) algorithm [20] is $\mathcal{O}(N^{M-1})$ [26], [31]. The time complexity of the FPL (Fonseca, Paquete and López-Ibáñez's) algorithm [23] is $\mathcal{O}(N^{M-2}\log N)$. The HOY (Hypervolume Overmars and Yap) algorithm [24] has an $\mathcal{O}(N^{\frac{M}{2}}\log N)$ time complexity. The Walking Fish Group (WFG) algorithm [10] has a time complexity $\mathcal{O}(2^{N+1})$ [10], but Lacour et al. [30] recently tightened this upper bound to $\mathcal{O}(N^{M-1})$. Chan's algorithm [28] has an $\mathcal{O}(N^{\frac{M}{3}}\text{polylog}N)$ time complexity. The time complexity of Quick Hypervolume (QHV) [29] was initially proved to be $\mathcal{O}(N(M + \log^{N-2}\log N)2^{NM})$ but it was later tightened to $\mathcal{O}(2^{M(N-1)})$ [18]. A modified version of QHV, called QHV-II has a time complexity $\mathcal{O}(M^{N-1})$ [18]. The time complexity of the Hypervolume Box Decomposition Algorithm (HBDA) [30] is $\mathcal{O}(N^{\lfloor\frac{M-1}{2}\rfloor+1})$. Table I summarizes the time complexity of various algorithms to compute the hypervolume. The time complexity of some of these algorithms [10], [29] for computing the hypervolume has been analyzed and modified [18], [30]. In this paper, we focus our analysis on one of these algorithms, namely, the Hypervolume by Slicing Objectives (HSO) algorithm, proposed by While *et al.* [20].

To compute the hypervolume in an M-dimensional objective space, HSO computes $(M-1)$-dimensional hypervolumes, recursively. [9], [22] have done some improvements to the

978-1-7281-8393-0/21/$31.00 ©2021 IEEE

TABLE I: Algorithms for the hypervolume problem with their time complexity.

Algorithm	**Time Complexity**
LebMeasure [16]	$\mathcal{O}(N^{M})$
HSO [20]	$\mathcal{O}(N^{M-1})$
FPL [23]	$\mathcal{O}(N^{M-2}\log N)$
HOY [24]	$\mathcal{O}(N^{\frac{M}{2}}\log N)$
WFG [10]	$\mathcal{O}(2^{N+1})$
Chan's algorithm [28]	$\mathcal{O}(N^{\frac{M}{3}}\text{polylog}N)$
QHV [29]	$\mathcal{O}(N(M+\log^{N-2}\log N)2^{NM})$
QHV-II [18]	$\mathcal{O}(M^{N-1})$
HBDA [30]	$\mathcal{O}(N^{\lfloor\frac{M-1}{2}\rfloor+1})$

HSO algorithm. In [22], two heuristics to improve the HSO algorithm have been proposed. The first one maximizes the number of dominated points in a slice. The second one calculates the number of non-dominated partial points in each slice explicitly for each objective and estimates the amount of work required to process each slice. However, the issue in the complexity analysis of HSO has not been addressed so far. In [9], it has been shown that the ordering of the objectives has a crucial role in the performance of the HSO algorithm. It has also been shown that the same set of points can be the worst-case scenario and the best-case scenario. These two scenarios are derived based on the order in which the objectives are chosen.

In this paper, we thoroughly analyze the HSO algorithm and show that the known worst-case time complexity of the HSO algorithm [9], [20], [22] is not accurate. We obtain a bound on the worst-case time complexity of the HSO algorithm. In our approach, we have obtained the maximum number of slices with a given number of points and objectives in HSO. Based on our analysis, we present an efficient implementation of the HSO algorithm *using memoization* [32]. We obtain a polynomial time complexity using the efficient implementation of HSO on the worst-case scenario of HSO (which takes exponential time). However, it is worth noting that the worst-case time complexity of the efficient implementation of HSO is indeed exponential.

The remainder of this paper is organized as follows. Section II describes the HSO algorithm and provides an analysis of its time complexity. In this section, we also derive the maximum number of slices for a given number of points and objectives in HSO. Section III provides an improved version of the HSO algorithm's implementation using memoization. Our empirical comparison of performance between HSO with its improved version is shown in Section IV. Finally, Section V provides our conclusions and some possible paths for future research.

II. HSO ALGORITHM

The HSO algorithm is based on the idea of processing one objective of the points at a time [20], "slicing" with respect to that particular objective. To compute the hypervolume of N points in an M-objective space, HSO initially sorts the points in descending order based on the first objective value. These sorted values are then used to create the cross-sectional "slices", with respect to the first objective. As the points are sorted based on the first objective value, the first slice contains only the highest value point in the first objective. The second slice contains only the points with the two highest values and so on. The last slice contains all the points. This process creates N slices, and each slice has $M-1$ objectives. The $M-1$ objective hypervolume of each of these N slices are calculated recursively, and the hypervolume of each slice is multiplied by its depth[1]. After this, these M-objective values are added in order to obtain the complete hypervolume. The process to compute the hypervolume is summarized in Algorithm 1.

Algorithm 1 $\text{HSO}(\mathbb{P}, \texttt{lb}, \texttt{ub}, \mathcal{R})$

```
Input: $\mathbb{P}$: Set of points, lb: First objective, ub: Last objective, $\mathcal{R}$: Reference point.
    $\langle$lb, lb+1, . . . , ub$\rangle$ objectives are considered while calculating the hypervolume
Output: $m$-objective hypervolume of the non-dominated points in $\mathbb{P}$ where $m$ =
    ub−lb+1
$\mathbb{P}' \leftarrow$ ND-POINTS($\mathbb{P}$, lb, ub) ▷ Set of non-dominated points among the points of
    $\mathbb{P}$ considering the objectives from lb to ub
Sort the points in $\mathbb{P}'$ in descending order based on lb$^{th}$ objective
if lb = ub then                              ▷ Base Case: Only one objective remaining
    return $\mathbb{P}'$[1]$_{lb}$          ▷ 1-objective hypervolume which is the value of the first
    point after sorting based on the lb$^{th}$ objective
vol ← 0
for i ← 1 to $|\mathbb{P}'|$ do
    Slice$_i$ ← $\{\mathbb{P}'[1] \cup \mathbb{P}'[2] \cup \ldots \cup \mathbb{P}'[i]\}$ ▷ i$^{th}$ slice contains initial i
    points
    vol$_{slice_i}$ ← HSO(Slice$_i$, lb+1, ub, $\mathcal{R}$)                ▷ Recursive call
    if i = $|\mathbb{P}'|$ then                                               ▷ Last slice
        depth ← $\mathbb{P}'$[i]$_{lb}$ − $R_{lb}$ ▷ Depth is the difference between the value of
        the lb$^{th}$ objective for i$^{th}$ point and reference point
    else
        depth ← $\mathbb{P}'$[i]$_{lb}$ − $\mathbb{P}'$[i+1]$_{lb}$  ▷ Depth is the difference between the
        value of the lb$^{th}$ objective for i$^{th}$ and i+1$^{th}$ point
    vol ← vol + vol$_{slice_i}$ × depth
return vol
```

