# Worst-case Conditional Hardness and Fast Algorithms with Random Inputs for Non-dominated Sorting

Sorrachai Yingchareonthawornchai[*]
Institute for Clarity in Documentation
Dublin, Ohio
webmaster@marysville-ohio.com

Proteek Chandan Roy[†]
Institute for Clarity in Documentation
Dublin, Ohio
trovato@corporation.com

Bundit Laekhanukit[‡]
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Eric Torng
Inria Paris-Rocquencourt
Rocquencourt, France

Kalyanmoy Deb
Rajiv Gandhi University
Doimukh, Arunachal Pradesh, India

## ABSTRACT

We study computational complexity aspect of non-dominated sorting problem (NDS): Given a set $P$ of $n$ points in $\mathbb{R}^m$, and for each point $p \in P$, compute $\ell$ the length of longest domination chain $p_1 \succ p_2 \succ \cdots \succ p_\ell = p$ where $x$ dominates $y$ (denoted as $x \succ y$) if $x$ is not larger than $y$ in every coordinate. Non-dominated sorting (NDS) has emerged as a critical component for multi-objective optimization problems (MOPs). For small dimensions, $\Theta(n \log n)$-time is known for $m \leq 3$. For a fixed small $m$, the best bound is $O(n \log^{m-2} n \log\log n)$. For higher dimensions, the first $O(mn^2)$-time is known in 2002. There is no improvement since then.

In this paper, we argue that the running time $O(mn^2)$ is optimal by proving a matching conditional lower bound: for any constant $\epsilon > 0$, and $\omega(\log n) \leq m \leq n^{o(1)}$, there is no $O(mn^{2-\epsilon})$-time algorithm for NDS unless a popular conjecture in fine-grained complexity theory is false. To complete our results, we present an algorithm with average-case running time $O(mn + n^2/m + n \log^2 n)$ on the inputs that are drawn from uniform distribution.

## CCS CONCEPTS

• **Theory of computation → Algorithm design techniques**;

## KEYWORDS

Non-dominated Sorting, Multi-objective Optimization, Complexity Theory

**ACM Reference Format:**
Sorrachai Yingchareonthawornchai, Proteek Chandan Roy, Bundit Laekhanukit, Eric Torng, and Kalyanmoy Deb. 2020. Worst-case Conditional Hardness and Fast Algorithms with Random Inputs for Non-dominated Sorting . In *Proceedings of the Genetic and Evolutionary Computation Conference 2020 (GECCO '20).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[*]The secretary disavows any knowledge of this author's actions.
[†]Dr. Trovato insisted his name be first.
[‡]This author is the one who did all the really hard work.

## 1 INTRODUCTION

We study the computational complexity of the non-dominated sorting problem (NDS). Let $P$ be a set of $n$ points in $\mathbb{R}^m$. We say that a point $p$ *dominates*[1] another point $q$, denoted by $p \succ q$ if $p_i \leq q_i$ for positive $i \leq m$, i.e., $p$ is no larger than $q$ in every coordinate and $p \neq q$. Note that it is possible that $p$ and $q$ are mutually non-dominated to each other. A point $p$ is *non-dominated* w.r.t. $P$ if $p$ is not dominated by any other points in $P$. Given a set of $P$ points, the non-dominated sorting problem asks to compute the *rank function* $R : P \to \mathbb{N}$ defined as follows: $R(p) = 1$ if $p$ is a non-dominated point, and $R(p) = 1 + \max\{R(q) : q \succ p\}$, otherwise. The rank function is also known as the layer number and front number, and the non-dominated sorting is also known as layer-of-maxima. In the context of multi-objective optimization, it is equivalent to say a point or a solution, and to say an objective or an coordinate.

Non-dominated sorting has emerged as a critical component for multi-objective optimization problems (MOPs). In contrast to single objective optimization where we try to find the best possible solution, the desired result of an MOP is typically a set of Pareto-optimal solutions that reflect the trade-offs among different objectives. An NDS algorithm is a computational bottleneck for multi- and many-objective evolutionary algorithms (MOEAs). Other key operations such as crossover, mutation or tournament selection are typically fast (linear time) compared to an NDS algorithm. Stated another way, speeding up non-dominated sorting will allow MOEAs to run with larger populations, more generations, and more objectives leading to better solutions for most problem domains.

The non-dominated sorting problem is completely solved when $m = 2$ or 3 with a worst-case time complexity of $\Theta(n \log n)$ [7, 26, 33, 34]. For a fixed $m > 2$, $O(n \log^{m-1} n)$-time algorithms are known using divide-and-conquer (D&C), often referred to as Jensen's sort [1, 3, 4, 15, 25, 28]. Recently, an improved $O(n \log^{m-2} n \log\log n)$-time algorithm is presented by [8]. However, the algorithms quickly become intractable for even moderate $m$. For general $m$, the first $O(mn^2)$-time algorithm is due to Deb *et al.* [12]. Since then there

[1]we use notion of $p_i \leq q_i$ to be consistent with the MOEA community where, in the context of minimization, the point $p$ is "better" when every coordinate is smaller than $q$.

have been several algorithms achieving the same worst-case bounds, but focusing on practical running time in various instances [14, 18, 32, 38, 41, 42, 46, 47]. Until now, the $O(mn^2)$-time bound has stood for almost two decades for general $m$.

### 1.1 Our Results

We argue that the running time $O(mn^2)$ is already optimal assuming the *Hitting Set Conjecture*. We first define Hitting Set Problem (HS): Given two families of sets $A$ and $B$ containing $n$ sets each over the universe $\{1, \dots, m\}$ where $m = \omega(\log n)$, decide if there exists a set $a \in A$ that intersects (hits) every set $b \in B$ at least one element.

We now state the conjecture.

**Hitting Set Conjecture.** For any constant $\epsilon > 0$, and $m$ where $\omega(\log n) \leq m < (\log n)^{O(1)}$, there is no $O(mn^{2-\epsilon})$-time algorithm for Hitting Set Problem.

We discuss why Hitting Set Conjecture is believable (and why complexity theorists do believe) in Section 3. A quick explanation is that it captures hardness of many other problems in similar way as the P vs. NP conjecture does. We now state our first result:

Theorem 1.1. *For any constant $\epsilon > 0$, and $m$ where $\Omega(\log n) < m < (\log n)^{O(1)}$, there is no $O(mn^{2-\epsilon})$-time algorithm for non-dominated sorting unless HS conjecture is false.*

This gives theoretical explanation that the lack of the improvement in last decades was that, in fact, the bound $O(mn^2)$ is (conditionally) optimal in the worst case. Given that the worst-case complexity of NDS is well understood, we turn our attention to the average-case complexity of NDS.

