Speeding-up of the Non-Dominated Sorting Genetic Algorithm-II by Selective De-Correlation

by

Abhinav Gorantla

A Thesis
Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved October 2025 by the
Graduate Supervisory Committee

K. Selçuk Candan, Chair
Huan Liu
Maria Luisa Sapino

ARIZONA STATE UNIVERSITY
December 2025

ABSTRACT

Non-Dominated Sorting (NDS) algorithms are used widely in many fields and for a wide-range of applications. Primarily, NDS is used to rank tuples in a dataset, but this ranking is sometimes the intermediate results that is used for other downstream tasks. In this thesis, I aim to optimize NDS algorithms in an attempt to optimize pareto-dominance based genetic algorithms, specifically NSGA-II. I achieve this by studying correlations between objectives in a Multi-Objective Optimization (MOO) Problem and leverage this information to optimize NDS. We propose and evaluate two optimization methods leveraging the information of correlations between objective functions in an MOO problem. This thesis first presents a divide and conquer approach, D-NDS(De-correlate and Merge NDS) and then a second approach, SD-NDS (Selective De-Correlate NDS) which orders objectives in the objective tuple in the most optimal order to reduce the number of unnecessary dominance checks performed. Additionally, this thesis also explores optimizing each of the dominance check itself, this is something, to the best of my knowledge, that no one in the past has tried to optimize. This research also provides an algorithmic analysis which explains how and why SD-NDS works. The experiments on synthetic data and MOO problems presented in this thesis shows a clear reduction in time taken by NSGA-II when we use SD-NDS in conjunction with baseline NSGA-II compared to when we run vanilla NSGA-II.

DEDICATION

To my family, mentors, friends and everyone who believed in me and supported me along the way.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. K. Selçuk Candan, for his unwavering support throughout my master's program. Thank you for believing in me when I did not believe in myself. Your mentorship during my masters degree challenged me every day, helped me discover my potential, and gave me opportunities to grow as a researcher and taught me lessons that will guide my career.

I am grateful to my committee members, Dr. Huan Liu and Dr. Maria Luisa Sapino, for their guidance on my thesis and for mentoring me on other projects, including CausalBench and Causal Skylines.

This work was supported by the National Science Foundation (NSF) under Grant No. 2311716, Elements: CausalBench—A Cyberinfrastructure for Causal-Learning Benchmarking for Efficacy, Reproducibility, and Scientific Collaboration. Computing resources for my experiments were provided by the NSF-supported platform, Chameleon Cloud (chameleoncloud.org).

To my lab mates and colleagues—Pratanu Mandal, Ahmet Kapkic, Shubhodeep Mitra, Ertugrul Coban, Tasneema Azad, Shu Wan, Dr. Paras Sheth, Dr. Yoonhyuk Choi, Dr. Geunsoo Jang—thank you for the discussions and for helping me learn from your experiences. Your support made this work possible. I especially thank Pratanu Mandal for our collaboration on Causal Skylines. Thank you, Shubhodeep, for showing me bike paths around Tempe; they helped me switch off when work felt all-consuming.

To my family and my extended family—especially my mother and my father—thank you for your support throughout my life and for teaching me invaluable lessons.

To the friends I met at ASU and in Tempe - cycling mates, course project partners, classmates, housemates, especially Abhiram, Niranjan and Vishrut - your steady support, direct and indirect, carried me through. And finally, to my friend Abhiram, thank you for encouraging me to take chances, both in life and in my career and for everything from gym

sessions to research discussions. Your dedication to research and your work ethic have inspired me these past two years.

TABLE OF CONTENTS

LIST OF TABLES

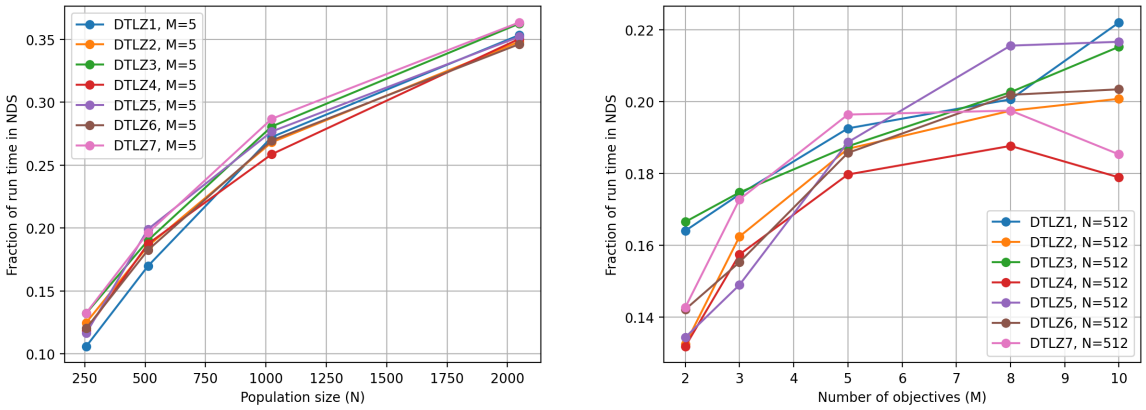
LIST OF FIGURES

Chapter 1

MOTIVATION

Multi-Objective Evolutionary Algorithms (MOEAs) are a common way to solve multi-objective optimization problems. MOEAs use concepts from Darwin's theory of evolution such as the concept of the "survival of the fittest" to solve optimization and search problems. A well-known class of MOEAs is Genetic Algorithms (GAs). Genetic algorithms use common evolutionary operations such as crossover and mutation to solve Multi-Objective Optimization (MOO) Problems and to perform search in large parameter spaces. Traditionally, Multi-Objective Optimization algorithms convert the multiple objectives in the problem (objective vector) into a single scalar value and then, they use this scalar value to determine the fitness ranking of the candidates in the population (Deb *et al.*, 2016). This process of converting an objective vector to a single scalar value involves weighting the individual objectives and most often this weighting process is subjective and in some applications, the weights assigned to objectives may vary depending on the use-case. In order to eliminate this subjective step of assigning weights to each objective in an MOO problem, a sub-class of genetic algorithms has evolved which uses the entire raw objective vector to determine the fitness ranking of the candidates in the population (Srinivas and Deb, 1994; Deb *et al.*, 2002; Deb and Jain, 2014; Horn *et al.*, 1994; Zitzler and Thiele, 1998; Zitzler *et al.*, 2001), this is achieved using the concept of pareto-dominance.