A slice can contain many points. However, not all points "contained" in a slice will contribute to the hypervolume of that slice. Some points may be dominated in the remaining objectives, and they will not contribute to the hypervolume.

Now, we analyze the time complexity of the HSO algorithm. The time to obtain the set of non-dominated points is $\mathcal{O}(N \log N)$ for $M = 2, 3$ [33] and for $M \geq 4$ the time complexity is $\mathcal{O}(N \log^{M-2} N)$ [33]. The worst-case of the HSO algorithm occurs when all the points in each slice are non-dominated. This maximizes the number of points in each processed slice [22]. The only exception to the aforementioned scenario is when the slices are created for a single objective. We use Example 1 to demonstrate this worst-case scenario.

Example 1. *Consider* 4 *non-dominated points in a* 3*-dimensional space as shown in Fig. 1. The recursion tree to compute the hypervolume of these points is shown in Fig. 2.*

---

[1] The "depth" of slice i with respect to objective j is defined as the difference between the j^{th} objective value of the $(i+1)^{th}$ and the i^{th} points. The depth of the last slice with respect to the j^{th} objective is the difference between the j^{th} objective value of the reference point and the last point.

point	**Obj-1**	**Obj-2**	**Obj-3**
p_1	4	4	1
p_2	3	3	2
p_3	2	2	3
p_4	1	1	4

Fig. 1: Scenario for the worst case of HSO, with 4 points in 3-dimension (Example reproduced from [22]).

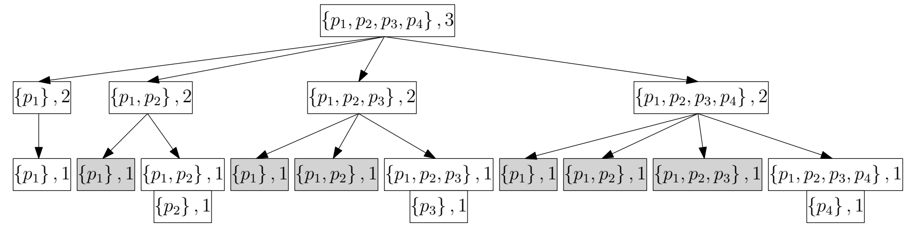


Fig. 2: Recursion tree in the worst case of HSO, to compute the hypervolume of 4 non-dominated points in 3-dimensional objective space.

The worst-case time complexity of the HSO algorithm is given by the recurrence relation in Eq. (1), which is different from the recurrence relation defined in [9], [20], where the time to obtain all the non-dominated points has not been considered. Also, the authors have not considered the time to sort the points with respect to the objectives in every slice.

$$\begin{aligned} T\left(N, M\right) &= \mathcal{O}(N \log^{M-2} N) + N \log N + \sum_{K=1}^{N} T\left(K, M{-}1\right) \\ &= \mathcal{O}(N \log^{M-2} N) + \sum_{K=1}^{N} T\left(K, M{-}1\right) \qquad (1) \end{aligned}$$

In Eq. (1), $T(N, M)$ gives the time to compute the hypervolume of N points in an M-dimensional objective space. We provide next a discussion on the recurrence relation:

- The term $\mathcal{O}(N \log^{M-2} N)$ corresponds to the time required to obtain all the non-dominated points.
- The term $N \log N$ corresponds to the time to sort the N points based on the first objective.
- The summation corresponds to the hypervolume contributed by N slices in dimension $M-1$.
- $T\left(K, M{-}1\right)$ corresponds to the time required to compute the hypervolume of K points in $M-1$ objectives.

Eq. (2) provides the base case for the recurrence used in Eq. (1). In this case, finding the non-dominated point require $\mathcal{O}(N)$ time.

$$T\left(N, 1\right) = N \qquad (2)$$

To illustrate the recursive nature of the HSO algorithm, we use Fig. 2, which shows the recursion tree for Example 1. The overall worst-case time complexity of the HSO algorithm is the sum of the time spent on each node of the recursion tree.

Let $\Gamma_{n,m}$ denote the number of slices (or nodes in the recursion tree) having n points with m objectives. Let the number of nodes with m objectives be denoted by $\Gamma_{*,m}$ and the number of nodes having n points be denoted as $\Gamma_{n,*}$. Therefore, $\sum_{n=1}^{N} \sum_{m=1}^{M} \Gamma_{n,m} = \sum_{n=1}^{N} \Gamma_{n,*} = \sum_{m=1}^{M} \Gamma_{*,m}$. The number of nodes having $n (1 \leq n \leq N)$ points with $m (1 \leq m \leq M)$ objectives is shown in Table II. Now, we obtain the total number of nodes in the recursion tree as follows: The number of nodes with M objectives is obtained using Eq. (3) and this node will be the root of the recursion tree.

$$\Gamma_{*,M} = 1 \qquad (3)$$

The number of nodes with $M-1$ objectives is obtained using Eq. (4) and all these nodes will be at level-1 of the recursion tree.

$$\Gamma_{*,M-1} = 1 + 1 + 1 + \cdots + 1 + 1 = N \qquad (4)$$

The number of nodes with $M-2$ objectives is obtained using Eq. (5) and all these nodes will be at level-2 of the recursion tree.