In fact, the average-case complexity of NDS is less understood. Average-case means we are interested in the expected running time over a uniform distribution of inputs. Only the special cases of the problem are known. One natural special case is when we just compute just first rank points. This problem is known as computing the maxima. In this case, a $O(mn)$-time algorithm is known due to Bentley *et al.* [1]. He *et al.* [20] consider a more general case where we output top $k$ ranks. For $m \in \{2, 3\}$, for any constant $\epsilon > 0$ depending on $m$, it is possible to compute first $n^{1/m-\epsilon}$ ranks in expected time $O(mn)$. For $m \geq 4$, it is possible to compute first $n^{1/(2m)-\epsilon}$ ranks in expected time $O(mn)$.

In this paper, we prove the first average-case complexity of NDS to be $o(mn^2)$ whenever $m = \omega(1)$ under the random input assumption: each coordinate is independent, and all $n!$ permutations are equally likely.

Theorem 1.2. *Under the random input assumption, there is an algorithm that takes as input a set of points $P$ in $\mathbb{R}^m$, and outputs the rank function (as defined in non-dominated sorting problem) in expected time $O(n^2/m + mn + n\log^2 n)$, which is linear when $m = \Omega(\sqrt{n})$.*

**Organization.** We discuss related work in MOEA communities in Section 2. Then, we provide a brief background for the new emerging field known as fine-grained complexity theory in Section 3. We prove Theorem 1.1 in Section 4. We prove Theorem 1.2 by describing algorithms which we call Minimum Multi-Sort (MMS) in Section 5, and analyzing its worst-case and average-case time complexity in Section 6. Finally, we conclude in Section 7.

## 2 RELATED WORK

Most MOEAs generate a new population of solutions from the current population where only the "best" solutions of the current population contribute to the next population. These MOEAs such as NSGA, NSGA2, SPEA2, PAES, PESA, EPCS, MOEA/DD, RVEA[10–12, 27, 30, 37, 40, 48, 49] use NDS to identify the "best" solutions of the current generation. How they define the best solution differs by algorithm. Some use all fronts; others use only the first front.

Apart from the area of multi-objective optimization, non-dominated sorting has been studied in other application areas such as gene selection and data clustering [6, 19, 21, 29, 36]. In these applications, they use only on the first front. In the next section, we discuss previous state-of-the-art solutions to this problem and mention our contribution to this field.

Srinivas and Deb provided the first non-dominated sorting algorithm in their MOEA named NSGA which ran in $O(mn^3)$ time and requires $O(n)$ space [40]. This was improved to $O(mn^2)$ time at the cost of using $O(n^2)$ space in the NSGA-II algorithm [12]. Several methods improved upon NSGA-II by eliminating some unnecessary comparisons by inferring some dominance relationships using the results of already completed comparisons and intelligently choosing which solutions to compare next. These include Tang *et al.*'s arena principle non-dominated sorting algorithm [41], Clymont and Keedwell's deductive sort [32], Wang and Yao's corner sort [42], and Fang *et al.*'s [14] domination tree– all of which run in $O(mn^2)$ time and use $O(n)$ space, in the worst case.

An alternative approach is to use divide-and-conquer (D&C), often referred to as Jensen's sort [1, 3, 4, 15, 25, 28]. For $m > 2$, D&C requires $O(n \log^{m-1} n)$ time which is good for small $m$ but quickly becomes intractable for even moderate $m$.

Zhang *et al.* identified the following key issue with almost every existing non-dominated sorting algorithm [46]: they work by computing each front in order. Zhang *et al.* presented an improved algorithm, ENS, that overcomes this issue by first sorting all the solutions using a single objective. Sorting requires $O(n \log n)$ time. They then process the solutions in this sorted order comparing each solution against the solutions located before its position in the sorted list to determine its exact front. Despite this clever optimization for ENS and the ability to eliminate half of the comparisons in the worst case, ENS still has a worst-case time complexity of $O(mn^2)$ and a space complexity of $O(n)$.

Several papers have proposed improvements to ENS by adding a data structure to capture non-domination information to more quickly identify when solutions in a front do not dominate each other. These efforts include Gustavasson *et al.*'s variant of a bucket $k$-dimensional tree ($k-d$ tree) [17] and Zhang *et al.*'s non-domination tree (ND-tree) [47].

Roy *et al.* proposed the best order sort (BOS) algorithm [38] which improves upon ENS by sorting the solutions by each objective and then only comparing the current solution against the solutions that are better than current solution's best objective (provided by partial rank). BOS requires more time for up front sorting $O(mn \log n)$ than ENS but is able to prune away more solution in later comparisons. Roy *et al.* showed that BOS performs well empirically in many different settings, but they did not provide any theoretical analysis of BOS performance. We build upon BOS and the ND-tree tree to provide a new NDS algorithm that we call Minimum Multi-sort or MMS.

Some approaches [13, 24, 31] deal with the problem of dynamic or online update of the non-dominated set. These algorithms require more time than static NDS algorithms since the addition or removal of one point may disrupt the non-domination structure.

## 3 FINE-GRAINED COMPLEXITY THEORY

The theory of NP-completeness is one of the most successful theory in complexity, which classifies computational problems into those that can be decided in polynomial time and those that might not admit a polynomial-time algorithm. This gives a notion of "easy" and "hard" problems, i.e., one may deem problems that admit polynomial-time algorithms *easy*, while the others are *hard*. However, an easy problem might not be as easy as it sounds. For example, although we can say that a problem tractable in $O(n^{100})$-time is easy theoretically, this running time is too impractical as it would take forever to solve a problem even on input of size $n = 2$. Therefore, one wish to have an algorithm that runs in reasonable ranges of time, e.g., quadratic, linear or even sublinear time. Is this always possible? To date, we know that some problems like *Sorting* admit no linear-time algorithm on comparison-based machines, while many fundamental problems like *All-Pairs Shortest Paths*, although we cannot prove it, does not seem to admit a subcubic-time algorithm. This motivates the study of running-time lower-bounds for easy problems, which is now emerged into the area of *fine-grained complexity*. Roughly speaking, the goal in fine-grained complexity is to study the best possible running times for solving "easy problems".

