Two-Stage Greedy Approximated Hypervolume Subset Selection for Large-Scale Problems

Yang Nan, Hisao Ishibuchi*, Tianye Shu, and Ke Shang

Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation
Department of Computer Science and Engineering
Southern University of Science and Technology, Shenzhen 518055, China
{nany@mail., hisao@, 12132356@mail.}sustech.edu.cn, kshang@foxmail.com

Abstract. Recently, it has been demonstrated that a solution set that is better than the final population can be obtained by subset selection in some studies on evolutionary multi-objective optimization. The main challenge in this type of subset selection is how to efficiently handle a huge candidate solution set, especially when the hypervolume-based subset selection is used for many-objective optimization. In this paper, we propose an efficient two-stage greedy algorithm for hypervolume-based subset selection. In each iteration of the proposed greedy algorithm, a small number of promising candidate solutions are selected in the first stage using the rough hypervolume contribution approximation. In the second stage, a single solution among them is selected using the more precise approximation. Experimental results show that the proposed algorithm is much faster than state-of-the-art hypervolume-based greedy subset selection algorithms at the cost of a slight deterioration of the selected subset quality.

Keywords: Evolutionary multi-objective optimization · Hypervolume subset selection · Two-stage hypervolume contribution approximation.

1 Introduction

Recently, the use of an unbounded external archive was examined in many studies [1–10] in the evolutionary multi-objective optimization (EMO) community. It was shown in [6–8] that the selected subset from the unbounded external archive is usually better than the final population. This is because in general the final population is not the best subset of all examined solutions. For example, final solutions can be dominated by other generated and discarded solutions in previous generations [10]. The main difficulty in subset selection from the unbounded external archive is a huge candidate solution set. For example, more than two million non-dominated solutions are included in a candidate solution set in [7]. Thus, efficient subset selection algorithms are needed.

Hypervolume subset selection (HSS) is a popular research topic since the hypervolume indicator [11] is the only performance indicator with Pareto compliant property [12]. In general, the HSS problem is to select a subset S_{sub} from

* Corresponding Author: Hisao Ishibuchi

a candidate set S_c so that the hypervolume of S_{sub} is maximized. Thus, the HSS problem can be formalized as follows:

$$S_{sub}^{*} = \underset{S_{sub} \subset S_c, |S_{sub}|=k}{\arg\max} HV(S_{sub}), \tag{1}$$

where S_{sub}^{*} is the optimal subset with k solutions, S_c is the given set of n candidate solutions (i.e., $|S_c| = n$), and $HV(S_{sub})$ is the hypervolume of S_{sub}. HSS methods can be classified into four categories [13]: exact methods, evolutionary methods, local search methods, and greedy methods.

Among them, the greedy methods [14–18] are the most well-known since they can obtain a near optimal subset (i.e., $(1 - 1/e)$ to the optimal subset is guaranteed [19]) within a reasonable computation time. Bradstreet *et al.* [14,15] proposed two basic greedy HSS methods: the greedy reduction and the greedy inclusion. However, the efficiency of the basic greedy HSS methods is low. Jiang *et al.* [16] proposed a hypervolume contribution update strategy that can significantly reduce the computation time of the basic greedy HSS methods. To further improve the efficiency of the greedy HSS method, Chen *et al.* [17] proposed a lazy greedy inclusion HSS method. In the lazy greedy HSS method, the submodular property of the hypervolume indicator [20] is utilized to avoid unnecessary hypervolume contribution calculations. Currently, the lazy greedy HSS method is the most efficient greedy HSS method.

Since the above-mentioned greedy HSS methods use the exact hypervolume contribution calculation, their computation time is very large in high-dimensional cases (i.e., many-objective problems). This is because the time complexity of the exact hypervolume contribution calculation is $O(k^{m-1})$ [13] where k is the number of solutions involved in hypervolume contribution calculation and m is the dimension of solutions (i.e., the number of objectives). To address this issue, Shang *et al.* [18] proposed a greedy approximated HSS method which uses an R2-based hypervolume contribution approximation method [21] instead of the exact calculation. As a result, the greedy approximated HSS method is much faster than the exact greedy HSS methods (including the lazy greedy HSS method). The selected subset quality (i.e., hypervolume value) by the approximated method is slightly worse than that by the exact HSS methods.

As pointed out in [22], the computation time of the greedy HSS methods (both the exact and approximated methods) severely increases as the number of candidate solutions increases. This is because the greedy methods need to examine all the unselected candidate solutions in each iteration. Thus, the above-mentioned greedy methods are not applicable to the large-scale subset selection where the number of candidate solutions is huge (e.g., more than 2,000,000 non-dominated solutions in the unbounded external archive [7]).

In this paper, we propose a two-stage greedy approximated HSS method to efficiently select a subset from a huge candidate solution set. In the proposed method, we use a two-stage hypervolume contribution approximation method for a hypervolume-based EMO algorithm to select a good solution in each iteration of greedy inclusion. Each iteration is divided into two stages. In the first stage, we roughly approximate the hypervolume contribution of each unselected candidate solution. Then, we preselect only a small number of promising candidate solutions. In the second stage, the hypervolume contribution of each preselected promising solution is more precisely approximated and one of them is selected. In this manner, we do not have to precisely approximate the hypervolume contribution of each candidate solution. Thus, the proposed two-stage method is much faster than other greedy methods for large-scale subset selection.