$$\Gamma_{*,M-2} = N + (N{-}1) + (N{-}2) + \cdots + 1 = \mathcal{O}(N^2) \qquad (5)$$

The number of nodes with $M-3$ objectives is obtained using Eq. (6) and all these nodes will be at level-3 of the recursion tree.

$$\begin{aligned} \Gamma_{*,M-3} = \textstyle\sum_{i_1=1}^{N} i_1 + \sum_{i_1=1}^{N-1} i_1 + \sum_{i_1=1}^{N-2} i_1 + \cdots + \\ \textstyle\sum_{i_1=1}^{2} i_1 + 1 = \mathcal{O}(N^3) \qquad (6) \end{aligned}$$

The number of nodes with $M-4$ objectives is obtained using Eq. (7) and all these nodes will be at level-4 of the recursion tree.

$$\begin{aligned} \Gamma_{*,M-4} = \sum_{i_1=1}^{N} \sum_{i_2=1}^{i_1} i_2 + \sum_{i_1=1}^{N-1} \sum_{i_2=1}^{i_1} i_2 + \cdots + \sum_{i_1=1}^{2} \sum_{i_2=1}^{i_1} i_2 + 1 \\ = \mathcal{O}(N^4) \qquad (7) \end{aligned}$$

The number of nodes with 1 objective is obtained using Eq. (8) and all these nodes will be at the last level (*i.e.*, level-$M-1$) of the recursion tree.

$$\begin{aligned} \Gamma_{*,1} = \sum_{i_1=1}^{N} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} \cdots \sum_{i_{M-3}=1}^{i_{M-4}} i_{M-3} + \\ \sum_{i_1=1}^{N-1} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} \cdots \sum_{i_{M-3}=1}^{i_{M-4}} i_{M-3} + \cdots + \\ \sum_{i_1=1}^{2} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} \cdots \sum_{i_{M-3}=1}^{i_{M-4}} i_{M-3} + 1 \\ = \mathcal{O}(N^{M-1}) \qquad (8) \end{aligned}$$

In [9], [20], [22], the authors have computed the exact number of nodes with a single objective, *i.e.*, $\Gamma_{*,1}$ and this

TABLE II: Number of nodes with different numbers of points and objectives.

		Number of points					
		1	**2**	**3**	$\cdots$	**$N-1$**	**N**
Number of Objectives	**M**	0	0	0	$\cdots$	0	1
	$M-1$	1	1	1	$\cdots$	1	1
	$M-2$	N	$N-1$	$N-2$	$\cdots$	2	1
	$M-3$	$\sum_{i_1=1}^{N} i_1$	$\sum_{i_1=1}^{N-1} i_1$	$\sum_{i_1=1}^{N-2} i_1$	$\cdots$	$\sum_{i_1=1}^{2} i_1$	1
	$M-4$	$\sum_{i_1=1}^{N} \sum_{i_2=1}^{i_1} i_2$	$\sum_{i_1=1}^{N-1} \sum_{i_2=1}^{i_1} i_2$	$\sum_{i_1=1}^{N-2} \sum_{i_2=1}^{i_1} i_2$	$\cdots$	$\sum_{i_1=1}^{2} \sum_{i_2=1}^{i_1} i_2$	1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
	1	$\sum_{i_1=1}^{N} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} \cdots \sum_{i_{M-3}=1}^{i_{M-4}} i_{M-3}$	$\sum_{i_1=1}^{N-1} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} \cdots \sum_{i_{M-3}=1}^{i_{M-4}} i_{M-3}$	$\sum_{i_1=1}^{N-2} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} \cdots \sum_{i_{M-3}=1}^{i_{M-4}} i_{M-3}$	$\cdots$	$\sum_{i_1=1}^{2} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} \cdots \sum_{i_{M-3}=1}^{i_{M-4}} i_{M-3}$	1

value is $\binom{N+M-2}{M-1}$. The total number of nodes in the recursion tree is obtained using Eq. (9).

$$\begin{aligned}\texttt{node} &= \sum_{m=M}^{1} \Gamma_{*,m} = 1 + N + \mathcal{O}(N^2) + \cdots + \mathcal{O}(N^{M-1}) \\ &= \mathcal{O}(N^M) \qquad (9)\end{aligned}$$

We have computed the maximum number of nodes in the recursion tree for $N(5 \leq N \leq 100)$ points with $M(5 \leq M \leq 10)$ objectives with the help of Table II, which is shown in Fig. 3. The time complexity of the HSO algorithm in the worst case is obtained using Eq. (10).

$$\begin{aligned}T(N,M) = &\sum\nolimits_{n=1}^{N} \mathcal{O}(n \log^{M-2} n)\Gamma_{n,M} + \\ &\sum\nolimits_{n=1}^{N} \mathcal{O}(n \log^{M-3} n)\Gamma_{n,M-1} + \\ &\sum\nolimits_{n=1}^{N} \mathcal{O}(n \log^{M-4} n)\Gamma_{n,M-2} + \cdots + \\ &\sum\nolimits_{n=1}^{N} \mathcal{O}(n \log n)\Gamma_{n,2} + \sum_{n=1}^{N} \mathcal{O}(n)\Gamma_{n,1} \\ = &\ \mathcal{O}(N \log^{M-2} N)\Gamma_M + \mathcal{O}(N \log^{M-3} N)\Gamma_{M-1} + \\ &\mathcal{O}(N \log^{M-4} N)\Gamma_{M-2} + \cdots + \mathcal{O}(N \log N)\Gamma_2 + \\ &\mathcal{O}(N)\Gamma_1 \\ = &\ \mathcal{O}(N \log^{M-2} N)\left[\Gamma_M + \Gamma_{M-1} + \cdots + \Gamma_2\right] + \\ &\mathcal{O}(N)\Gamma_1 \\ = &\ \mathcal{O}(N \log^{M-2} N)\left[\mathcal{O}(N^M) - \mathcal{O}(N^{M-1})\right] + \\ &\mathcal{O}(N)\mathcal{O}(N^{M-1}) \\ = &\ \mathcal{O}(N^{M+1} \log^{M-2} N) \qquad (10)\end{aligned}$$

Thus, the worst-case time complexity of the HSO algorithm is $\mathcal{O}(N^{M+1} \log^{M-2} N)$.