The development of fine-grained complexity theory follows the seminal results of Williams [43], which binds the running time of solving a polynomial-time solvable problem to that of the subexponential-time algorithm for solving *k-SAT*. To be specific Williams [43] shows that the existence of an algorithm that solves a fundamental problem like finding a furthest-pair of points in truly subquadratic-time (i.e., $O(n^{2-\epsilon})$ time for some constant $\epsilon > 0$) would imply a surprising algorithm that solves $k$-SAT faster than $2^{(1-\epsilon)n}$, for some constant $\epsilon > 0$ depending on $k$, which would contradict the believe that no such algorithm for $k$-SAT exists [9, 22, 23]. The conjecture on the running time lower bound of $k$-SAT is known as the *Strong Exponential-Time Hypothesis* (SETH) [9].

To date, many popular hypotheses have been raised as bases to prove conditional lower bounds of "easy" problems, e.g., the *All-Pair Shortest Paths* (APSP) conjecture, the *Orthogonal Vectors* (OV) conjecture, and the *Hitting Set* (HS) conjecture; please see [44, 45] and references therein. While many researchers do not have strong faith in SETH, most of them still believe that the APSP, OV and HS conjectures are likely to be true. One reason is that these conjectures imply that it is impossible to improve the running times of many fundamental problems in which the best running times are from trivial algorithms (and no improvement since then). For example, the HS conjecture [2] that we use in this paper implies that there is no truly subquadratic-time algorithm for computing the *radius* of a sparse graph [2] or computing *Earth Mover Distance* [35] of two sets of points. This (partially) answers the question why there have been no improvements over trivial algorithms for these two problems. The HS conjecture is also studied in [16].

## 4 CONDITIONAL HARDNESS OF NON-DOMINATED SORTING

Before we prove Theorem 1.1, we introduce the following intermediate problem.

**Bichromatic Binary Non-Dominating Problem (BBND).** Given two sets of points $A$ and $B$ where $|A| = |B| = n$, and each point is a vector over $\{0,1\}^m$ where $m = \Omega(\log n)$, decide if there exists a point $a \in A$ that is not dominated (i.e., there exists a positive $i \le m$ such that $a_i < b_i$) by any point $b \in B$.

Lemma 4.1 (Reduction from HS to BBND). *If BBND can be solved in time $T(m,n)$, then HS can be solved in time $O(T(m,n))$.*

Proof. By representing each set as a binary vector, we can describe an equivalent formulation of the hitting set problem in terms of two sets of vectors: Given two sets $A, B$ of vectors over $\{0,1\}^m$ where $m = \omega(\log n)$, decide if there exists a vector $a \in A$ such that for all $b \in B$, $\sum_{i=1}^m a_i \cdot b_i > 0$.

Now, given an input instance $(A, B)$ for HS, we create a new set $A' := \{a' : a \in A\}$ where $a'_i = 1$ if $a_i = 0$, and $a'_i = 0$ if $a_i = 1$ for $i \le m$. The instance for BBND is then $(A', B)$. Clearly, this construction takes linear time. We now prove the completeness and soundness of the reduction. For completeness, assume that $(A, B)$ is a yes-instance to HS. That is, there exists $a \in A$ such that for all $b \in B$, $\sum_{i=1}^m a_i \cdot b_i > 0$. This means there is a positive $i \le m$ such that $a_i = b_i = 1$. By construction (since we flip bitwise from $a$ to $a'$), for the same $i$, we have $a'_i = 0$, but $b_i = 1$, and thus $a'_i < b_i$. This holds for every $b \in B$. Therefore, this particular vector $a' \in A'$ is not dominated by any $b \in B$, which is a yes-instance for BBND . Next, we prove the soundness. Assume that $(A', B)$ is a yes-instance to BBND. So, there is a point $a' \in A'$ that is not dominated by any point $b \in B$. This means for each $b \in B$, there is a positive $i \le m$ such that $a'_i < b_i$. Since $a'$ and $b$ are binary vectors, the only possibility is when $a'_i = 0$ and $b_i = 1$. By construction, we have $a_i = 1$ and $b_i = 1$ for the same $i$. This holds for every $b \in B$. Hence, this vector $a$ has the property that $\sum_{i=1}^m a_i \cdot b_i > 0$ for all $b \in B$, which is a yes-instance for HS. □

Next we show that we can solve *BBND* by using a non-dominated sorting algorithm. In particular, it is enough to compute all first rank solutions.

Lemma 4.2. *The input $(A, B)$ for BBND is a yes-instance if and only if there exists $a \in A$ such that $R_{A \cup B}(a) = 1$ in the solution population $A \cup B$ where $R_{A \cup B}(p)$ is the rank of a solution $p$ in the population $A \cup B$.*

Proof. We first show the backward direction. Let $P := A \cup B$ be the solution population. Since there is $a \in A$ with $R_{A \cup B}(a) = 1$, the definition of rank implies that $a$ is not dominated by any point in the population $P$. In particular, $a$ is not dominated by any point $b \in B$. Therefore, $(A, B)$ is a yes-instance for BBND. We next prove the forward direction. Since $(A, B)$ is a yes-instance for BBND, there is $a \in A$ such that $a$ is not dominated by any point $b \in B$. If $R_{A \cup B}(a) = 1$, then we are done. We now assume otherwise. Consider the population $P := A \cup B$. Since $R_{A \cup B}(a) > 1$, we can trace back to any point with the first rank along the domination chain starting at $a$. Let $x$ be such a point. Clearly, $R_{A \cup B}(x) = 1$ and $x$ dominates $a$. It remains to show that $x \in A$. Suppose $x \in B$, we have that $a$ is dominated by $x$ that belongs to $B$, which is a contradiction. □

We now prove Theorem 1.1.

Proof of Theorem 1.1. By Lemma 4.1, it is enough to solve BBND problem in time $O(mn^{2-\epsilon})$. Suppose there is a $O(mn^{2-\epsilon})$-time algorithm for non-dominated sorting. We prove that we can solve BBND in time $O(mn^{2-\epsilon})$ as follows: Given an instance $(A, B)$, we compute the rank of all populations $A \cup B$, and then output yes if there is $a \in A$ with $R_{A \cup B}(a) = 1$, and no otherwise. The correctness follows immediately from Lemma 4.2. □

## 5 MINIMUM MULTI-SORT (MMS)

Our input $P$ is a set of solutions $\{s_j \in \mathbb{R}^m \mid 1 \le j \le n\}$ where $s_j^i$ is the value of solution $j$ in objective $i$. Our goal is to compute the rank of all solutions in $P$. We assume without loss of generality that solutions are unique but may have identical values in some objectives.