The remainder of this paper is organized as follows. In Section 2, the existing greedy HSS methods are briefly reviewed. The proposed two-stage greedy approximated HSS method is described in Section 3. Experimental results are reported in Section 4 to demonstrate the superiority of the proposed method. Finally, the conclusion is given in Section 5.

2 Greedy Hypervolume Subset Selection

For a large candidate set (i.e., $k \ll n$), the use of greedy reduction is unrealistic. Thus, in this paper, we focus only on greedy inclusion HSS methods where k solutions are selected from the candidate set S_c with n solutions one by one. In this section, we explain greedy exact and greedy approximated HSS methods.

2.1 Greedy Exact HSS Methods

Basic greedy inclusion HSS (GI-HSS [14]) The framework of the basic greedy inclusion HSS method (GI-HSS) is shown in Algorithm 1, which is also the framework of all greedy HSS methods. As explained in Section 1, the size of the candidate solution set S_c is n (i.e., $|S_c| = n$) and the size of the subset S_{sub} to be selected is k (i.e., $|S_{sub}| = k$). First, S_{sub} is empty. Then, a candidate solution is selected from S_c and added to S_{sub} in each iteration one by one. GI-HSS is terminated when $|S_{sub}|$ reaches k. In each iteration, GI-HSS calculates the hypervolume contribution of each unselected candidate solution. Then, the candidate solution with the largest hypervolume contribution is added to S_{sub}. Since the hypervolume contribution calculation is time-consuming, the efficiency of GI-HSS is low, especially when n is large (i.e., $k \ll n$).

Algorithm 1: Basic framework of greedy methods

```
input  : $S_c$ (candidate solution set), $k$ (selected subset size)
output: $S_{sub}$ (selected subset)
begin
1   $S_{sub} \longleftarrow \emptyset$;
2   while $|S_{sub}| < k$ do
3       Calculate the hypervolume contribution to $S_{sub}$ of each solution $s$ in
          $S_c$;
4       Select the best candidate solution $a$ with the largest hypervolume
          contribution to $S_{sub}$ from $S_c$;
5       $S_{sub} \longleftarrow S_{sub} \cup \{a\}$; $S_c \longleftarrow S_c \backslash \{a\}$;
```

Greedy inclusion HSS with hypervolume contribution updating (UGI-HSS [16]) To improve the efficiency of the basic greedy HSS method, Jiang *et al.* [16] proposed a hypervolume update strategy. In each iteration, GI-HSS calculates the hypervolume contribution of each candidate solution whereas UGI-HSS updates their hypervolume contribution more efficiently. As a result, UGI-HSS is much faster than GI-HSS.

Lazy greedy inclusion HSS (LGI-HSS [17]) The UGI-HSS method improves the efficiency of the basic greedy HSS method by reducing the computation time of the hypervolume contribution calculation for each candidate solution. In contrast, the LGI-HSS method improves the efficiency of GI-HSS by reducing the number of hypervolume contribution calculations. That is, LGI-HSS uses the submodular property of the hypervolume indicator [20] to avoid unnecessary hypervolume contribution calculations. Currently, the LGI-HSS method is the most efficient greedy exact HSS method.

2.2 Greedy Approximated HSS Method

Since the time complexity of hypervolume contribution calculation increases exponentially as the number of objectives m increases [13], the greedy exact HSS methods are impractical in high-dimensional cases (e.g., $m > 10$). To overcome this issue, a greedy approximated HSS method (GAHSS [18]) was proposed. In GAHSS, the hypervolume contribution of each candidate solution is approximated using the R2-based hypervolume contribution approximation method [21]. In this subsection, we briefly explain the mechanism of the R2-based hypervolume contribution approximation method and the framework of GAHSS.

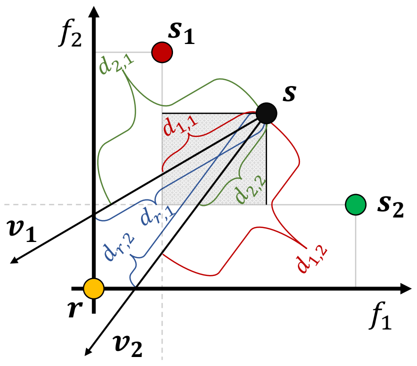

Fig. 1. Illustration of R2-based hypervolume contribution approximation

R2-based hypervolume contribution approximation (R2-HVC [21]) Fig. 1 illustrates the R2-based hypervolume contribution approximation method. Solutions $\boldsymbol{s}_1$, $\boldsymbol{s}_2$ and $\boldsymbol{s}$ form a solution set S_c, and $\boldsymbol{r}$ is the reference point. To approximate the hypervolume contribution of the solution $\boldsymbol{s}$, R2-HVC uses a vector set Λ (i.e., $\Lambda = \{\boldsymbol{v}_1, \boldsymbol{v}_2\}$ in Fig. 1) to detect the boundary of the hypervolume contribution region of $\boldsymbol{s}$ (i.e., shaded region in Fig. 1). The average length of the line segment for each vector from $\boldsymbol{s}$ to the boundary (e.g., $d_{1,1}$ for $\boldsymbol{v}_1$ and $d_{2,2}$ for $\boldsymbol{v}_2$) is used as the hypervolume contribution approximation for $\boldsymbol{s}$. Thus, the computation time and the approximation quality directly depends on the number of vectors in Λ.