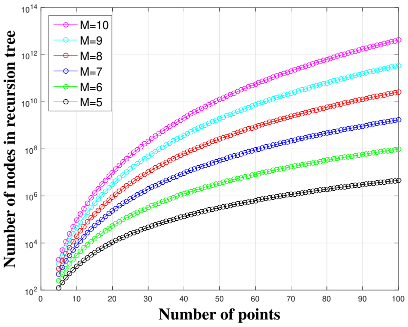


Fig. 3: Number of nodes in recursion tree for $N(5 \leq N \leq 100)$ points with $M(5 \leq M \leq 10)$ objectives.

III. IMPROVEMENT IN HSO

Consider the set of four points shown in Fig. 1. The recursion tree for these points is shown in Fig. 2. It is evident from Fig. 2, that in some cases we are computing the hypervolume of the same set of points for the same number of objectives, multiple times. For example, the hypervolume of the point p_1 in 1-dimension is computed four times. The hypervolume of points p_1, p_2 in 1-dimension is computed three times and the hypervolume of points p_1, p_2, p_3 in 1-dimension is computed twice. The redundant computations are shown in gray. There is no need to compute the hypervolume of the same set of points for the same number of objectives, multiple times. Thus, to make the HSO algorithm faster, whenever we get a slice, we first check whether the same slice has already been generated or not. *If* the slice has already been generated, then we use that hypervolume. *Otherwise*, we recursively compute the hypervolume of the slice and store it. This improved HSO algorithm is summarized in Algorithm 2. The changes introduced with respect to Algorithm 1 is marked in **boldface**.

A. Procedure to Check a Slice

A simple approach to check whether a slice has been generated or not is to check all the previously generated slices with the same number of points and objectives. However, this approach is inefficient. To make it efficient, we have used the *Trie* data structure [34], [35], where a node corresponds to the point of the slice. For this purpose, we create a $2D$ matrix (say $H_{\texttt{Trie}}$) of size $M \times N$ where a particular cell (m^{th} row and n^{th} column) corresponds to a trie data structure which stores the slices having n points with m objectives. Whenever we get a slice having n points with m objectives, we search

Algorithm 2 IMPROVED-HSO($\mathbb{P}$, $\texttt{lb}$, $\texttt{ub}$, $\mathcal{R}$)

```
Input: $\mathbb{P}$: Set of points, lb: First objective, ub: Last objective, $\mathcal{R}$: Reference point.
    $\langle$lb, lb+1, ..., ub$\rangle$ objectives are considered while calculating the hypervolume
Output: $m$-objective hypervolume of the non-dominated points in $\mathbb{P}$ where $m$ = ub−lb+1
n ← $|\mathbb{P}|$                                        ▷ Number of points in $\mathbb{P}$
m ← ub−lb+1                                      ▷ Number of objectives
if $H_{\texttt{Trie}}$[m][n] contains $\mathbb{P}$ then
    vol ← m-objective hypervolume of the non-dominated points in $\mathbb{P}$ stored in leaf node of the trie
    return vol
$\mathbb{P}'$ ← ND-POINTS($\mathbb{P}$, lb, ub) ▷ Set of non-dominated points among the points of $\mathbb{P}$ considering the objectives from lb to ub
Sort the points in $\mathbb{P}'$ in descending order based on lb$^{th}$ objective
if lb = ub then                                   ▷ Base Case: Only one objective remaining
    return $\mathbb{P}'$[1]$_{\texttt{lb}}$        ▷ 1-objective hypervolume which is the value of the first point after sorting based on the lb$^{th}$ objective
vol ← 0
for i ← 1 to $|\mathbb{P}'|$ do
    Slice$_{\texttt{i}}$ ← $\{\mathbb{P}'[1] \cup \mathbb{P}'[2] \cup \ldots \cup \mathbb{P}'[i]\}$ ▷ i$^{th}$ slice contains initial i points
    vol$_{\texttt{slice}_\texttt{i}}$ ← IMPROVED-HSO(Slice$_{\texttt{i}}$, lb+1, ub, $\mathcal{R}$)       ▷ Recursive call
    if $i = |\mathbb{P}'|$ then                                  ▷ Last slice
        depth ← $\mathbb{P}'$[i]$_{\texttt{lb}}$ − $R_{\texttt{lb}}$ ▷ Depth is the difference between the value of the lb$^{th}$ objective for i$^{th}$ point and reference point
    else
        depth ← $\mathbb{P}'$[i]$_{\texttt{lb}}$ − $\mathbb{P}'$[i+1]$_{\texttt{lb}}$ ▷ Depth is the difference between the value of the lb$^{th}$ objective for i$^{th}$ and i+1$^{th}$ point
    vol ← vol + vol$_{\texttt{slice}_\texttt{i}}$ × depth
Insert $\mathbb{P}$ and the vol in $H_{\texttt{Trie}}$[m][n]
return vol
```

that slice in the trie, which is at m^{th} row and n^{th} column. As there are n points in a slice, so the trie's depth is n, and the time required to search/insert a particular slice in the trie is $\mathcal{O}(n)$.

When we create a trie, a node in the trie can have at most n children. To reduce the number of children at each node, we sort the points of the slice in ascending order using counting sort [34][2] before insertion/search and this takes $\mathcal{O}(N)$ time. Thus, the overall time complexity to check whether a slice has already been generated or not is $\mathcal{O}(N)$. Also, the time to insert a slice in the trie is $\mathcal{O}(N)$. After sorting the points, the number of children of a node (which stores point p_i) at depth d is given as follows.

$$\text{No. of children} = N - i - n + d + 1 \tag{11}$$

Example 2. *Let there be five points* $\{p_1, \ldots, p_5\}$ *and* 4 *objectives. So, the size of the* $2D$ *matrix is* 4×5. *Let's assume that we have different slices of* 3 *points with* 2 *objectives. So, we check for the* 2^{nd} *row and the* 3^{rd} *column in the* $2D$ *matrix. Ten different slices having three points are shown in Fig. 4.*

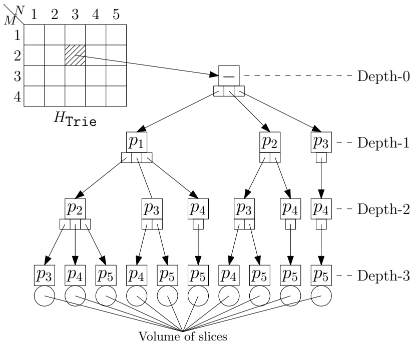


Fig. 4: $M \times N$ Trie to check whether a slice has already been generated. The circles store the hypervolume of the particular slice.