We divide the problem of ranking solutions into two phases, ordering and ranking. We first order all the solutions in $1 \le h \le m$ objectives. We discuss our choice of $h$ in our analysis section. We then extract the minimum unprocessed solution from each of our $h$ ordered objectives and rank that solution if it has not already been ranked until all solutions are ranked. We build upon Zhang *et al.*'s ENS method [46]. We improve upon ENS by ordering each solution in $h$ objectives. Compared to ENS, we spend more time in ordering but hopefully spend less time in ranking because we compare each solution against fewer other solutions.

### 5.1 Ordering Phase

In the ordering phase, we order the solutions in $P$ based on each objective $i$ for $1 \le i \le h$ using an ordering function $<_i$ which we define below. We first define the lexical order of solutions in $P$, denoted by $<_\ell$, using objectives 1 to $m$ in order as follows. For any two solutions $s_u$ and $s_v$ in $P$, let $k$ be the smallest integer such that $s_u^k \ne s_v^k$. If $s_u^k < s_v^k$, then $s_u <_\ell s_v$; else $s_v <_\ell s_u$. It follows that $<_\ell$ defines a total order on the solutions in $P$. Then, for $1 \le i \le h$ and any two solutions $s_u$ and $s_v$, $s_u <_i s_v$ if ($s_u^i < s_v^i$) or (($s_u^i = s_v^i$) and ($s_u <_\ell s_v$)); otherwise $s_v <_i s_u$. That is, we first order $s_u$ and $s_v$ by their values $s_u^i$ and $s_v^i$. If that does not resolve their order, we order $s_u$ and $s_v$ by their lexical order.

For each $1 \le i \le h$, we store the solutions $P$ ordered by $<_i$ in an ordering data structure $Q_i$ that supports two operations: (i) construct $Q_i$ given $P$ and $<_i$ and (ii) extract minimum which will be used during the ranking phase. We consider two standard data structures for $Q_i$. The first is a sorted linked list or sorted array which supports construction in $O(n \log n)$ time and extract minimum in $O(1)$ time. The second is a binary heap which supports construction in $O(n)$ time and extract minimum in $O(\log n)$ time. For simplicity, it is easier to think about the sorted linked list or sorted array, but the binary heap may be faster, especially when we process some but not all the solutions in $Q_i$ during the ranking phase.

### 5.2 Ranking Phase

We perform ranking in rounds. In a round, for objective $1 \le i \le h$, we process $q_i$ which is the minimum unprocessed solution from

**Algorithm 1:** Minimum Multi-Sort or MMS

```
Input   : Population $P = \{s_j \in \mathbb{R}^m \mid 1 \le j \le n\}$
Output : Ranking $R$ of solutions in $P$
// Ordering phase
1  $R \leftarrow \{\}$ // no solutions ranked yet
2  $Q_1 \leftarrow$ sort $P$ using lexical order ;
3  Initialize $L_i = \varnothing\ \forall i = 1$ to $n$// no solutions ranked
4  $Q_i \leftarrow Order(P, i),\ \forall i = 2$ to $n$// construct heap
5  for $j = 1$ to $n$ do
6      for $i = 1$ to $h$ do
7          Put $q_i \leftarrow ExtractMin(Q_i)$ in the sorted order in $Q_i$
8          if all $n$ solutions are extracted once then
9              $index \leftarrow i$
10             break out of both loops
11         end
12     end
13 end
14 $objSeq \leftarrow$ Find order of objectives. Use the reverse order till
       depth $index$ from $Q$, other objectives are randomly
       ordered// global
15 $C(P) \leftarrow m$// it counts # obj. to compare, global
16 $B \leftarrow$ Create binary tree with rank 1
// Ranking phase
17 while $|R| < n$ do
18     for $i = 1$ to $h$ do
19         $q_i \leftarrow ExtractTop(Q_i)$ // $Q_i$ is already sorted till
               $index$
20         $C(q_i) \leftarrow C(q_i) - 1$
21         if $q_i \notin R$ then
22             rank($q_i$) $\leftarrow$ INSERT($L_i, q_i, B$)
23             $R \leftarrow R \cup \{$rank($q_i$)$\}$
24         else
25             INSERTINTORANK($L_i, q_i$, rank($q_i$))
26         end
27     end
28 end
29 return $R$
```

$Q_i$ adding $q_i$ to a ranking data structure $L_i$ which organizes the processed solutions from $Q_i$ to facilitate fast ranking. There are two possibilities for how we process $q_i$. If $q_i$ is unranked ($q_i \notin R$), then we simultaneously rank $q_i$ and insert $q_i$ into $L_i$ and $R$. If $q_i$ is already ranked ($q_i \in R$), then we just insert $q_i$ into $L_i$. In both cases, we do not modify the ranks of already ranked items.

We first consider the case where $q_i$ is unranked ($q_i \notin R$). The key observation (due to Zhang *et al.* [46]) is that no solution $s \in P \setminus L_i$ can dominate $q_i$ because $q_i <_i s$ which means either $q_i^i < s^i$ or $q_i <_\ell s$. Thus, we only need to compute the rank of $q_i$ against the solutions in $L_i$. The exact details of how we compute this rank depends on the details of $L_i$. We assume that Insert($L_i$,$q_i$) will insert $q_i$ into $L_i$ while determining and returning $q_i$'s rank.

We next consider the case where $q_i$ is already ranked ($q_i \in R$) but was previously unprocessed in $Q_i$. In this case, we assume that InsertIntoRank($L_i$, $q_i$, rank($q_i$)) will correctly insert $q_i$ into $L_i$.

The algorithm can safely terminate if all solutions are ranked before $n$ rounds, so we only continue if there are unranked solutions ($|R| < n$). If the algorithm can terminate after relatively few rounds, say $n/h$, then the binary heap implementation of $Q_i$ may outperform the sorted array or sorted linked list implementation of $Q_i$.

We now describe a basic implementation of the ranking data structure $L$ using arrays of linked lists. Observe that solutions in $L$ can be partitioned into a list of solutions with the same rank. Let $L^k$ be the solutions with rank $k$, and $r$ be the maximum rank in $L$. We have that $L = L^1 \sqcup L^2 \sqcup \ldots \sqcup L^r$ where $\sqcup$ denotes disjoint union. So, $L^k$ can be indexed by $k$ using an array, and each $L^k$ can be a linked list of solutions with rank $k$.