For each vector, to obtain the corresponding line segment (e.g., $d_{1,1}$ for $\boldsymbol{v}_1$ in Fig. 1), we calculate the length of each of all line segments determined by other solutions in S_c and the reference point (e.g., $d_{1,1}$, $d_{2,1}$ and $d_{\boldsymbol{r},1}$ in Fig. 1). The line segment with the minimum length (e.g., $d_{1,1}$ for v_1) is the corresponding line segment and is used for hypervolume contribution approximation.

To approximate the hypervolume contribution of each candidate solution $\boldsymbol{s}_i \in S_c$ to the subset S_{sub} (i.e., $HVC(\boldsymbol{s}_i, S_{sub}, \boldsymbol{r})$), we need to calculate the length of each of all related line segments and store them in a matrix M_i as

$$M_i = \begin{bmatrix} d^i_{1,1} & d^i_{1,2} & \cdots & d^i_{1,|\Lambda|} \\ d^i_{2,1} & d^i_{2,2} & \cdots & d^i_{2,|\Lambda|} \\ \vdots & \vdots & \ddots & \vdots \\ d^i_{|S_{sub}|,1} & d^i_{|S_{sub}|,2} & \cdots & d^i_{|S_{sub}|,|\Lambda|} \\ d^i_{\boldsymbol{r},1} & d^i_{\boldsymbol{r},2} & \cdots & d^i_{\boldsymbol{r},|\Lambda|} \end{bmatrix}. \tag{2}$$

In this matrix M_i, each row refers to the corresponding solution in S_{sub} or the reference point $\boldsymbol{r}$, and each column refers to the corresponding vector in the vector set Λ. The hypervolume contribution of $\boldsymbol{s}_i$ is approximated by the average of the minimum value in each column (i.e., $\min(d^i_j) = \min\{d^i_{1,j}, ..., d^i_{|S_{sub}|,j}, d^i_{\boldsymbol{r},j}\}$) as $(\min(d^i_1) + \ldots + \min(d^i_{|\Lambda|}))/|\Lambda|$. For more details, see [21].

Greedy approximated HSS (GAHSS [18]) In GAHSS, a tensor $T_{\min}$ is used to calculate the approximated hypervolume contribution of each candidate solution. Its structure is

$$T_{\min} = \begin{bmatrix} \min(d^1_1) & \min(d^1_2) & \cdots & \min(d^1_{|\Lambda|}) \\ \vdots & \vdots & \ddots & \vdots \\ \min(d^{|S_c|}_1) & \min(d^{|S_c|}_2) & \cdots & \min(d^{|S_c|}_{|\Lambda|}) \end{bmatrix}. \tag{3}$$

Each row of $T_{\min}$ refers to the corresponding candidate solution (from M_i in Eq. (2)), and each column of $T_{\min}$ refers to the corresponding vector. For a candidate solution $\boldsymbol{s}_i \in S_c$, each value $\min(d^i_j)$ in $T_{\min}$ is the minimum value of the j-th column of M_i (i.e., $\min(d^i_j) = \min\{d^i_{1,j}, \ldots, d^i_{|S_{sub}|,j}, d^i_{\boldsymbol{r},j}\}$). Then, each row of $T_{\min}$ is used to approximate the hypervolume contribution of each candidate solution in S_c.

Algorithm 2: Greedy approximated HSS

```
input  : $S_c$ (candidate solution set), $k$ (selected subset size), $\Lambda$ (vector set)
output: $S_{sub}$ (selected subset)
begin
1   $S_{sub} \longleftarrow \emptyset$;
2   Calculate the tensor $T_{\min}$ in (3) using the reference point $\boldsymbol{r}$ for all
      candidate solutions in $S_c$ and all vectors in $\Lambda$;
3   while $|S_{sub}| < k$ do
4       Approximate the hypervolume contribution to $S_{sub}$ of each candidate
          solution in $S_c$ using $T_{\min}$;
5       Select the best candidate solution $\boldsymbol{a}$ with the largest hypervolume
          contribution from $S_c$;
6       $S_{sub} \longleftarrow S_{sub} \cup \{\boldsymbol{a}\}$; $S_c \longleftarrow S_c \backslash \{\boldsymbol{a}\}$;
7       Calculate the tensor $T$ in (4) using $\boldsymbol{a}$ for all candidate solutions in $S_c$
          and all vectors in $\Lambda$;
8       Update $T_{\min}$ using $T_{\min}$ and $T$;
```

When a new solution $\boldsymbol{a}$ is added to the subset S_{sub}, we first calculate a tensor T using $\boldsymbol{a}$ for all candidate solutions in S_c and all vectors in Λ as:

$$T = \begin{bmatrix} d^1_{\boldsymbol{a},1} & d^1_{\boldsymbol{a},2} & \cdots & d^1_{\boldsymbol{a},|\Lambda|} \\ \vdots & \vdots & \ddots & \vdots \\ d^{|S_c|}_{\boldsymbol{a},1} & d^{|S_c|}_{\boldsymbol{a},2} & \cdots & d^{|S_c|}_{\boldsymbol{a},|\Lambda|} \end{bmatrix}. \tag{4}$$

Similar to $T_{\min}$, each row of T refers to the corresponding candidate solution, and each column of T refers to the corresponding vector. The i-th row of T is a new row of M_i: $(d^i_{\boldsymbol{a},1}, \ldots, d^i_{\boldsymbol{a},|\Lambda|})$. In this manner, each element of $T_{\min}$ is updated as $\min(d^i_j) = \min\{\min(d^i_j), d^i_{a,j}\}$.