B. Time Complexity

The worst-case time complexity of the improved HSO algorithm proposed here is given by the recurrence relation in Eq. (12).

$$\begin{aligned} T\left(N, M\right) &= N + \mathcal{O}(N \log^{M-2} N) + N \log N + \\ &\quad \sum\nolimits_{K=1}^{N} T\left(K, M{-}1\right) \\ &= \mathcal{O}(N \log^{M-2} N) + \sum\nolimits_{K=1}^{N} T\left(K, M{-}1\right) \end{aligned} \tag{12}$$

In the recurrence relation (12), the first term N corresponds to the time required to find whether a slice has already been generated or not. The base case for Eq. (12) is the same as that from Eq. (2) when there is only one objective left. Note that the recurrence relation for our improved HSO is the same as the recurrence relation for HSO. Therefore, the worst-case time complexity of our improved HSO is also $\mathcal{O}(N^{M+1} \log^{M-2} N)$.

In the worst case, HSO explores $\sum_{n=1}^{N} \sum_{m=1}^{M} \Gamma_{n,m}$ nodes/slices. However, our improved HSO does not explore these many nodes, even in its worst-case. Our improved HSO always explores unique slices. There are $\binom{N}{n}$ ways to select n points from N points. So, the number of unique slices having n points with m objectives is given by $\binom{N}{n}$. However, all $\binom{N}{n}$ slices may not be generated during the improved HSO algorithm's execution. The maximum number of explored slices in our improved HSO having n points with m objectives is given by Eq. (13). The overall maximum number of explored slices in our improved HSO is given by Eq. (14).

$$\#\text{Slices}_{n,m} = \min\left(\binom{N}{n}, \Gamma_{n,m}\right) \tag{13}$$

$$\#\text{Slices} = \sum_{n=1}^{N} \sum_{m=1}^{M} \min\left(\binom{N}{n}, \Gamma_{n,m}\right) \tag{14}$$

We obtain the time complexity of our improved HSO in the worst-case scenario for HSO, which is given by Eq. (15).

$$\begin{aligned} T\left(N, M\right) &= N + \mathcal{O}(N \log^{M-2} N) + N \log N + \\ &\quad \sum\nolimits_{K=1}^{N} T'\left(K, M{-}1\right) \\ &= \mathcal{O}(N \log^{M-2} N) + \sum_{K=1}^{N} T'\left(K, M{-}1\right) \end{aligned} \tag{15}$$

---

[2] Let a slice have 3 points $\{p_2, p_5, p_4\}$. Then, after sorting, we will get $\{p_2, p_4, p_5\}$. This sorting can be performed using counting sort, which will take $\mathcal{O}(N)$ time.

This recurrence is different from the recurrence in Eq. (1). In this recurrence relation, $T'(K, M{-}1)$ is the time to compute the hypervolume of K points considering $M-1$ objectives. Now, we discuss how to solve $T'\,(K, M{-}1)$.

Initially, when we have a slice of K points with $M-1$ objectives, we first check whether the slice has already been obtained, which requires $\mathcal{O}(N)$ time. Then, we need to find the non-dominated points out of K points with $M-1$ objectives. In the worst-case scenario, all these K points remain non-dominated. Therefore, the time required to obtain all the non-dominated points is $\mathcal{O}(K\log^{M-3}K)$. We then sort these K points, which takes $\mathcal{O}(K\log K)$. After sorting these points, we have K slices (ranging from 1 to K points) with $M-2$ objectives. The slices with points ranging from 1 to $K-1$ with $M-2$ objectives have already been processed, and checking this requires $\mathcal{O}(N)$ time for each slice. Now, we obtain the hypervolume of K points with $M-2$ objectives and so on.

$$
\begin{aligned}
T'(K, M{-}1) &= N + \mathcal{O}(K\log^{M-3}K) + K\log K + \\
&\quad \underbrace{\{N + N + \ldots + N\}}_{K-1 \text{ times}} + T'\,(K, M{-}2) \\
&= N + \mathcal{O}(K\log^{M-3}K) + K\log K + N(K{-}1) + \\
&\quad T'\,(K, M{-}2) \\
&= \Big[\mathcal{O}(K\log^{M-3}K) + \mathcal{O}(NK)\Big] + T'\,(K, M{-}2) \\
&= \Big[\mathcal{O}(K\log^{M-3}K) + \mathcal{O}(NK)\Big] + \\
&\quad \Big[\mathcal{O}(K\log^{M-4}K) + \mathcal{O}(NK)\Big] + T'\,(K, M{-}3) \\
&= \Big[\mathcal{O}(K\log^{M-3}K) + \mathcal{O}(NK)\Big] + \\
&\quad \Big[\mathcal{O}(K\log^{M-4}K) + \mathcal{O}(NK)\Big] + \ldots + \\
&\quad [K\log K + \mathcal{O}(NK)] + T'\,(K, 1) \\
&= \Big[\mathcal{O}(K\log^{M-3}K) + \mathcal{O}(NK)\Big] + \\
&\quad \Big[\mathcal{O}(K\log^{M-4}K) + \mathcal{O}(NK)\Big] + \ldots + \\
&\quad [\mathcal{O}(K\log K) + \mathcal{O}(NK)] + [\mathcal{O}(K)] \\
&= \mathcal{O}(MK\log^{M-3}K + MNK)
\end{aligned}
\tag{16}
$$

Thus, the solution to the recurrence in Eq. (15) is obtained using Eq. (17).

$$
\begin{aligned}
T\,(N, M) &= N\log^{M-2}N + \textstyle\sum_{K=1}^{N} T'\,(K, M{-}1) \\
&= N\log^{M-2}N + \sum_{K=1}^{N}\Big(MK\log^{M-3}K + MNK\Big) \\
&= N\log^{M-2}N + \mathcal{O}(MN^2\log^{M-3}N) + \\
&\quad MN\frac{1}{2}N(N+1) \\
&= \mathcal{O}(MN^2\log^{M-3}N + MN^3)
\end{aligned}
\tag{17}
$$

Thus, the time complexity of our improved HSO in the worst-case scenario of HSO is $\mathcal{O}(MN^2\log^{M-3}N + MN^3)$. However, in the same scenario, the original HSO takes $\mathcal{O}(N^{M+1}\log^{M-2}N)$ time.