To implement InsertIntoRank$(L, q, \text{rank}(q))$, we simply add the new solution $q$ into $L^{\text{rank}(q)}$. One can verify that Algorithm 1 always has rank$(q) \leq r + 1$. If rank$(q) = r + 1$, we create a new list $L^{r+1}$ which will be initialized to hold just solution $q$.

To implement Insert$(L, q, SP)$, we find the rank of $q$ and then insert into $L^{\text{rank}(q)}$. To find the rank of $q$, we use the following domination check (DC) primitive. Given $1 \leq j \leq r$ and $q$, $DC(j, q)$ is true if any solution in $L^j$ dominates $q$; otherwise $DC(j, q)$ is false. We then check $DC(j, q)$ starting with $j = 1$ and incrementing $j$ until $DC(j, q)$ becomes false. Then $rank(q)$ is this value of $j$.

## 6 ANALYSIS

We first show that the worst-case running time of MMS is no worse than brute-force search. We then show that MMS performs especially well on random input instances with many objectives; specifically, MMS-heap can achieve an expected running time that is linear in the input size for random input instances with $\Omega(\sqrt{n})$ objectives.

### 6.1 Worst-Case Running Time

Proposition 1. *The worst-case time complexity of MMS-sort and MMS-heap is $O(mn^2)$ time.*

We divide the proof of Proposition 1 into two steps that count the running time for each phase, ordering and ranking.

Lemma 6.1. *Ordering takes $O(mn + hn \log n)$ and $O(hn + n \log n)$ time, respectively, for MMS-sort and MMS-heap.*

Proof. For both algorithms, we start by sorting the first objective ($<_1$ or equivalently $<_\ell$) which takes $O(mn + n \log n)$ time [5]. For MMS-sort, we sort objective $i$ using $<_i$ for $2 \leq i \leq h$. The key observation is that for solutions $s_u$ and $s_v$ where $s_u^i = s_v^i$, we can order them by $<_\ell$ in $O(1)$ time because we have totally ordered $P$ by $<_\ell$. Thus, MMS-sort requires $O(hn \log n)$ time to sort the solutions for the remaining $h - 1$ objectives. Similarly, MMS-heap requires $O(hn)$ time to construct the remaining $h - 1$ heaps; we assume that MMS-heap uses the sorted list/array to store $Q_1$ since it has paid the full cost of sorting. The result follows by summing the cost of computing $Q_1$ and the remaining $Q_i$. □

Next, we compute the running time of the ranking phase which depends on the number of ranking rounds $k$ that are needed. In the worst case, $k = n$; in the best case, $k = n/h$.

Lemma 6.2. *The ranking phase with round $k$ runs in $O(mnk)$ time for both MMS-sort and MMS-heap.*

Proof. Both algorithms process a total of $hk$ solutions in the ranking phase. Extracting these $hk$ solutions takes $O(hk)$ time for MMS-sort and $O(hk \log n)$ time for MMS-heap. Only $n$ of these $hk$ solutions need to be ranked which requires $O(mnk)$ time for both algorithms since each solution must be checked against at most $k-1$ other solutions, and each check requires at most $m-1$ objective value comparisons. Finally, all $hk$ solutions need to be inserted into $L_i$ which takes a total of $O(hk)$ time for both algorithms. Because $mnk$ is strictly larger than $hk \log n$, both algorithms have a worst-case time complexity of $O(mnk)$ for the ranking phase. □

Proposition 1 follows from Lemmas 6.1 and 6.2, $h \leq m$ and $k \leq n$.

### 6.2 Average-Case Running Time

In the worst-case analysis, we made a few pessimistic assumptions. First, we assumed that sorting $P$ to compute $<_\ell$ requires $O(mn + n \log n)$ time. Second, we assumed that the number of rounds $k$ in the ranking phase would be $n$ rather than something smaller like $O(n/h)$. Third, we assumed that $|L_i| = k$ (note that $L_i$ are the solutions found from $Q_i$ which is an ordering data structure related to $i$-th objective) for each extracted solution $q_i$ that we needed to compare in the ranking phase. Finally, in the ranking phase, we assumed that checking whether a solution $s$ in $L_i$ dominates $q_i$ requires $O(m)$ time.

We now perform an average case analysis where the solutions in $P$ are "random". Specifically, we derive an upper bound on the expected number of comparisons performed by MMS where we make the following assumptions about the input instance. First, we assume that no two solutions have the same value in the same objective. Second, we assume that for each objective, all $n!$ possible permutations are equally likely. Third, we assume that the permutations for each objective are independent; that is, there is no correlation between the ranking in one objective to the ranking in a second objective.

The key analysis we will tighten is the number of comparisons required to rank all $n$ solutions in the ranking phase which we denote as $R(P)$. In Lemma 6.2, when ranking each of the $n$ solutions $q_i$, we assumed that $|L_i| = k - 1$ and that it would take $O(m)$ comparisons to determine if $s_u \in L_i$ dominates $q_i$ or not. We now do a more careful analysis with our assumptions.

Consider an arbitrary solution $p_j$ at the moment which is supposed to be ranked in the ranking phase. Without loss of generality, let $i$ be the $Q_i$ that $p_j$ was extracted from, and let $S_j = L_i$ denote the previously extracted solutions from $Q_i$ when $p_j$ is extracted. Finally, for any $s \in S_j$, let $Y(s, p_j)$ denote the number of comparisons required to determine if $s > p_j$ if we compare $s$ with $p_j$.

Then an upper bound on the total amount of work done for computing the rank of all $n$ solutions is

$$R(P) \leq \sum_{j=1}^{n} \Big( \sum_{s \in S_j} Y(s, p_j) \Big) \tag{1}$$

This is an upper bound because we assume that we check if all solutions $s \in S_j$ dominate $p_j$ whereas we skip some domination checks depending on the outcome of other checks.

We first derive an upper bound on $E[Y(s, p_j)]$.