More specifically, some genetic algorithms (GAs) like NSGA-I, II, III(Srinivas and Deb, 1994; Deb *et al.*, 2002; Deb and Jain, 2014) use the Non-Dominated Sorting (NDS) operator to achieve this pareto-dominance based fitness ranking. Since the introduction of these pareto-dominance based GAs, one major direction of research has been to algorithmically improve the run-time of this NDS operator. More specifically, one of the major improve-



(a) Varying Population Size

(b) Varying Number of Objectives

Figure 1.1: Fraction of time spent performing Non-Dominated Sorting in NSGA-II.

ments from NSGA(Srinivas and Deb, 1994) to NSGA-II(Deb *et al.*, 2002) was the reduction in the worst-case time complexity of the NDS operator used from $O(MN^3)$ to $O(MN^2)$. Following this research direction, I analyzed how much time, in a single NSGA-II run, is spent performing NDS. For this analysis I used ENS-SS(Zhang *et al.*, 2015) as the NDS algorithm and the default settings for crossover and mutation for NSGA-II as presented in (Blank and Deb, 2020). For the test problems, I used the DTLZ test suite (Deb *et al.*, 2005).

The results of my analysis presented in Figure 1.1 show that as 10-15% of the run-time is spent performing NDS in a single NSGA-II run, and this number almost triples to 35% of the total run-time when I increase the population size by ten times. In Figure 1.1b, I see that as I increase the number of objectives in each test problem, there is an increase in the fraction of time spent performing NDS.

Following the observations above, it is obvious that reducing the time complexity of NDS operations is going to noticeably reduce the time taken to solve an MOO problem using NSGA-II. In addition to pareto-dominance based GAs like NSGA(Srinivas and Deb, 1994; Deb *et al.*, 2002; Deb and Jain, 2014), SPEA(Zitzler and Thiele, 1998; Zitzler *et al.*, 2001) and NPGA(Horn *et al.*, 1994), Non-Dominated Sorting (NDS) is used in many fields, including machine learning (Hashemi *et al.*, 2023), information retrieval (Burlacu, 2022; Vrajitoru,

2000), diversity-focussed Bayesian Optimization methods (Ahmadianshalchi *et al.*, 2024) and also in applications like spacecraft trajectory design, cantilever plate design (Deb, 2005), and more engineering optimization problems (Deb and Jain, 2003). Most recent work in optimization of Non-Dominated sorting has been in reducing the number of dominance checks. These efforts have resulted in optimized NDS algorithms like the ENS family of algorithms (Zhang *et al.*, 2015, 2018) and DCNS family of algorithms (Mishra *et al.*, 2019). The common feature of both of these methods is that they leverage pre-sorting to reduce the number of redundant dominance checks during the NDS operation. However, to the best of my knowledge, there is no work done that studies how the complexity of the MOO problem itself relates to the time taken to perform the NDS operation. One way to measure the complexity of an MOO is to look at the correlation between its objectives, if the correlation between two objectives is negative but both of them are to be minimized, then it is a harder problem to solve than if those two objectives were positively correlated.

In this thesis the objective is to look at the correlations of between objectives of a Multi-Objective Optimization (MOO) problem and leverage this information to optimize algorithmic time-complexity of applying the NDS operation on this data.

Chapter 2

INTRODUCTION

Pareto-Dominance based genetic algorithms like NSGA-II (Deb *et al.*, 2002) utilize Non-Dominated Sorting (NDS) in their selection step. Algorithms like NDS which use Pareto-Dominance can be optimal only when the optimization parameters are either highly negatively correlated or highly positively correlated. This requirement depends on the optimization criteria for the objective variates. Negative correlation between the optimization parameters is beneficial when the optimization objectives of these parameters is opposite (one parameter requires maximization and other requires minimization), similarly, positive correlation is beneficial when the optimization objectives of the parameters align with each other.

This thesis aims to extend the selective de-correlation method proposed in our work, Causal Search for Skylines (CSS), to Non-Dominated Sorting (NDS) in an attempt to make the NDS algorithm more efficient.

I study correlation between objectives in MOO problems and study two different selective de-correlation methods. The first method proposed in this thesis, D-NDS, divides the population by de-correlating in the objective space. The second method, SD-NDS, introduces three heuristics that use correlations between objectives to suggest the optimal variate ordering to reduce the time consumption for the NDS step in a GA run.

In this thesis, we observe that D-NDS has some inefficiencies in its algorithm and there are already some divide and conquer based NDS algorithms like the DCNS class of algorithms proposed by (Mishra *et al.*, 2016) that achieve NDS through the divide and conquer approach much more efficiently, however, they do not follow any heuristic to optimally divide the data.

The second proposed method showed promising results when tested on synthetic data. SD-NDS is evaluated on different objectives that are related to one another in different ways and showed that our proposed minCorr heuristic properly identifies suitable variate orderings accurately. I then integrate our proposed SD-NDS method into the NSGA-II workflow and see the optimization in run-time also in this case.

Research Objectives

My research objective is to optimize the MOO algorithm, NSGA-II, by optimizing Non-Dominated Sorting (NDS) which is one of its core components. This is done by applying our knowledge of the impact of correlations on pareto-optimization problems and selectively de-correlating the data.

I also want to thoroughly investigate the performance of our proposed optimization methodology on objectives that are related to each other in different ways (competing, non-competing and un-related).

Thesis Structure

The rest of this thesis is organized as follows:

- Chapter 3 Background and Related Work - This section goes over some core concepts that are crucial to get a good understanding of the research presented in this thesis and discuss some recent advancements in the fields of MOO, NDS and Genetic Algorithms and present how the research presented in this thesis adds-on and closes gaps in the work that is already existing in the community.

- Chapter 4 Methodology - The two methods to optimize NDS proposed in this thesis are presented here. I also go over why our first approach, D-NDS, failed and how our second approach, SD-NDS, works.

- Chapter 5 Experimental Evaluation - This chapter includes a thorough evaluation of our proposed method to optimize NDS algorithms.
- Chapter 6 Conclusion - Includes a summary of the research presented in this thesis and highlights the results obtained in our experiments and presents possible future research directions.