IV. EXPERIMENTAL ANALYSIS

In this section, we experimentally evaluate the effectiveness of our improved HSO with respect to the original HSO. For our experimental setup, we adopted the following hardware configuration: an x86_64 processor with 24 cores running at 1,200 MHz.

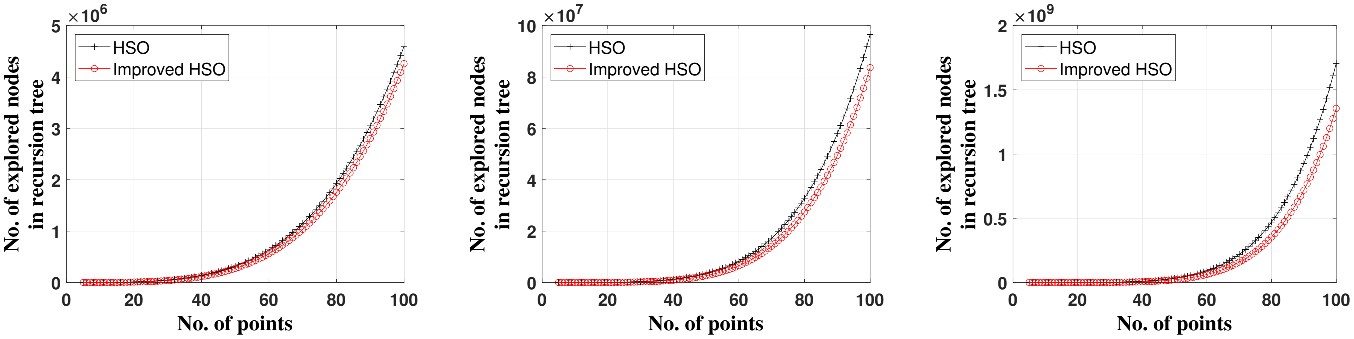
We have computed the maximum number of slices (or nodes in a recursion tree) explored by the original HSO and our improved HSO while computing the hypervolume for $N(5 \le N \le 100)$ points with $M(5 \le M \le 10)$ objectives. The maximum number of nodes explored by the original HSO has been calculated from the elements of Table II. The maximum number of nodes explored by our improved HSO is calculated using Eq. (14). Fig. 5 shows the maximum number of nodes in the recursion tree when both the original HSO and our improved HSO are used to obtain the hypervolume of $N(5 \le N \le 100)$ points in $M(5 \le M \le 10)$-dimensional objective space. From this figure, it is evident that with an increase in the number of objectives, the difference between the nodes explored by the original HSO and our improved HSO, increases.

To evaluate the performance of our improved HSO over the original HSO algorithm, we have randomly generated $N(5 \le N \le 30)$ points in 10, 15 and 20 dimensions. The reference point is considered to be the origin. We cannot compute the hypervolume for a large number of points and the objective because of hardware restrictions. The number of explored nodes in the recursion tree to compute the hypervolume of these N points for different objectives is shown in Fig. 6. We have also computed the running time to compute the hypervolume, which is shown in Fig. 7. From these two figures, it is evident that with an increase in the number of points and the objectives, the improved HSO seems to have better performance than that of the original HSO. The source code of our proposed algorithm is available at `https://github.com/sumitiitp/HSO`.

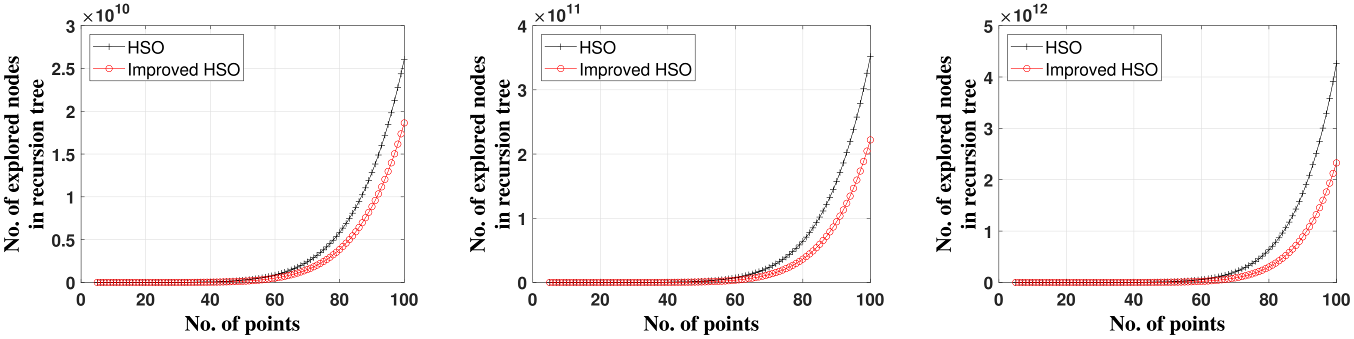
V. CONCLUSIONS AND FUTURE WORK

This paper has analyzed one of the hypervolume computation algorithms currently available: the Hypervolume by Slicing Objectives (HSO) algorithm [20]. This analysis identified an error in the worst-case time complexity analysis of the HSO algorithm and paved the way for the development of an improved implementation of this algorithm by restricting the repeated hypervolume computations using memoization. The time complexity of this improved version of HSO in the worst-case scenario of the HSO algorithm has also been obtained, which is $\mathcal{O}(M^2N^3)$. The maximum number of slices with a given number of points and objectives is also obtained for both the original HSO and for our improved version of the algorithm.