Lemma 6.3. *For any solution $p_j$ and any $s \in S_j$ when $p_j$ is first extracted, $E[Y(s, p_j)] < 2$.*

Proof. Because of our assumptions that each objective is independent and all $n!$ permutations are equally likely, the probability that $p_j^l < s^l$ in any checked objective $l$ is $1/2$. Since we can stop once we find an objective $l$ where $p_j^l < s^l$, this is exactly the geometric distribution $NB(1, 1/2)$ which has expected value 2 except that we stop when we finish $m - 1$ trials; this early termination leads to a slightly smaller expected value. It follows that $E[Y(s, p_j)] < E[NB(1, 1/2)] = 2$. □

We next simplify $E[R(P)]$.

Lemma 6.4. $E[R(P)] \leq 2(\sum_{j=1}^{n} E[|S_j|])$.

Proof. We take the expectation of Equation 1 and then apply linearity of expectations to get $E[R(P)] \leq \sum_{j=1}^{n} E[\sum_{s \in S_j} Y(s, p_j)]$. We evaluate $E[\sum_{s \in S_j} Y(s, p_j)]$ by considering each possible size of $|S_j|$ using conditional probabilities. That is, $E[\sum_{s \in S_j} Y(s, p_j)] = E[\sum_{k=0}^{n-1}(\sum_{s \in S_j} Y(s, p_j) \mid k = |S_j|) \times Pr(k = |S_j|)]$. We can move the expected value inside the summation over $k$ to get $E[\sum_{s \in S_j} Y(s, p_j)] = \sum_{k=0}^{n-1}(E[(\sum_{s \in S_j} Y(s, p_j) \mid k = |S_j|)] \times Pr(k = |S_j|))$. Applying linearity of expectations, this simplifies to $E[\sum_{s \in S_j} Y(s, p_j)] = \sum_{k=0}^{n-1}(\sum_{s \in S_j} E[(Y(s, p_j) \mid k = |S_j|)] \times Pr(k = |S_j|))$. Applying Lemma 6.3, we get that $E[\sum_{s \in S_j} Y(s, p_j)] < \sum_{k=0}^{n-1}(2k \times Pr(k = |S_j|))$. Applying the definition of expected value, we get that $E[\sum_{s \in S_j} Y(s, p_j)] < 2E[S_j]$ and the result follows. □

We compute $\sum_{j=1}^{n} E[|S_j|]$ bounding $E[|S_j|]$ for $1 \leq j \leq n$.

Lemma 6.5. $E[|S_j|] \leq \frac{n}{h}$.

Proof. Consider an arbitrary solution $p_j$ and its associated value $E[|S_j|]$. We view the operation of MMS with respect to finding the first occurrence of $p_j$ as follows. In the first round, MMS performs $h$ trials where each trial has probability $1/n$ of being solution $p_j$ since all permutations of the $n$ solutions are equally likely in each objective. If $p_j$ appears during the first round, $|S_j| = 0$ since no solutions are chosen before $p_j$. In the $k$th round, if $p_j$ has not been found in any earlier round, MMS performs $h$ trials where each trial has probability $1/(n - k + 1)$ of being solution $p_j$ since all permutations of the $n$ solutions are equally likely in each objective and the assumption that $p_j$ has not been found in any earlier round means that there are only $n - k + 1$ solutions remaining to choose from in each objective. If $p_j$ appears during the $k$th round, $|S_j| = k - 1$ since $k - 1$ solutions are chosen before $p_j$ in the given objective.

In summary, MMS performs $nh$ trials where trial $t$ has probability $Pr(t = p_j \mid p_j$ has not been selected earlier$) = 1/(n - \lfloor (t - 1)/h \rfloor)$ of being $p_j$ assuming we have not seen $p_j$ in an earlier trial, and the resulting value of $|S_j|$ in the $t$th trial is $val(t) = \lfloor (t - 1)/h \rfloor$. We denote the probability that the $t$th trial is the first occurrence of $p_j$ as $Pr(t = fp_j)$ which is distinct from $Pr(t = p_j \mid p_j$ has not been selected earlier$)$. Then $E[|S_j|] = \sum_{t=1}^{nh} val(t) Pr(t = fp_j)$.

We simplify this summation by making two modifications which together cannot decrease the summation's value. The first change is to make $Pr(t = p_j \mid p_j$ has not been selected earlier$) = 1/n$ which is no larger than it was before. The second is to increase the number of trials from $nh$ to $\infty$. These two changes increase the probability that $p_j$ is selected in a later trial with a larger $val(t)$.

Let $pr'(t = fp_j)$ denote the probability that trial $t$ is the first occurrence of $p_j$ with these two changes, and let $Z' = \sum_{t=1}^{\infty} val(t)\, pr'(t = fp_j)$. We know $E[|S_j|] \leq Z'$. We note that $val(t) \leq t/h$, so $Z' \leq \sum_{t=1}^{\infty} t/h\, pr'(t = fp_j)$ which leads to $Z' \leq 1/h \sum_{t=1}^{\infty} t\, pr'(t = fp_j)$. We next observe that $\sum_{t=1}^{\infty} t\, pr'(t = fp_j)$ is exactly the expected value of the geometric distribution $NB(1, 1/n)$. This is because we are performing repeated trials with probability $1/n$ of success until we get our first success. The expected value of $NB(1, 1/n) = n$. Thus $Z' \leq n/h$, and the result follows. □

Corollary 1. $E[R(P)] \leq 2n^2/h$.

Proof. This follows directly from Lemmas 6.3, 6.4, and 6.5. □

We now bound the remaining comparisons performed where we separately analyze MMS-sort and MMS-heap. For both algorithms, computing $<_\ell$ requires only $O(n \log n)$ time because we assume that all solutions are distinct in each objective (same running time even when they are not distinct using three-way quicksort [39]). For MMS-sort, performing the initial sort for the other $h - 1$ objectives requires $O(nh \log n)$ time using a $\Theta(n \log n)$ sorting algorithm like mergesort. The extract minimum operations in the ranking phase are done in $O(nh)$ time. For MMS-heap, building the remaining $h - 1$ heaps requires only $O(nh)$ time since constructing binary heaps can be done in linear time. We now analyze the time required by MMS-heap for the extract minimum operations in the ranking phase.

Lemma 6.6. *The expected cost of the extract minimum operations performed by MMS-heap is $O(n \log^2 n)$.*

Proof. Without loss of generality, we assume that we extract the minimum from all $h$ heaps before checking to see if all solutions have been found. Let $U$ be the random variable which denotes this quantity; note that $U \leq n$.

We consider two cases: $h \leq 2 \ln n$ and $h > 2 \ln n$. If $h \leq 2 \ln n$, the worst case is we perform $hn$ extract minimum operations which gives us a total cost of $O(nh \log n)$ which is $O(n \log^2 n)$ since $h \leq 2 \ln n$ and the result follows.