Chapter 3

BACKGROUND AND RELATED WORK

Multi Objective Optimization

Multi-Objective Optimization (MOO) is a well-studied problem in many scientific fields including computer science, mechanical engineering (Carroll and Johnson, 1984; Kohira *et al.*, 2018) and chemical engineering (Rangaiah, 2016). Most multi-objective optimization problems have more than one solution, unlike a single objective optimization problem which almost always has a unique (optimal) solution. This set of solutions (optimized objective function values) for a Multi-Objective optimization problem is called a "*Pareto Front*". The set of decision variables corresponding to this "*Pareto Front*" is called the "*Pareto Set*" (Ahmadianshalchi *et al.*, 2024). There are many ways to solve an MOO problem. One of the easiest ways is to scalarize the multiple objectives by assigning weights to each of the individual objectives in the MOO problem and then sort the data using this single scalar objective (Deb *et al.*, 2016). While this method provides a straight-forward and simple way to convert a MOO problem to a single objective problem, the issue lies in finding out the weighting of the multiple objectives. These weights are often subjective and vary from individual to individual. Imprecise weighting of these objectives can also cause some favorable solutions to not be returned to the end-user (false negatives). Pareto-dominance based Multi-Objective Optimization methods like NSGA(Srinivas and Deb, 1994; Deb *et al.*, 2002; Deb and Jain, 2014), NPGA(Horn *et al.*, 1994), SPEA(Zitzler and Thiele, 1998; Zitzler *et al.*, 2001) do not depend on weighting to solve the optimization problem. They instead take in a set of objective functions, one for each of the objectives in the MOO problem. Such algorithms give us a *set* of solutions which are pareto-optimal for our MOO

problem. All the solutions in this "pareto-optimal" set are as good as one another. This is better than the initially discussed weighting method in the sense that these algorithms return all the possible optimal solutions for the given MOO problem. The end-user then is given the option to choose whatever subset of solutions they want to use out of this pareto-optimal set. Such a method leaves out the subjectivity in defining weights for each objective and completely negates the possibility of having false negatives.

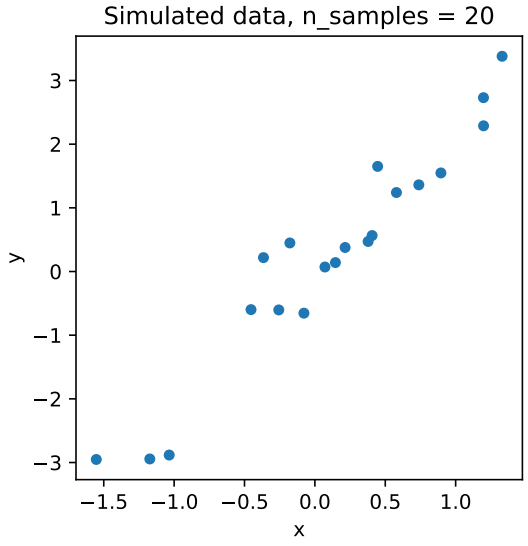
Genetic algorithms are an important subset of such pareto-dominance based methods and are being widely used to solve Multi-Objective Optimization (MOO) Problems. Some recent works in this area-(Tanabe and Ishibuchi, 2020; Kohira *et al.*, 2018; Altinoz, 2022) show that genetic algorithms like NSGA-II (Deb *et al.*, 2002), SPEA (Zitzler and Thiele, 1998; Zitzler *et al.*, 2001), NPGA (Horn *et al.*, 1994) are being widely used to solve multi-objective engineering problems. The train gear design problem proposed by (Carroll and Johnson, 1984) is a classic engineering problem that was solved using genetic algorithms in (Deb and Jain, 2003). Several other real-world multi-objective optimization (MOO) problems have been summarized in (Altinoz, 2022; Tanabe and Ishibuchi, 2020; Kohira *et al.*, 2018).

One disadvantage of using the pareto-dominance based methods is the fact that the non-dominated sorting part of the algorithm is time and memory intensive and can be significantly slower than other methods.

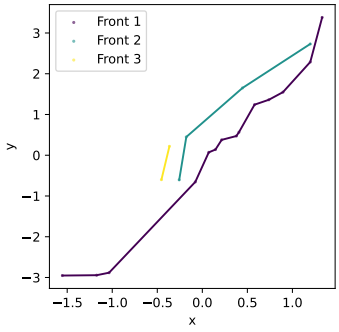
Non-Dominated Sorting

The goal of Non-Dominated Sorting is to assign partial order to all the points in a dataset based on more than one objective functions (Deb, 2005). The points in the dataset have a partial order in the sense that all points in the p^{th} pareto-optimal frontier cannot be dominated by any points in every q^{th} pareto-optimal frontier where, $p < q$.

The output of NDS operation is illustrated in Figure 3.1.



(a) Data Plot



(b) NDS Output

Figure 3.1: Non-Dominated Sorting (NDS) output

A class of Genetic Algorithms (GAs) which solve MOO problems use the concept of pareto-dominance to ascertain the fitness ranking of the population. This ranking is performed using Non-Dominated Sorting (NDS). Non-Dominated Sorting involves arranging all the candidates in an order such that given any two candidates, C_m and C_n, with ranks m, n, such that, $m < n$, C_n does not dominate C_m. In other words, C_m is as good or better than C_n in all the objectives. Non-Dominated sorting is known to be an expensive algorithm both in terms of memory and time consumption.

The Non-Dominated Sorting Genetic Algorithm (NSGA) is one of the first algorithms to have used Non-Dominated sorting process in its selection step. The implementation of NDS in NSGA had a time complexity of $O(MN^3)$. This is the most naive method to perform Non-Dominated Sorting. The successor to NSGA proposed in (Deb *et al.*, 2002) had a more time optimal NDS algorithm called FastNDS and had a time complexity of $O(MN^2)$ and a space complexity of $O(N^2)$, where N is the population size and M is the number of objectives. The main drawback of using FastNDS is that the memory usage of

this algorithm increases exponentially as we increase the population size. FastNDS also performs all the N^2 possible dominance checks before the sorting process even starts. This causes it to have a best case *and* worst case time complexity of $O(MN^2)$. Work done by (Vrajitoru, 2000) proves both theoretically and empirically that larger population size is more important than larger number of generations when solving an MOO with genetic algorithms, this makes it even more important for the space complexity to be independent of the population size. Following FastNDS, several other Non-Dominated Sorting methods have been proposed. Some notable ones include Jensen's method (Jensen, 2003), which is the first divide and conquer approach to Non-Dominated Sorting, Efficient Non-Dominated Sorting(ENS) (Zhang *et al.*, 2015), Tree based methods - T-ENS(Zhang *et al.*, 2018), ENS-NDT(Gustavsson and Syberfeldt, 2018), dominance degree based methods - DDA-NS(Zhou *et al.*, 2016), DDA-ENS (Mishra and Senwar, 2020) and the DCNS (Mishra *et al.*, 2019) class of algorithms, which also use the divide and conquer approach.