The framework of GAHSS is shown in Algorithm 2. As in GI-HSS, we first initialize the subset S_{sub} as an empty set. Then, the tensor $T_{\min}$ is initialized as the tensor using the reference point $\boldsymbol{r}$ for each candidate solution i and each vector j (i.e., $\min(d^i_j) = d^i_{\boldsymbol{r},j}$, where $d^i_{\boldsymbol{r},j}$ is an element of M_i). In each iteration, we first use $T_{\min}$ to calculate the approximated hypervolume contribution of each candidate solution. Then, the best candidate solution with the largest approximated hypervolume contribution is added to S_{sub}, and the tensor T is calculated based on the newly added solution. Finally, the tensor $T_{\min}$ is updated using $T_{\min}$ and T.

3 Proposed Two-Stage Greedy Approximated HSS

As shown in Algorithm 2, GAHSS calculates the tensor T using the newly added solution $\boldsymbol{a}$ for all candidate solutions in S_c and all vectors in Λ in each iteration. Thus, its computation time can be unacceptably large when the size of the candidate set is huge. To address this issue, we propose a two-stage GAHSS (TGAHSS) method in this paper. In each iteration of TGAHSS, we select a small number of promising candidate solutions from S_c in the first stage. Then, we select a single candidate solution from them in the second stage. The basic idea of the proposed two-stage method is to use a different vector set for hypervolume contribution approximation in each stage. By using a small vector set (i.e., only a small number of vectors) in the first stage, we can significantly decrease the computation time without severely degrading the quality of the selected subset.

We need two tensors $T^1_{\min}$ and $T^2_{\min}$ in TGAHSS for the first and second stages, respectively. Let the two vector sets for the first and second stages be Λ_1 and Λ_2 (where $|\Lambda_1| > |\Lambda_2|$), respectively. The number of candidate solutions used in the second stage is n_2. $T^1_{\min}$ and $T^2_{\min}$ in TGAHSS are described as:

$$T^1_{\min} = \begin{bmatrix} \min(d^1_1) & \cdots \min(d^1_{|\Lambda_1|}) \\ \vdots & \ddots & \vdots \\ \min(d^{|S_c|}_1) & \cdots \min(d^{|S_c|}_{|\Lambda_1|}) \end{bmatrix}, \quad T^2_{\min} = \begin{bmatrix} \min(d^1_1) & \cdots \min(d^1_{|\Lambda_2|}) \\ \vdots & \ddots & \vdots \\ \min(d^{|S_c|}_1) & \cdots \min(d^{|S_c|}_{|\Lambda_2|}) \end{bmatrix}. \quad (5)$$

Each row of $T^1_{\min}$ and $T^2_{\min}$ refers to the corresponding candidate solution, and each column of $T^1_{\min}$ and $T^2_{\min}$ refers to the corresponding vector in Λ_1 and Λ_2, respectively. In each iteration of TGAHSS, we update the entire $T^1_{\min}$ and only a small part of $T^2_{\min}$.

The framework of TGAHSS is shown in Algorithm 3. Different from GAHSS, we need to initialize two tensors $T^1_{\min}$ and $T^2_{\min}$ at the beginning (Line 2 in Algorithm 3). In each iteration, we first use $T^1_{\min}$ to roughly approximate the hypervolume contribution of each candidate solution and select n_2 promising candidate solutions (Lines 4-5 in Algorithm 3). In the second stage, we only need to update a small part of $T^2_{\min}$, instead of the entire $T^2_{\min}$. That is, only n_2 rows of $T^2_{\min}$, which are related to the n_2 promising candidate solutions, need to be updated. After that, the hypervolume contribution of each of the n_2 candidate solutions is approximated (using much more vectors in the second stage than those in the first stage: $|\Lambda_1| > |\Lambda_2|$) from the n_2 rows of $T^2_{\min}$ (Lines 6-7 in Algorithm 3). Then, the best solution $\boldsymbol{a}$ with the largest approximated hypervolume contribution is selected from the n_2 promising candidate solutions, and added to the subset S_{sub} (Line 9 in Algorithm 3). Finally, the first tensor $T^1_{\min}$ is updated using the newly added solution $\boldsymbol{a}$ (Lines 10, 11 in Algorithm 3).

4 Experimental Results

In this section, we first examine the proposed two-stage greedy approximated HSS (TGAHSS) method under different parameter settings. Then, the proposed method is compared with two state-of-the-art methods: the greedy approximated HSS (GAHSS [18]) and the lazy greedy inclusion HSS (LGI-HSS [17]).