As part of our future work, we are interested in analyzing other hypervolume computation algorithms with the aim of

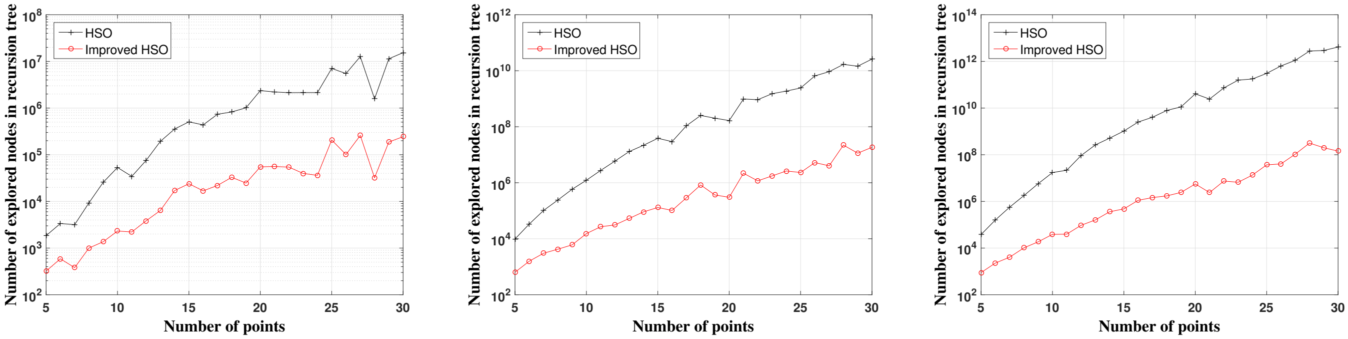


(a) $M = 5$ (b) $M = 6$ (c) $M = 7$



(d) $M = 8$ (e) $M = 9$ (f) $M = 10$

Fig. 5: The maximum number of nodes explored by the original HSO and our improved HSO for $N(5 \leq N \leq 100)$ points with $M(5 \leq M \leq 10)$ objectives.



(a) $M = 10$ (b) $M = 15$ (c) $M = 20$

Fig. 6: The number of nodes explored by the original HSO and our improved HSO for $N(5 \leq N \leq 20)$ random points for $10, 15, 20$ objectives.

identifying elements in their analysis that could lead to a more efficient implementation of them.

ACKNOWLEDGEMENTS

The last author acknowledges support from CONACyT grant no. 1920 and from SEP-Cinvestav grant (application no. 4).

REFERENCES

[1] C. A. Coello Coello, S. González Brambila, J. Figueroa Gamboa, M. G. Castillo Tapia, and R. Hernández Gómez, "Evolutionary Multiobjective Optimization: Open Research Areas and Some Challenges Lying Ahead," *Complex & Intelligent Systems*, vol. 6, pp. 221–236, July 2020.

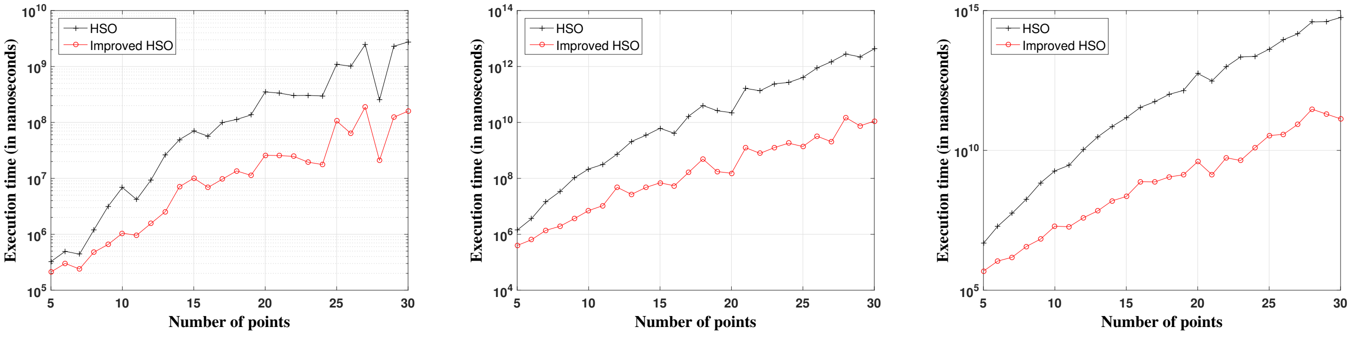
[2] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, August 2014.

[3] B. Li, J. Li, K. Tang, and X. Yao, "Many-Objective Evolutionary Algorithms: A Survey," *ACM Computing Surveys*, vol. 48, no. 1, September 2015.

[4] S. Mishra, S. Saha, and S. Mondal, "A Multiobjective Optimization Based Entity Matching Technique for Bibliographic Databases," *Expert Systems with Applications*, vol. 65, pp. 100–115, December 15 2016.

[5] S. Zapotecas-Martínez, C. A. Coello Coello, H. E. Aguirre, and K. Kiyoshi, "A Review of Features and Limitations of Existing Scalable Multi-Objective Test Suites," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 130–142, February 2019.

[6] M. Ravber, M. Mernik, and M. Crepinkek, "The Impact of Quality



(a) $M = 10$ (b) $M = 15$ (c) $M = 20$

Fig. 7: Running time of the original HSO and our improved HSO for $N(5 \leq N \leq 20)$ random points for $10, 15, 20$ objectives.

Indicators on the Rating of Multi-Objective Evolutionary Algorithms," *Applied Soft Computing*, vol. 55, pp. 265–275, June 2017.

[7] M. Li and X. Yao, "Quality Evaluation of Solution Sets in Multiobjective Optimisation: A Survey," *ACM Computing Surveys*, vol. 52, no. 2, May 2019, article number: 26.

[8] E. Zitzler, "Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications," Ph.D. dissertation, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.