We now consider the second case: $h > 2 \ln n$. What is the probability that $U$ exceeds $2n \ln(n/h)$ given $h > 2 \ln n$? We first compute the probability that a specific solution $p$ has not been sampled by this value of $U$. Solution $p$ is not sampled in one objective with probability $(n - 2n \ln(n/h)) = 1 - 2 \ln(n/h)$ since we assume the orderings are all equally likely. To not be sampled in all $h$ objectives occurs with probability $(1 - 2 \ln n/h)^h$ since we assume each dimension is independent. This probability is upper bounded by $e^{-2 \ln n} = 1/n^2$. This implies that the probability that any solution is not sampled in any of the $h$ objectives is at most $n/n^2 = 1/n$.

We can then upper bound the number of comparisons needed for extract minimum operations as follows. With probability at most $1/n$, $U$ is $n$ which means we have $hn$ extract minimum operations. With probability at least 1, $U$ is at most $2n \ln n/h$ which means we have at most $2n \ln n$ extract minimum operations. Thus, the expected cost of the extract minimum operations is at most $1/n \times hn \times \log n + 1 \times 2n \ln n \times \log n = h \log n + 2n \ln n \log n$, and the result follows. □

Finally, combining Corollary 1 and Lemma 6.6, we derive the following upper bound on the expected runtime of MMS-heap and MMS-sort. The main difference is that MMS-sort incurs a $\Theta(nh \log n)$ term for sorting in the ordering phase.

Theorem 6.7. *The expected runtime of MMS-heap is* $O\left(n^2/h + nh + n \log^2 n\right)$*, and the expected runtime of MMS-sort is* $O\left(n^2/h + nh \log n\right)$*.*

These results imply that MMS-heap achieves an expected running time that is linear in the input size for input instances with at least $\sqrt{n}$ objectives, and MMS-sort is slightly worse by a $\sqrt{\log n}$ factor.

Corollary 2. *For* $m = \Omega(\sqrt{n})$*, the expected runtime of MMS-heap is* $\Theta(n\sqrt{n}) = O(mn)$ *using* $h = \sqrt{n}$*, and the expected runtime of MMS-sort is* $\Theta(n\sqrt{n \log n})$ *using* $h = \sqrt{n/\log n}$*.*

## 7 CONCLUSION

Non-dominated sorting has emerged as a critical component for multi-objective optimization problems (MOPs). The best known running time for general $m$ is $O(mn^2)$, and has stood for almost two decades. In this work, we prove a matching conditional lower bounds based on Hitting Set Conjecture, and provide the first efficient algorithm on the inputs that are drawn from uniform distribution. An open problem is to either improve the average-case runtime or prove a conditional lower bound under inputs that are drawn from uniform distribution.

## REFERENCES

[1] 2014. A Provably Asymptotically Fast Version of the Generalized Jensen Algorithm for Non-dominated Sorting. In *Parallel Problem Solving from Nature - PPSN XIII*, Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipic, and Jim Smith (Eds.). Lecture Notes in Computer Science, Vol. 8672. Springer International Publishing.

[2] Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. 2016. Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*. Society for Industrial and Applied Mathematics, USA, 377–391.

[3] Jon Louis Bentley. 1980. Multidimensional Divide-and-conquer. *Commun. ACM* 23, 4 (April 1980), 214–229.

[4] Jon L. Bentley, Kenneth L. Clarkson, and David B. Levine. 1990. Fast Linear Expected-time Alogorithms for Computing Maxima and Convex Hulls. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 179–187.

[5] Jon L. Bentley and Robert Sedgewick. 1997. Fast Algorithms for Sorting and Searching Strings *(SODA '97)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 360–369.

[6] S. Borzsony, D. Kossmann, and K. Stocker. 2001. The Skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on.* 421–430.

[7] Adam L. Buchsbaum and Michael T. Goodrich. 2004. Three-Dimensional Layers of Maxima. *Algorithmica* 39, 4 (2004), 275–286.

[8] Maxim Buzdalov. 2019. Make Evolutionary Multiobjective Algorithms Scale Better with Advanced Data Structures: Van Emde Boas Tree for Non-dominated Sorting. In *EMO (Lecture Notes in Computer Science)*, Vol. 11411. Springer, 66–77.

[9] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. 2009. *The Complexity of Satisfiability of Small Depth Circuits.* Springer-Verlag, Berlin, Heidelberg, 75–85. https://doi.org/10.1007/978-3-642-11269-0_6

[10] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff. 2016. A Reference Vector Guided Evolutionary Algorithm for Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation* 20, 5 (Oct 2016), 773–791.

[11] David Corne, Joshua D. Knowles, and Martin J. Oates. 2000. The Pareto Envelope-Based Selection Algorithm for Multi-objective Optimisation. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN VI)*. Springer-Verlag, London, UK, UK, 839–848.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on* 6, 2 (Apr 2002), 182–197.

[13] M. Drozdík, Y. Akimoto, H. Aguirre, and K. Tanaka. 2015. Computational Cost Reduction of Nondominated Sorting Using the M-Front. *IEEE Transactions on Evolutionary Computation* 19, 5 (Oct 2015), 659–678.

[14] Hongbing Fang, Qian Wang, Yi-Cheng Tu, and Mark F. Horstemeyer. 2008. An Efficient Non-dominated Sorting Method for Evolutionary Algorithms. *Evol. Comput.* 16, 3 (Sept. 2008), 355–384.

[15] Félix-Antoine Fortin, Simon Grenier, and Marc Parizeau. 2013. Generalizing the Improved Run-time Complexity Algorithm for Non-dominated Sorting. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO '13)*. ACM, New York, NY, USA, 615–622.

[16] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. 2017. Completeness for First-Order Properties on Sparse Structures with Algorithmic Applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*. Society for Industrial and Applied Mathematics, USA, 2162–2181.

[17] Patrik Gustavsson and Anna Syberfeldt. 2017. A New Algorithm Using the Non-Dominated Tree to Improve Non-Dominated Sorting. *Evolutionary Computation* 26, 1 (2017), 89–116.

[18] Patrik Gustavsson and Anna Syberfeldt. 2018. A New Algorithm Using the Non-Dominated Tree to Improve Non-Dominated Sorting. *Evolutionary Computation* 26, 1 (2018).