Algorithm 3: Two-stage greedy approximated HSS

```
input  : $S_c$ (candidate solution set), $k$ (selected subset size), $\Lambda_1$ (first-stage
         vector set), $\Lambda_2$ (second-stage vector set)
output: $S_{sub}$ (selected subset)
begin
1    $S_{sub} \longleftarrow \emptyset$;
2    Calculate the tensor $T^1_{\min}$ and $T^2_{\min}$ in (5) using the reference point $\boldsymbol{r}$ for
       all candidate solutions in $S_c$ and all vectors in $\Lambda_1$ and $\Lambda_2$, respectively;
3    while $|S_{sub}| < k$ do
4        /* First stage */
5        Approximate the hypervolume contribution to $S_{sub}$ of each candidate
           solution in $S_c$ using $T^1_{\min}$;
6        Select $n_2$ candidate solutions with the largest hypervolume
           contributions from $S_c$;
7        /* Second stage */
8        Update the corresponding $n_2$ rows of $T^2_{min}$ which are related to the n2
           candidate solutions;
9        Approximate the hypervolume contribution of each of the $n_2$ candidate
           solutions to $S_{sub}$ using the corresponding rows of $T^2_{min}$;
10       Select the best solution $\boldsymbol{a}$ with the largest approximated hypervolume
           contribution from the $n_2$ candidate solutions;
11       $S_{sub} \longleftarrow S_{sub} \cup \{\boldsymbol{a}\}$; $S_c \longleftarrow S_c \backslash \{\boldsymbol{a}\}$;
12       /* Update of $T^1_{min}$ */
13       Calculate the tensor $T^1$ using $\boldsymbol{a}$ for all the candidate solutions and all
           vectors in $\Lambda_1$;
14       Update $T^1_{\min}$ using $T^1_{\min}$ and $T^1$;
```

4.1 Experimental Settings

We use eight candidate solution sets in [24][1], which are generated in the following manner. First, under the termination condition of 100,000 solution evaluations, NSGA-III [25] is applied to eight test problems: DTLZ1-2 [26] and Minus-DTLZ1-2 [27] with five and ten objectives (i.e., $m = 5, 10$). Next, all examined solutions (i.e., 100,000 solutions) are stored for each test problem. Then, all non-dominated solutions among the stored solutions are used as a candidate solution set for each test problem. The size of each candidate solution set (i.e., n) is shown in Table 1.

The selected subset size k is set to 100. As suggested in [28], the reference point for hypervolume subset selection is specified as $(1 + 1/H) \times nadir$ where $nadir$ is the estimated nadir point of the candidate set and $H = 4, 2$ for $m = 5, 10$, respectively. For performance evaluation, the reference point is set to $(1 + 1/H) \times trueNadir$ where $trueNadir$ is the true nadir point of each test problem.

In the first stage of the proposed TGAHSS method, the first vector in the vector set Λ_1 is specified as $(1/\sqrt{m}, \ldots, 1/\sqrt{m})$. Other vectors in Λ_1 are gen-

[1] https://github.com/HisaoLabSUSTC/BenchSS

Table 1. Size of each candidate solution set

Problem	$m = 5$	$m = 10$
DTLZ1	29,194	30,194
DTLZ2	45,605	62,601
Minus-DTLZ1	35,798	76,701
Minus-DTLZ2	48,741	85,631

erated using the UNV method [30]. In the second stage, the vector set Λ_2 is generated using the UNV method as suggested in [29]. In the GAHSS method, the vector set is also generated by the UNV method as in its original paper [18]. Each HSS method is executed 21 times independently.

All experiments are performed on a machine with AMD Ryzen Threadripper 3990X 64-Core Processor 2.90 GHz and Windows 10 Pro.

4.2 Performance of TGAHSS under Different Parameter Settings

In the proposed TGAHSS, there are three parameters: the number of first-stage vectors $|\Lambda_1|$, the number of second-stage vectors $|\Lambda_2|$, and the number of second-stage solutions n_2. We set the number of the second-stage vectors $|\Lambda_2|$ as $|\Lambda_2| = 100$, which is the same setting as in GAHSS. In this subsection, we examine the sensitivity of the performance of TGAHSS to the other two parameters. For the number of first-stage vectors $|\Lambda_1|$, we examine the settings of $|\Lambda_1| = \{1, 2, 10, 20, 30, 40, 50\}$. For the number of second-stage solutions n_2, we examine the settings of $n_2 = \{1, 2, 5, 10, 20, 50, 100, 500\}$.

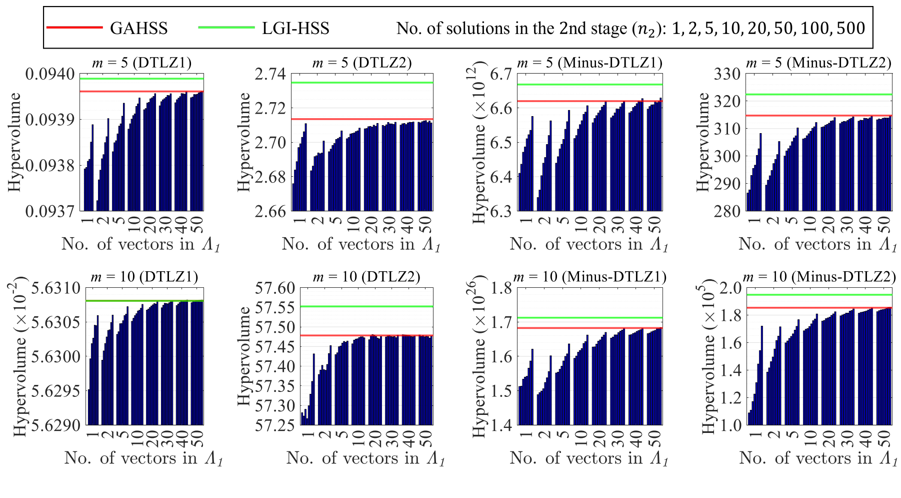

Fig. 2. Hypervolume of TGAHSS with different parameter settings compared to GAHSS and LGI-HSS.