[9] L. Bradstreet, L. While, and L. Barone, "A Fast Incremental Hypervolume Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 714–723, December 2008.

[10] L. While, L. Bradstreet, and L. Barone, "A Fast Way of Calculating Exact Hypervolumes," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 1, pp. 86–95, February 2012.

[11] D. Brockhoff, T. Wagner, and H. Trautmann, "On the Properties of the R2 Indicator," in *2012 Genetic and Evolutionary Computation Conference (GECCO'2012)*. Philadelphia, USA: ACM Press, July 2012, pp. 465–472, ISBN: 978-1-4503-1177-9.

[12] C. A. Coello Coello and N. Cruz Cortés, "Solving Multiobjective Optimization Problems using an Artificial Immune System," *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 163–190, June 2005.

[13] H. Ishibuchi, H. Masuda, Y. Tanigaki, and Y. Nojima, "Modified Distance Calculation in Generational Distance and Inverted Generational Distance," in *Evolutionary Multi-Criterion Optimization, 8th International Conference, EMO 2015*, A. Gaspar-Cunha, C. H. Antunes, and C. Coello Coello, Eds. Guimarães, Portugal: Springer. Lecture Notes in Computer Science Vol. 9019, March 29 - April 1 2015, pp. 110–125.

[14] E. Zitzler and S. Künzli, "Indicator-based Selection in Multiobjective Search," in *Parallel Problem Solving from Nature - PPSN VIII*, X. Y. et al., Ed. Birmingham, UK: Springer-Verlag. Lecture Notes in Computer Science Vol. 3242, September 2004, pp. 832–842.

[15] O. Schütze, X. Esquivel, A. Lara, and C. A. Coello Coello, "Using the Averaged Hausdorff Distance as a Performance Measure in Evolutionary Multiobjective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 4, pp. 504–522, August 2012.

[16] M. Fleischer, "The Measure of Pareto Optima. Applications to Multi-objective Metaheuristics," in *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, Eds. Faro, Portugal: Springer. Lecture Notes in Computer Science. Volume 2632, April 2003, pp. 519–533.

[17] J. Deng and Q. Zhang, "Approximating Hypervolume and Hypervolume Contributions Using Polar Coordinate," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 913–318, October 2019.

[18] A. Jaszkiewicz, "Improved quick hypervolume algorithm," *Computers & Operations Research*, vol. 90, pp. 72–83, February 2018.

[19] L. M. S. Russo and A. P. Francisco, "Extending Quick Hypervolume," *Journal of Heuristics*, vol. 22, no. 3, pp. 245–271, June 2016.

[20] L. While, P. Hingston, L. Barone, and S. Huband, "A Faster Algorithm for Calculating Hypervolume," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, pp. 29–38, February 2006.

[21] L. While, "A New Analysis of the LebMeasure Algorithm for Calculating Hypervolume," in *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*, C. A. C. Coello, A. H. Aguirre, and E. Zitzler, Eds. Guanajuato, México: Springer. Lecture Notes in Computer Science Vol. 3410, March 2005, pp. 326–340.

[22] L. While, L. Bradstreet, L. Barone, and P. Hingston, "Heuristics for Optimising the Calculation of Hypervolume for Multi-Objective Optimization Problems," in *2005 IEEE Congress on Evolutionary Computation (CEC'2005)*, vol. 3. Edinburgh, Scotland: IEEE Service Center, September 2005, pp. 2225–2232.

[23] C. M. Fonseca, L. Paquete, and M. L.-I. nez, "An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator," in *2006 IEEE Congress on Evolutionary Computation (CEC'2006)*. Vancouver, BC, Canada: IEEE, July 2006, pp. 3973–3979.

[24] N. Beume, "S-Metric Calculation by Considering Dominated Hypervolume as Klee's Measure Problem," *Evolutionary Computation*, vol. 17, no. 4, pp. 477–492, Winter 2009.

[25] L. Bradstreet, "The Hypervolume Indicator for Multi-objective Optimisation: Calculation and Use," Ph.D. dissertation, Department of Computer Science & Software Engineering, The University of Western Australia, Australia, April 2011.

[26] L. While and L. Bradstreet, "Applying the WFG algorithm to calculate incremental hypervolumes," in *2012 IEEE Congress on Evolutionary Computation (CEC'2012)*. Brisbane, Australia: IEEE Press, June 10-15 2012, pp. 489–496.

[27] A. P. Guerreiro, C. M. Fonseca, and M. T. Emmerich, "A Fast Dimension-Sweep Algorithm for the Hypervolume Indicator in Four Dimensions," in *24th Canadian Conference on Computational Geometry (CCCG'2012)*, Charlottetown, Prince Edward Island, Canada, 8-10 August 2012, pp. 77–82.

[28] T. M. Chan, "Klee's Measure Problem Made Easy," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS'2013)*. Berkeley, California, USA: IEEE, 26-29 October 2013, pp. 410–419, ISBN 0272-5428.

[29] L. M. S. Russo and A. P. Francisco, "Quick Hypervolume," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 481–502, August 2014.

[30] R. Lacour, K. Klamroth, and C. M. Fonseca, "A Box Decomposition Algorithm to Compute the Hypervolume Indicator," *Computers & Operations Research*, vol. 79, pp. 347–360, March 2017.

[31] A. P. Guerreiro, C. M. Fonseca, and L. Paquete, "The Hypervolume Indicator: Problems and Algorithms," *arXiv preprint arXiv:2005.00515*, 2020.

[32] J. M. Robson, "Algorithms for Maximum Independent Sets," *Journal of Algorithms*, vol. 7, no. 3, pp. 425–440, 1986.

[33] H.-T. Kung, F. Luccio, and F. P. Preparata, "On Finding the Maxima of a Set of Vectors," *Journal of the ACM*, vol. 22, no. 4, pp. 469–476, 1975.

[34] D. E. Knuth, *The Art of Computer Programming*. Pearson Education, 1997, vol. 3.

[35] D. E. Willard, "New Trie Data Structures which Support Very Fast Search Operations," *Journal of Computer and System Sciences*, vol. 28, no. 3, pp. 379–394, 1984.