Most recent work in optimizing Non-Dominated Sorting (NDS), like the ENS(Zhang *et al.*, 2015), DDA(Zhou *et al.*, 2016; Mishra and Senwar, 2020) and DCNS(Mishra *et al.*, 2019) methods have focused on reducing unnecessary dominance checks. A significant breakthrough in the reduction of space complexity in NDS occurred with the introduction of Efficient Non-Dominated Sorting (ENS) (Zhang *et al.*, 2015). The ENS family of NDS algorithms (Zhang *et al.*, 2015; Mishra and Coello, 2018; Zhang *et al.*, 2018) reduce the number of redundant dominance comparisons by pre-sorting the population before non-dominated sorting is applied. The first ENS paper (Zhang *et al.*, 2015) proposes two methods, ENS-SS and ENS-BS. These two methods are distinguished by the way in which the search for a Pareto front to which a candidate belongs is done. In ENS-SS, this search is performed in a sequential manner leading to a time complexity of $O(MN\sqrt{N})$ and space complexity of $O(1)$,whereas, in ENS-BS, this search is done using Binary Search leading to a lower best-case time complexity of $O(MNlog(N))$ and space complexity of $O(1)$.

Other methods like T-ENS(Zhang *et al.*, 2018) have used different data structures like trees to further optimize the time complexity of NDS to $O(MNlogN/logM)$. However, their study (Zhang *et al.*, 2018) does not include a study of how this optimized time complexity translates to reduction in run-times as compared to NSGA-II runs using ENS(Zhang *et al.*, 2015). The authors of ENS-NDT introduce a new data structure called the NDTree which further improves the worst-case time complexity of the ENS algorithm when $M > logN$.

The dominance degree method presented in (Zhou *et al.*, 2016) optimizes NDS by using a dominance degree matrix. This method is further optimized by (Mishra and Senwar, 2020) where the authors show that the DDA-NS method proposed in (Zhou *et al.*, 2016) has a worst case time complexity of $O(MN^2 + N^3)$ which makes this algorithm exponentially worse as we keep increasing the number of samples, N, in our population. The optimized dominance degree method, DDA-ENS, presented in (Mishra and Senwar, 2020) claims a worst case time complexity of $O(MN^2)$ and achieves this by taking in ideas from ENS (Zhang *et al.*, 2015) and integrating it with the dominance degree approach presented in (Zhou *et al.*, 2016).

And finally, the divide and conquer based methods proposed in (Mishra *et al.*, 2019) also uses the pre-sorting approach. The DCNS family of algorithms proposed in (Mishra *et al.*, 2019) initially perform a lexical sort of the entire population and recursively divide the entire population until we obtain N singleton sets of Pareto fronts. Once these N sets of Pareto fronts are obtained, they propose a merge algorithm which iteratively merges two consecutive sets of Pareto fronts together. This is done in such a way that the dominance order in each of the sets of Pareto fronts is preserved at each of the merge steps. The core idea behind the DCNS class of algorithms is to reduce the number of dominance checks by ordering the candidates in our population in smaller sets rather than ordering the candidates by considering the *entire* population. Using these optimization methods, the DCNS class of algorithms achieve a best case time complexity of $O(MNlogN)$ and a worst case time

complexity of $O(Nlog^{M-1}N)$.

Most of the existing research on optimizing Non-Dominated Sorting algorithms has been on theoretically reducing the time and memory complexity. Although these methods have achieved significant success, there is an important direction that they have not considered: How do the characteristics of objective functions relate to the efficiency of the Non-Dominated Sorting algorithms? The research presented in this thesis dives deep in this direction and proposes methods that leverage the relationship between objective functions in an MOO problem to optimize existing NDS algorithms.

Evolutionary Algorithms

Evolutionary Algorithms are inspired by Darwin's theory of evolution. They take inspiration from how humans and other living things on the surface of planet earth evolve (Deb, 2011). These kinds of algorithms make use of evolution concepts like mutation, crossover and selection to perform tasks like numerical optimization and searching parameter spaces.

This thesis is going to focus on a subset of Evolutionary Algorithms called Genetic algorithms.

Genetic Algorithms are a class of Evolutionary algorithms and use evolutionary concepts such as mutation and crossover to explore the search space of decision variables in a multi-objective optimization problem. In the context of genetic algorithms, we use the term "fitness function" when referring to the term "objective function".

Some steps common to most genetic algorithms are:

1. Sample an initial population.
2. Evaluate fitness functions.
3. Select parent population.
4. Perform crossover on the selected population.

5. Perform mutation.

6. If convergence condition met, then, terminate, else, go to Step 2.

Chapter 4

METHODOLOGY

This thesis presents two ways to leverage correlations between objectives in an MOO problem to optimize solving them when using a pareto-dominance based genetic algorithms like NSGA-II (Deb *et al.*, 2002). First, I go over how I applied the divide and conquer approach we proposed in our work, Causal Search for Skylines (CSS), in an attempt to optimize Non-Dominated Sorting and then introduce the second approach, SD-NDS, which proposes heuristics to find most optimal variate ordering to optimize the process of Non-Dominated Sorting in NSGA-II. Before proceeding to describe the details of my proposed methods, let us go over the intuitions behind why the proposed methods work:

1. **De-correlate and Merge NDS (D-NDS)** - The idea behind de-correlate and merge NDS was to leverage divide and conquer style optimization while also making use of the insights gained from studying correlations between objectives in MOO problems. An example of how correlations affect MOO problems is shown in Figure 4.2. The basic idea here is that, as the objectives become more competing (correlations between them become negative), the difficulty of the MOO problem increases and therefore it takes longer time to solve it. In the worst case, when all the objectives are perfectly negatively correlated with one another, this will result in NDS generating a partial ordering of candidates where every candidate is assigned rank 1.