Table 2. Hypervolume of the subsets selected by TGAHSS$^{\text{UNV}}$ and TGAHSS. Λ_1 in TGAHSS$^{\text{UNV}}$ has a single vector generated by the UNV method.

Data Shape	m	TGAHSS$^{\text{UNV}}$	TGAHSS	TGAHSS/TGAHSS$^{\text{UNV}}$
DTLZ1	5	9.3800E-2	9.3888E-2 (+)	1.0009
	10	5.6300E-2	5.6306E-2 (+)	1.0000
DTLZ2	5	2.6693E+0	2.7109E+0 (+)	1.0156
	10	5.7368E+1	5.7431E+1 (+)	1.0011
Minus-DTLZ1	5	6.2018E+12	6.5753E+12 (+)	1.0602
	10	1.4123E+26	1.6209E+26 (+)	1.1477
Minus-DTLZ2	5	2.9341E+2	3.0824E+2 (+)	1.0506
	10	1.5589E+5	1.7201E+5 (+)	1.1034
(+/-/≈)			(8/0/0)	

Hypervolume of the selected subset Fig. 2 shows the average hypervolume of the subsets selected by TGAHSS with different parameter settings, which are compared with the results selected by GAHSS (i.e., red lines) and LGI-HSS (i.e., green lines). The horizontal axis of each figure is the number of first-stage vectors (i.e., $|\Lambda_1|$). For each specification of $|\Lambda_1|$, experimental results obtained by various specifications of the number of second-stage solutions (i.e., $n_2 = \{1, 2, 5, 10, 20, 50, 100, 500\}$) are shown as a group of bars.

As shown in Fig. 2, the hypervolume of the selected subset clearly increases as the number of first-stage vectors increases (i.e., as $|\Lambda_1|$ increases). When $|\Lambda_1|$ is small (e.g., the left-most group of bars for $|\Lambda_1| = 1$), the hypervolume of the subset selected by TGAHSS is significantly improved by increasing the number of solutions in the second stage (i.e., by increasing n_2). In Fig. 2, the hypervolume of the subset selected by TGAHSS is slightly worse than that selected by GAHSS (i.e., red lines) when $|\Lambda_1| = 1$ and $n_2 = 500$ (i.e., the right-most bar in the left-most bar group in each figure).

In our experiments, the first vector of Λ_1 is specified as $(1/\sqrt{m}, ..., 1/\sqrt{m})$. This is because much better results are obtained from this vector than the randomly specified first vector generated by the UNV method. In Table 2, TGAHSS is compared with its variant TGAHSS$^{\text{UNV}}$ under the setting of $|\Lambda_1| = 1$ and $n_2 = 500$ (i.e., the setting of the right-most bar in the left-most bar group in each figure in Fig. 2). TGAHSS uses the vector $(1/\sqrt{m}, ..., 1/\sqrt{m})$ as Λ_1 and TGAHSS$^{\text{UNV}}$ uses the UNV method to generate Λ_1. It is clear in Table 2 that TGAHSS outperforms TGAHSS$^{\text{UNV}}$.

Computation time of TGAHSS As shown in Fig. 3, the computation time of TGAHSS strongly depends on the number of first-stage vectors (i.e., $|\Lambda_1|$). When $|\Lambda_1| = 50$ (i.e., the right-most bar group in each figure in Fig. 3), the computation time of TGAHSS is about 1/3 less than that of GAHSS. However, when $|\Lambda_1| = 1$ (i.e., the left-most bar group in each figure in Fig. 3), TGAHSS is about ten times faster than GAHSS. This is because the computation time of TGAHSS in the first stage is much larger than that in the second stage when the number of candidate solutions is very large. We need to approximate the hypervolume contribution of all candidate solutions (e.g., 85,631 solutions for Minus-DTLZ1) in the first stage whereas we handle only a small number of candidate solutions in the second stage (i.e., up to 500 solutions in Fig. 3). Thus, the computation time of TGAHSS strongly depends on the number of first-stage vectors (i.e., $|\Lambda_1|$).

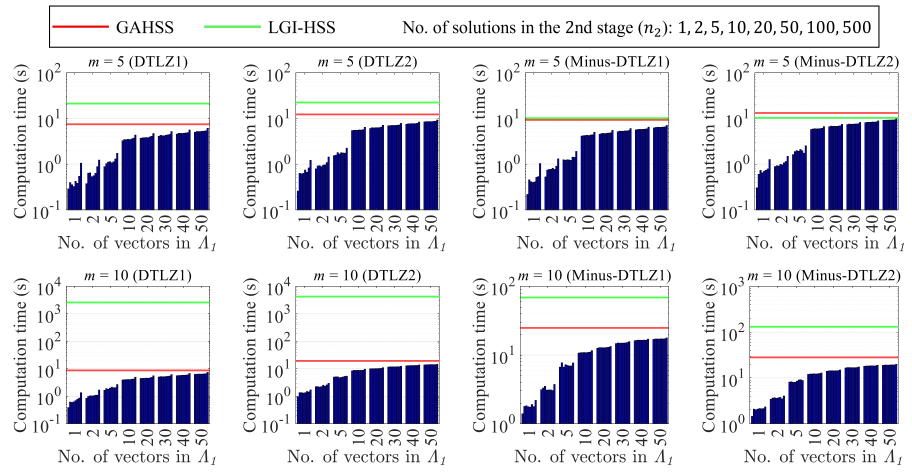

Fig. 3. Computation time of TGAHSS with different parameter settings compared to GAHSS and LGI-HSS.