[19] Julia Handl and Joshua Knowles. 2005. Exploiting the Trade-off — the Benefits of Multiple Objectives in Data Clustering. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO'05)*. Springer-Verlag, Berlin, Heidelberg, 547–560.

[20] Meng He, Cuong P. Nguyen, and Norbert Zeh. 2018. Maximal and Convex Layers of Random Point Sets. In *LATIN (Lecture Notes in Computer Science)*, Vol. 10807. Springer, 597–610.

[21] Alfred O. Hero and Gilles Fleury. 2004. Pareto-Optimal Methods for Gene Ranking. *Journal of VLSI signal processing systems for signal, image and video technology* 38, 3 (2004), 259–275.

[22] Russell Impagliazzo and Ramamohan Paturi. 2001. On the Complexity of k-SAT. *J. Comput. Syst. Sci.* 62, 2 (March 2001), 367–375. https://doi.org/10.1006/jcss.2000.1727

[23] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. 2001. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.* 63, 4 (Dec. 2001), 512–530. https://doi.org/10.1006/jcss.2001.1774

[24] A. Jaszkiewicz and T. Lust. 2018. ND-Tree-based update: a Fast Algorithm for the Dynamic Non-Dominance Problem. *IEEE Transactions on Evolutionary Computation* PP, 99 (2018), 1–1.

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

[26] Sanjiv Kapoor and Prakash V. Ramanan. 1989. Lower Bounds for Maximal and Convex Layers Problems. *Algorithmica* 4, 4 (1989), 447–459.

[27] J. Knowles and D. Corne. 1999. The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, Vol. 1. 105 Vol. 1.

[28] H. T. Kung, F. Luccio, and F. P. Preparata. 1975. On Finding the Maxima of a Set of Vectors. *J. ACM* 22, 4 (Oct. 1975), 469–476.

[29] Kevin Leyton-Brown and Yoav Shoham. 2008. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction* (1st ed.). Morgan and Claypool Publishers.

[30] K. Li, K. Deb, Q. Zhang, and S. Kwong. 2015. An Evolutionary Many-Objective Optimization Algorithm Based on Dominance and Decomposition. *IEEE Transactions on Evolutionary Computation* 19, 5 (Oct 2015), 694–716.

[31] K. Li, K. Deb, Q. Zhang, and Q. Zhang. 2017. Efficient Nondomination Level Update Method for Steady-State Evolutionary Multiobjective Optimization. *IEEE Transactions on Cybernetics* 47, 9 (Sept 2017), 2838–2849.

[32] Kent McClymont and Ed Keedwell. 2012. Deductive Sort and Climbing Sort: New Methods for Non-dominated Sorting. *Evol. Comput.* 20, 1 (March 2012), 1–26.

[33] Yakov Nekrich. 2011. A Fast Algorithm for Three-Dimensional Layers of Maxima Problem. In *WADS (Lecture Notes in Computer Science)*, Vol. 6844. Springer, 607–618.

[34] Franck Nielsen. 1996. Output-sensitive peeling of convex and maximal layers. *Inform. Process. Lett.* 59, 5 (1996), 255 – 259.

[35] Dhruv Rohatgi. 2019. Conditional Hardness of Earth Mover Distance. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019) (Leibniz International Proceedings in Informatics (LIPIcs))*, Dimitris Achlioptas and László A. Végh (Eds.), Vol. 145. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 12:1–12:17. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2019.12

[36] Dan Romik. 2015. *The Surprising Mathematics of Longest Increasing Subsequences* (1st ed.). Cambridge University Press.

[37] P.C. Roy, M.M. Islam, K. Murase, and Xin Yao. 2015. Evolutionary Path Control Strategy for Solving Many-Objective Optimization Problem. *Cybernetics, IEEE Transactions on* 45, 4 (April 2015), 702–715.

[38] Proteek Chandan Roy, Md. Monirul Islam, and Kalyanmoy Deb. 2016. Best Order Sort: A New Algorithm to Non-dominated Sorting for Evolutionary Multi-objective Optimization. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion (GECCO '16 Companion)*. ACM, 1113–1120.

[39] Robert Sedgewick and Kevin Wayne. 2011. *Algorithms* (4th ed.). Addison-Wesley Professional.

[40] N. Srinivas and Kalyanmoy Deb. 1994. Muiltiobjective Optimization Using Non-dominated Sorting in Genetic Algorithms. *Evol. Comput.* 2, 3 (Sept. 1994), 221–248.

[41] Suqin Tang, Zixing Cai, and Jinhua Zheng. 2008. A Fast Method of Constructing the Non-dominated Set: Arena's Principle. In *Proceedings of the 2008 Fourth International Conference on Natural Computation - Volume 01 (ICNC '08)*. IEEE Computer Society, 391–395.

[42] Handing Wang and Xin Yao. 2014. Corner Sort for Pareto-Based Many-Objective Optimization. *Cybernetics, IEEE Transactions on* 44, 1 (Jan 2014), 92–102.

[43] Ryan Williams. 2005. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* 348, 2 (2005), 357 – 365. https://doi.org/10.1016/j.tcs.2005.09.023 Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004).

[44] Virginia Vassilevska Williams. 2015. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk). In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Thore Husfeldt and Iyad Kanj (Eds.), Vol. 43. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 17–29. https://doi.org/10.4230/LIPIcs.IPEC.2015.17

[45] Virginia Vassilevska Williams. 2018. Some Open Problems in Fine-Grained Complexity. *SIGACT News* 49, 4 (Dec. 2018), 29–35. https://doi.org/10.1145/3300150.3300158

[46] X. Zhang, Y. Tian, R. Cheng, and Y. Jin. 2015. An Efficient Approach to Non-dominated Sorting for Evolutionary Multiobjective Optimization. *Evolutionary Computation, IEEE Transactions on* 19, 2 (April 2015), 201–213.

[47] X. Zhang, Y. Tian, R. Cheng, and Y. Jin. 2018. A Decision Variable Clustering-Based Evolutionary Algorithm for Large-Scale Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation* 22, 1 (Feb 2018), 97–112.

[48] X. Zhang, Y. Tian, and Y. Jin. 2015. A Knee Point-Driven Evolutionary Algorithm for Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation* 19, 6 (Dec 2015), 761–776.

[49] E. Zitzler, M. Laumanns, and L. Thiele. 2002. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, K.C. Giannakoglou et al. (Eds.). International Center for Numerical Methods in Engineering (CIMNE), 95–100.