   Since the correlations between objectives is a characteristic of the problem itself, there is no way for us to alter this. However dividing the population of data in the MOO problem into multiple subsets and ensuring that these subsets have objectives that are more positively correlated than the global population is possible.

In D-NDS, this process of dividing the entire population into subsets is done by applying K-Means clustering in the objective space on the population in the MOO problem. This effectively de-correlates the objectives in each of the resulting clusters. While this method showed improvements in 2 objective scenarios, this improvements did not generalize over varying number of objectives and sample sizes as we will see in Table 5.2.

2. **Selective De-correlation NDS (SD-NDS)** - SD-NDS selectively de-correlates the objectives in the MOO problem by variate ordering. This method, discussed in detail in Section 4, can optimize NDS in two ways. (1) by reducing the cost of each dominance check itself and (2) by pre-sorting the population in a way that is closer to the end result. The second optimization opportunity is crucial for pre-sorting based methods (Zhang *et al.*, 2015, 2018) and divide and conquer methods (Mishra *et al.*, 2019).

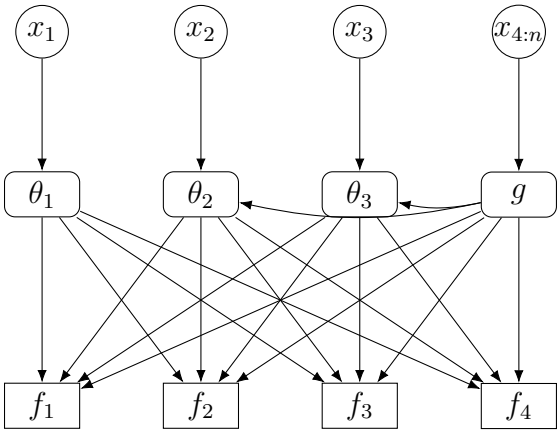
   In the Divide and Conquer based NDS algorithm (Mishra *et al.*, 2019), pre-sorting also tries to minimize the local negative correlations between objectives. This leads to lower inter-partition dominance checks and hence lower run-time.

In the rest of this thesis, "multi-objective optimization (MOO) problem" refers to an MOO problem where the objective is to minimize all objective functions (Min-Min) unless otherwise stated.

However, the readers should note that all other kinds of optimization problems – Max-Max, Max-Min – can be reduced to a Min-Min style optimization problem. This can be done by inverting the values of the objective to be maximized. One possible way to perform this inversion is to multiply the objective values by -1.

De-Correlate and Merge NDS generalizes the selective de-correlation based skyline search methodology we proposed in Causal Search for Skylines (CSS). In Causal Search for Skylines(CSS), we utilized the causal graph underlying the data to search for the optimal conditioning set which performed d-separation in such a way that correlations favorable to the Skyline search were preserved (or even improved) and the correlations slowing down the Skyline search were reduced.

However, when it comes to most MOO problems, the relationship between objectives and decision variables is non-linear (Deb *et al.*, 2005; Tanabe and Ishibuchi, 2020). An example to support this claim is given in Figure 4.1. This introduces one main challenge: we cannot come up with a general data partitioning strategy for the entire problem, since the correlations between variates change depending on the range of the input values. From the work done in CSS, we know that no correlation is favorable than having correlation that negatively impacts the performance of the Skyline algorithm.



(a) DTLZ5 causal graph.

$$
\begin{aligned}
&\text{Min. } f_1(\mathbf{x}) = (1 + g(\mathbf{x}_M)) \cos(\theta_1 \pi/2) \cdots \cos(\theta_{M-2}\pi/2) \cos(\theta_{M-1}\pi/2),\\
&\text{Min. } f_2(\mathbf{x}) = (1 + g(\mathbf{x}_M)) \cos(\theta_1 \pi/2) \cdots \cos(\theta_{M-2}\pi/2) \sin(\theta_{M-1}\pi/2),\\
&\text{Min. } f_3(\mathbf{x}) = (1 + g(\mathbf{x}_M)) \cos(\theta_1 \pi/2) \cdots \sin(\theta_{M-2}\pi/2),\\
&\vdots \quad \vdots\\
&\text{Min. } f_M(\mathbf{x}) = (1 + g(\mathbf{x}_M)) \sin(\theta_1 \pi/2),\\
&\text{with } \theta_i = \tfrac{\pi}{4(1+g(\mathbf{x}_M))} (1 + 2g(\mathbf{x}_M) x_i), \quad \text{for } i = 2, 3, \ldots, (M-1),\\
&\qquad g(\mathbf{x}_M) = \textstyle\sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2,\\
&\qquad 0 \le x_i \le 1, \quad \text{for } i = 1, 2, \ldots, n.
\end{aligned}
$$

(b) DTLZ5 Objective Function definition (Deb *et al.*, 2005)

Figure 4.1: DTLZ5 dependencies for $n_{\text{obj}} = 4$. Here, $\boldsymbol{x}$ is the decision vector and $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \ldots, f_4(\boldsymbol{x})]$ is the objective vector.

Therefore, D-NDS completely de-correlates the objectives in an MOO problem. This

de-correlation is performed by considering the objective space and applying a clustering algorithm such as K-Means in this objective space. Once the subsets of data are obtained, the objectives are no longer negatively correlated in these subsets. Now, NDS is applied on each of these subsets. The resulting NDS orders from each subset of data are then merged using the proposed merge algorithm in Algorithm 2. Any NDS algorithm can be plugged into D-NDS. D-NDS algorithm is presented in Algorithm 1 and the proposed merge algorithm is presented in Algorithm 2.