It is also shown in Fig. 3 that the specifications of the number of the second-stage solutions (i.e., different bars in each bar group in each figure of Fig. 3) have no strong effect on the computation time of TGAHSS. This is because the computation time of the first stage is much larger than that of the second stage. In Fig. 2 and Fig. 3, we can observe that TGAHSS (with $|\Lambda_1| = 1$ and $n_2 = 500$) can obtain a slightly worse subset using a much smaller computation time compared to GAHSS. We use this setting in the next subsection.

4.3 Comparison with State-of-the-Art Methods

In this section, we compare the proposed method with the two most efficient greedy HSS methods: the greedy approximated HSS method (GAHSS [18]) and the lazy greedy inclusion HSS method (LGI-HSS [17]). LGI-HSS is the most efficient greedy HSS method with exact hypervolume contribution calculation. Since GAHSS uses the approximate calculation, GAHSS is faster than LGI-HSS but its selected subset is worse than the subset selected by LGI-HSS.

Each algorithm is applied to each test problem 21 times. Average results over 21 runs are summarized in Fig. 4. The random method (black point in Fig. 4) where a subset is randomly selected from the candidate solution set is also used as a baseline for comparison. In Fig. 4, the proposed TGAHSS method

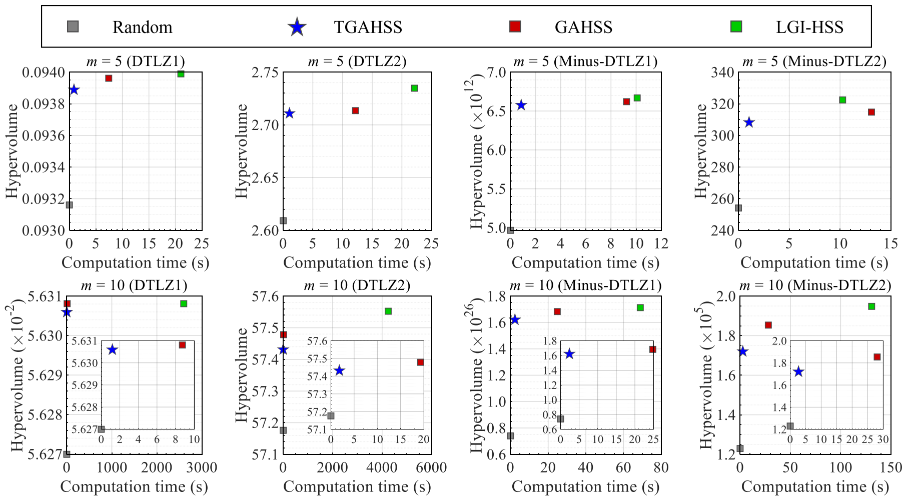

Fig. 4. Hypervolume and computation time of GAHSS, TGAHSS and LGI-HSS.

always locates around the knee region [31] of the trade-off curve generated by connecting the results obtained by the four methods. That is, TGAHSS is much better than GAHSS and LGI-HSS with respect to the computation time and slightly worse than GAHSS and LGI-HSS with respect to the subset quality in Fig. 4. For example, for 5-objective DTLZ2, the hypervolume of the subset selected by TGAHSS is slightly worse than that of GAHSS but TGAHSS is about ten times faster than GAHSS.

5 Conclusion

In this paper, we proposed a two-stage greedy approximated hypervolume subset selection method (TGAHSS) for large-scale candidate solution sets (e.g., 50,000 solutions). Experimental results showed that the proposed TGAHSS method is much faster than the two state-of-the-art greedy HSS methods. The quality of the subset selected by TGAHSS is slightly worse than those selected by GAHSS and LGI-HSS in terms of the hypervolume. In the future, we will examine the use of TGAHSS to generate an initial subset for local search HSS.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 62002152, 61876075), Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), The Stable Support Plan Program of Shenzhen Natural Science Fund (Grant No. 20200925174447003), Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531).