Algorithm 1: D-NDS, with ENS based implementation

$\mathcal{P} \leftarrow$ Population to be sorted
$m \leftarrow$ number of clusters to be generated
Result: FinalNDS

$\mathcal{C} \leftarrow$ KMeans$(\mathcal{P}, m)$
$\mathcal{NDS} \leftarrow \{\text{ENS}(C_i) \mid C_i \in \mathcal{C}\}$
FinalNDS $\leftarrow$ `NDSMerge`$(\mathcal{NDS}, m)$; `// Algorithm 2`
Return FinalNDS

Algorithm 2: NDSMerge

Data: $\mathcal{NDS}$: queue of NDS orders per cluster
Result: NDSOrder1
NDSOrder1 $\leftarrow$ None
NDSOrder2 $\leftarrow$ None
while len$(\mathcal{NDS}) > 0$ **do**
  if $\neg$ *NDSOrder1* **then**
    NDSOrder1 $\leftarrow \mathcal{NDS}$.popleft()
  end
  if $\neg$ *NDSOrder2* **then**
    NDSOrder2 $\leftarrow \mathcal{NDS}$.popleft()
  end
  `AllPF` $\leftarrow$ `List()`
  while len(*NDSOrder1*) $> 0 \ \vee$ len(*NDSOrder2*) > 0 **do**
    $PF_1 \leftarrow$ NDSOrder1.popleft()
    $PF_2 \leftarrow$ NDSOrder2.popleft()
    `CombinedData` $\leftarrow PF_1 \cup PF_2 \cup$ `Discarded`
    `sort(CombinedData, order=ASCENDING, key` $= arr \mapsto \sum arr$`)`
    `MergedPF` $\leftarrow$ `List()`
    `CurrBbox` $\leftarrow$ `None`
    foreach $c_1 \in$ *CombinedData* **do**
      if `IsDominating(`c_1, *CurrBbox*`)` **then**
        `Discarded.append(`c_1`)`
      end
      else
        if $\exists\, c_2 \in$ *MergedPF s.t.* `IsDominating(`c_1, c_2`)` **then**
          `Discarded.append(`c_1`)`
          `Break;`
        end
        else
          `MergedPF.append(`c_1`)`
        end
      end
    end
    `AllPF.append(MergedPF)`
  end
  NDSOrder1 $\leftarrow$ `AllPF`
end
return *NDSOrder1*

As we will see in my experimental evaluation of D-NDS in Chapter 5, D-NDS is inefficient, not only does it not improve the run-time of NDS algorithm, it actually increases the run-time. The following section presents a selective de-correlation based method, SD-NDS, which shows promising results in terms of making NDS efficient through run-time optimization.

Selective De-Correlation Based NDS (SD-NDS)

There are two worst cases possible in Non-Dominated Sorting (NDS), depending on the characteristics of the NDS algorithm I use. The first worst case occurs when there are many small Pareto fronts in the set of Pareto fronts and the second worst case occurs when there are few Pareto fronts and therefore, there are more candidates in each of the resulting Pareto Fronts. The occurrence of these kinds of Pareto front sets depends on the correlation of the objectives. As we see in Figure 4.2, when we consider minimizing on objectives represented on X and Y axes, we see that as the correlation between X and Y becomes more positive, the number of Pareto fronts in the data distribution increases and consequently the number of candidates in each Pareto front decreases. Alternatively, when the correlation between X and Y becomes more negative, the number of Pareto fronts decreases thereby increasing the number of candidates in each Pareto front. It is clear that for algorithms dependent on pre-sorting, like the ENS family of algorithms (Zhang *et al.*, 2015), specifically ENS-SS and ENS-BS, as the number of Pareto fronts increase, the Non-Dominated Sorting process becomes less costly. This is due to the algorithms seeking a candidate that dominates the current placement candidate. And the time complexity of this is directly proportional to the size of the Pareto front, this in turn is inversely proportional to the number of Pareto fronts. In other words, we can say with certainty that when using ENS-SS and ENS-BS algorithms for performing NDS, our best case would be having all objectives to be non-conflicting with one another. However, since the relationship between objectives is a characteristic of

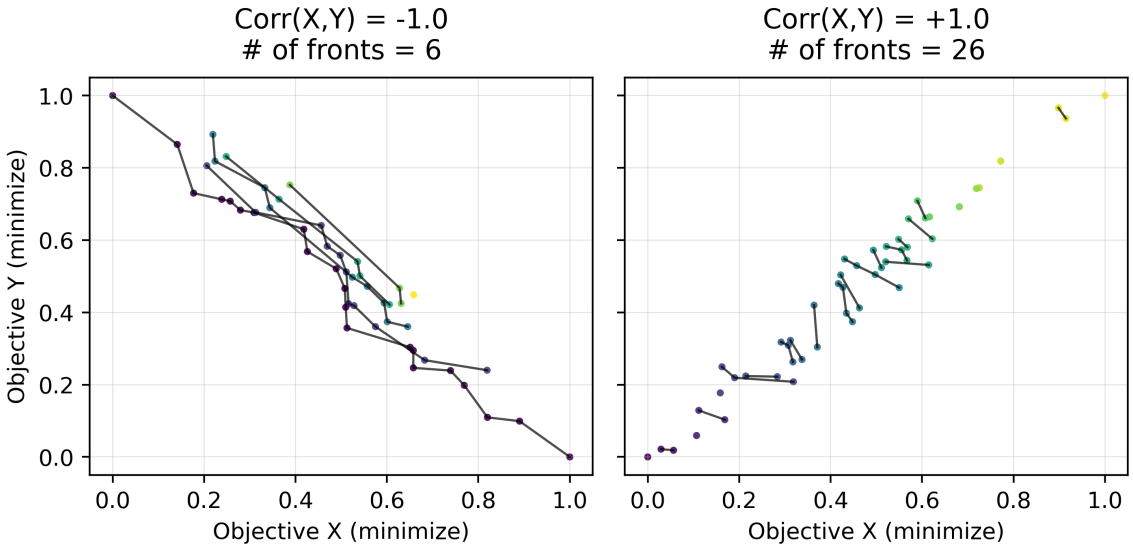


Figure 4.2: Effect of correlation between objectives on the number of resuting Pareto fronts in NDS

the optimization problem itself, it is not possible for us to change how the objectives are distributed.

Alternatively, we can leverage the correlations between the objectives and optimally solve the conflicts between them. A well-known approach to solving such constraint problems (Constraint Satisfaction Problems (CSPs)) is to first solve the most-constrained objective first (Norvig and Intelligence, 2002). The same logic can be used in Non-Dominated Sorting. In the case of Non-Dominated Sorting, the ”most-constrained objective” will be the one that is most negatively correlated (most competing) with the other objectives and the least constrained objective will be the most positively correlated objective. Putting these observations into the context of NDS, we can conclude that, when we order the objectives in the objective vector starting with the most negatively correlated objective and ending with the least negatively correlated objective, when we pre-sort the data, it is first ordered by the most negatively correlated objective. This means that the most conflicting objective is solved.

Using these conclusions as the basis, three heuristics are proposed to guide variate ordering:

1. Minimum Correlation First (*MinCorr*) - From the discussion above, it is clear to us that in most pre-sorting based NDS algorithms, we would want to have most negatively correlated (most conflicting) objectives first. To do this, we obtain a permutation $\Pi = [v_1, v_2, v_3, \ldots, v_M]$ such that:

$$\forall_{v_i, v_j} \ (i < j) \ \rightarrow \ \left(\sum_{h \in [v_1, v_2, \ldots, v_M]; h \neq v_i} corr(h, v_i) < \sum_{h \in [v_1, v_2, \ldots, v_M]; h \neq v_j} corr(h, v_j) \right).$$

   In cases where the NDS algorithm depends on divide and conquer strategy, the MinCorr heuristic reduces the number of dominance checks in each subset that the data is divided into. And in the cases where the NDS algorithm lexically pre-sorts the data, MinCorr heuristic ensures that there are minimal conflicts in the pre-sorted data.

2. Minimum Absolute Correlation First (*MinAbsCorr*) - Although ordering variates on the minimum correlation first (MinCorr) strategy solves the most constrained objective first and makes it easier to solve the rest of the objectives, in some cases, this might not be true. We hypothesize that Min Correlation first strategy when used with certain NDS algorithms might cause increases domination checks because of interactions between groups of data. We therefore consider another strategy and call it *MinAbsCorr Heuristic*. This strategy considers a permutation of variates, $\Pi = [v_1, v_2, v_3, \ldots, v_M]$ such that:

$$\forall_{v_i, v_j} \ (i < j) \ \rightarrow \ \left(\sum_{h \in [v_1, v_2, \ldots, v_M]; h \neq v_i} |corr(h, v_i)| < \sum_{h \in [v_1, v_2, \ldots, v_M]; h \neq v_j} |corr(h, v_j)| \right).$$

3. Maximum Correlation First (*MaxCorr*) - Finally, considering the second worst case

stated earlier in this section and presented in Figure 4.2 which involves positively correlated data with many but small Pareto fronts in the NDS results, we propose a third strategy and call it *MaxCorr Heuristic*. Here, we consider a permutation of variates such that:

$$\forall_{v_i,v_j} \ (i < j) \ \rightarrow \left(\sum_{h \in [v_1,v_2,...,v_M]; h \neq v_i} corr(h, v_i) > \sum_{h \in [v_1,v_2,...,v_M]; h \neq v_j} corr(h, v_j) \right).$$

The following subsection provides a theoretical analysis of SD-NDS and describes how it optimizes the process of Non-Dominated Sorting.

Algorithmic Analysis of SD-NDS

As mentioned in the previous section, three heuristics are proposed for SD-NDS, MinCorr Rank, MinAbsCorr Rank and MaxCorr Rank. The experimental results presented in Chapter 5 show that MinCorr Rank has the best performance across different cases. Considering this, in this section, I provide a theoretical analysis of how SD-NDS achieves optimization in existing Non-Dominated Sorting Algorithms.

SD-NDS reduces the algorithmic complexity of non-dominated sorting algorithms in two ways: (1) reducing the number of dominance checks across partitions in divide and conquer based NDS algorithms and (2) reducing the cost of each dominance check. Lexically sorting on data which has the most negatively correlated variate first ensures that the local partitions in a divide and conquer style NDS algorithm have low negative correlations, this in turn reduces the number of redundant dominance checks across partitions in later stages. This is similar to solving the "most-constrained" variable first in CSP problems (Norvig and Intelligence, 2002).

The reduction in the cost of each dominance check by variate ordering can be clearly observed by looking at the following example:

Example: Consider an MOO problem which can be modeled by the SCM in Equation 4.1.

$$\begin{aligned} X &= \mathcal{N}(\mu = 0, \sigma = 1) \\ Y &= 2 \times X + \mathcal{N}(\mu = 0, \sigma = 0.4) \\ Z &= Y + \mathcal{N}(\mu = 0, \sigma = 2) \\ W &= -2 \times Y + \mathcal{N}(\mu = 0, \sigma = 0.8) \end{aligned} \tag{4.1}$$

Generating data with such a model gives us data with the correlation matrix presented in Table 4.1.

	X	Y	Z	W
X	1.000000	0.981054	0.708279	-0.965826
Y	0.981054	1.000000	0.718038	-0.984291
Z	0.708279	0.718038	1.000000	-0.698541
W	-0.965826	-0.984291	-0.698541	1.000000
Avg Corr	**0.43**	**0.42**	**0.43**	**-0.41**

Table 4.1: Example correlation matrix

From Table 4.1 it is evident that the optimal variate ordering suggested by my proposed MinCorr Rank heuristic would be $< W, Y, Z, X >$. Let us lexically sort the data using the best ($< W, Y, Z, X >$) and the worst ($< X, Z, Y, W >$) variate orders. The results of this operation are presented in Figure 4.3.

We can clearly see how the cost of each dominance check can be reduced by varying the order in which these objectives are visited when performing a dominance check. If we look at the worst variate order according to minCorr heuristic $< X, Z, Y, W >$, it is obvious that we will have to compare all four variates to determine the dominance relation between two tuples. But when we order the variates according to the best variate order, $< W, Y, Z, X >$, there is a high probability of determining the dominance condition after comparing the first two (W, Y) or three variates (W, Y, Z). In other words, we are reducing the value of M in the worst case time complexity of NDS algorithms, $(O(MN^2))$.