References

1. Ishibuchi, H., Pang, L. M., Shang, K.: A new framework of evolutionary multi-objective algorithms with an unbounded external archive. In: Proc. Eur. Conf. Artif. Intell., pp. 283–290. Santiago de Compostela, Spain (2020).
2. Fieldsend, J. E., Everson, R. M., Singh, S.: Using unconstrained elite archives for multiobjective optimization. IEEE Trans. Evol. Comput. 7(3), 305–323 (2003).
3. Schütze, O., Coello Coello, C. A., Mostaghim, S., Talbi, E.-G., Dellnitz, M.: Hybridizing evolutionary strategies with continuation methods for solving multi-objective problems. J. Eng. Optim. 40(5), 383–402 (2008).
4. Brockhoff, D., Tran, T.-D., Hansen, N.: Benchmarking numerical multiobjective optimizers revisited. In: Proc. Conf. Genet. Evol. Comput., pp. 639–646. Madrid Spain (2015).
5. Tanabe, R., Ishibuchi, H., Oyama, A.: Benchmarking multi- and many-objective evolutionary algorithms under two optimization scenarios. IEEE Access 5, 19597–19619 (2017).
6. Ishibuchi, H., Setoguchi, Y., Masuda, H., Nojima, Y.: How to compare many-objective algorithms under different settings of population and archive sizes. In: Proc. IEEE Congr. Evol. Comput., pp. 1149–1156. Vancouver, BC, Canada (2016).
7. Ishibuchi, H., Pang, L. M., Shang, K.: Difficulties in fair performance comparison of multi-objective evolutionary algorithms. IEEE Comput. Intell. Mag. 17(1), 86–101 (2022).
8. Bringmann, K., Friedrich, T., Klitzke, P.: Generic postprocessing via subset selection for hypervolume and epsilon-indicator. In: Proc. Int. Conf. Parallel Probl. Solv. Nat., pp. 518–527. Ljubljana, Slovenia (2014).
9. Bezerra, L. C. T., López-Ibáñez, M., Stützle, T.: Archiver effects on the performance of state-of-the-art multi- and many-objective evolutionary algorithms. In: Proc. Conf. Genet. Evol. Comput., pp. 620–628. Prague Czech Republic (2019).
10. Li, M., Yao, X.: An empirical investigation of the optimality and monotonicity properties of multiobjective archiving methods. In: Proc. Evol. Multi-Criter. Optim., pp. 15–26. East Lansing, MI, USA (2019).
11. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. IEEE Trans. Evol. Comput. 3(4), 257–271 (1999).
12. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., Fonseca, V.G. da: Performance assessment of multiobjective optimizers: An analysis and review. IEEE Trans. Evol. Comput. 7(2), 117–132 (2003).
13. Shang, K., Ishibuchi, H., He, L., Pang, L. M.: A survey on the hypervolume indicator in evolutionary multiobjective optimization. IEEE Trans. Evol. Comput. 25(1), 1–20 (2021).
14. Bradstreet, L., Barone, L., While, L.: Maximising hypervolume for selection in multi-objective evolutionary algorithms. In: Proc. IEEE Congr. Evol. Comput., pp. 1744–1751. Vancouver, BC, Canada (2006).
15. Bradstreet, L., While, L., Barone, L.: Incrementally maximising hypervolume for selection in multi-objective evolutionary algorithms. In: Proc. IEEE Congr. Evol. Comput., pp. 3203–3210. Singapore (2007).
16. Jiang, S., Zhang, J., Ong, Y., Zhang, A. N., Tan, P. S.: A simple and fast hypervolume indicator-based multiobjective evolutionary algorithm. IEEE Trans. Cybern. 45(10), 2202–2213 (2015).

17. Chen, W., Ishibuchi, H., Shang, K.: Fast greedy subset selection from large candidate solution sets in evolutionary multi-objective optimization. IEEE Trans. Evol. Comput. 26(4), 750–764 (2022).
18. Shang, K., Ishibuchi, H., Chen, W.: Greedy approximated hypervolume subset selection for many-objective optimization. In: Proc. Genet. Evol. Comput. Conf., pp. 448–456. Lille France (2021).
19. Nemhauser, G. L., Wolsey, L. A., Fisher, M. L.: An analysis of approximations for maximizing submodular set functions–I. Math. Program. 14(1), 265–294 (1978).
20. Ulrich, T., Thiele, L.: Bounding the effectiveness of hypervolume-based $(\mu + \lambda)$-archiving algorithms. In: Proc. Int. Conf. Learning Intell. Optim., pp. 235–249. Paris, France (2012).
21. Shang, K., Ishibuchi, H., Ni, X.: R2-based hypervolume contribution approximation. IEEE Trans. Evol. Comput. 24(1), 185–192 (2020).
22. Nan, Y., Shang, K., Ishibuchi, H., He, L.: Improving local search hypervolume subset selection in evolutionary multi-objective optimization. In: Proc. IEEE Int. Conf. Syst. Man Cybern., pp. 751–757. Melbourne, Australia (2021).
23. Nan, Y., Shang, K., Ishibuchi, H., He, L.: A two-stage hypervolume contribution approximation method based on R2 indicator. In: Proc. IEEE Congr. Evol. Comput., pp. 2468–2475. Kraków, Poland (2021).
24. Shang, K., Shu, T., Ishibuchi, H., Nan, Y., Pang, L. M.: Benchmarking subset selection from large candidate solution sets in evolutionary multi-objective optimization. arXiv:2201.06700.
25. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints. IEEE Trans. Evol. Comput. 18(4), 577–601, (2013).
26. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multiobjective optimization. In: Evol. Multi. Optim., pp. 105–145. A. Abraham, L. Jain, and R. Goldberg, Eds. London: Springer-Verlag (2005).
27. Ishibuchi, H., Setoguchi, Y., Masuda, H., Nojima, Y.: Performance of decomposition-based many-objective algorithms strongly depends on Pareto front shapes. IEEE Trans. Evol. Comput. 21(2), 169–190 (2017).
28. Ishibuchi, H., Imada, R., Setoguchi, Y., Nojima, Y.: How to specify a reference point in hypervolume calculation for fair performance comparison. Evol. Comput. 26(3), 411–440 (2018).
29. Nan, Y., Shang, K., Ishibuchi, H.: What is a good direction vector set for the R2-based hypervolume contribution approximation. In: Proc. Conf. Genet. Evol. Comput., pp. 524–532. Lisbon, Portugal (2020).
30. Deng, J., Zhang, Q.: Approximating hypervolume and hypervolume contributions using polar coordinate. IEEE Trans. Evol. Comput. 23(5), 913–918 (2019).
31. Zhang X., Tian Y., Jin Y.: A knee point-driven evolutionary algorithm for many-objective optimization. IEEE Trans. Evol. Comput. 19(6), 761-776 (2015).