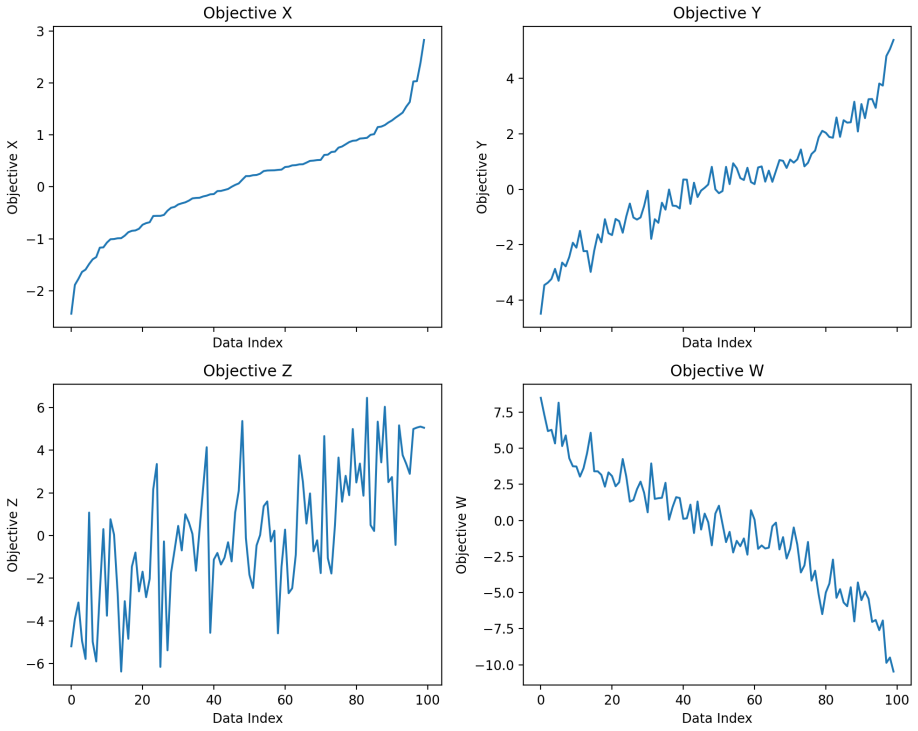


Figure 4.3: Trends in each objective in the problem defined in Equation 4.1

Placing the highest negative correlation variate first in the objective vector also reduces the number of inter-partition dominance checks by reducing the number of conflicts between these partions, consequently reducing the complexity of the N^2 term in the worst case time complexity of NDS algorithms, ($O(MN^2)$).

Having presented the proposed methods, the next section details how we setup experiments to evaluate the performance of the proposed methodologies presented in this thesis and then we present and analyze the results obtained.

Chapter 5

EXPERIMENTAL EVALUATION

In order to properly analyze our proposed methods to speed up Non-Dominated sorting, D-NDS and SD-NDS, I first run experiments on synthetic data. Experiments are run on three kinds of synthetic data:

- Positively correlated objectives - These experiments will help us analyze how our proposed method performs in cases where the objectives are non-conflicting with each other.
- Negatively correlated objectives - These experiments will help us analyze how our proposed method performs in cases where all the objectives in the optimization problem are conflicting/competing with one another.
- Positively and Negatively (Mixed) correlated objectives - This presents a realistic scenario where we have some objectives that are conflicting and some that are not conflicting with one another.

The rest of our experiment parameters are presented in Table 5.1.

Experiment Parameter	**Values**
Number of Objectives	{2, **5**, 10}
Correlation between Objectives	{Positive, **Mixed**, Negative}
Avg Correlation Spread	{**High**, Low}
Sample Size	{2048, **4096**, 16384}

Table 5.1: Experiment Settings

To run our experiments, we used Chameleon Cloud compute instances(Keahey *et al.*, 2020). The hardware configuration on these instances is: AMD EPYC 7763 64-Core

Processor, with 2 sockets and 64 cores per socket and with 2 threads per core. The maximum CPU frequency is 3.529GHz. The instance had 251.28GiB of usable memory. We used Ubuntu 24.04 operating system and used Python 3.12 to run our experiments.

Synthetic Data Generation

Before we dive deep into the experimental setup for the analysis of our proposed algorithms, let us first understand how we generate data for the synthetic experiments on D-NDS and SD-NDS.

To generate synthetic data for the experiments on the two proposed optimizations on existing NDS algorithms, D-NDS and SD-NDS, I used the Multivariate Normal Data Generator function provided by numpy (Harris *et al.*, 2020). This function generates multivariate data that has a similar distribution characteristics as those provided by the user. This function requires users to input the covariance matrix of the multivariate distribution and the means of each of the variates. The function then generates data based on these inputs. For all datasets, I used zero mean and unit variance for individual variates. The covariance matrix input to the numpy function was changed based on how the average correlation spread and the correlation type was varied as specified in Table 5.1

Such a data generation function made it easier to have control on the correlations between the variates (objectives) in my experiments.

We now get into the details of how we setup experiments, our default case and the case studies we perform to better understand how our proposed methods D-NDS and SD-NDS performs in specific situations.

Experiments on D-NDS

Experimental Setup

The default setup for our experimental analysis on D-NDS is using *five optimization objectives* with *mixed correlations* between them, a sample size of 4096.

Results

Table 5.2: Results for experiments on D-NDS Algorithm

Number of Objectives	Correlation between Objectives	Avg Correlation Spread between Objectives	Sample Size	Average Time Gain					
				ENS-SS	ENS-BS	FastNDS	DDA-NS	DDA-ENS	T-ENS
2	Positive & Negative	-	4096	7.95%	-88.87%	72.41%	32.48%	28.78%	58.61%
5	Mixed	High	4096	-165.28%	-111.58%	-244.11%	-1442.60%	-799.45%	-744.71%
10	Mixed	High	4096	-156.16%	-148.44%	-374.42%	-1351.33%	-877.44%	-1065.80%
5	Positive	High	4096	-147.91%	-172.20%	37.69%	-153.36%	-88.10%	-313.40%
5	Negative	High	4096	-142.52%	-142.25%	-508.39%	-2187.34%	-1187.89%	-1147.10%
5	Mixed	Low	4096	-127.29%	-101.86%	-227.81%	-1048.91%	-608.80%	-677.09%
5	Mixed	High	2048	-162.54%	-119.16%	-247.70%	-2242.72%	-1118.75%	-717.23%
5	Mixed	High	16384	-175.00%	-138.24%	-238.47%	-1875.83%	-962.82%	-706.41%

To analyze the performance of our D-NDS approach, we consider baseline NDS algorithms and the D-NDS versions of these algorithms. We run both of these on our synthetically generated data and calculate time gain with using Equation 5.1. Positive time gain means, D-NDS version of an NDS algorithm has lower run-time than the corresponding baseline version of the NDS algorithm.

$$\text{TimeGain} = \frac{Time_{\text{Baseline}} - Time_{\text{D-NDS}}}{Time_{Baseline}} \times 100 \tag{5.1}$$

The results presented in Table 5.2 show that D-NDS does not improve the run-time in any case. However, for the 2 objective experiments, we see that there is an improvement. This is because of the fact that as the number of objectives increase, the time gained by dividing the population into subsets and performing NDS is less than the time taken by the

merge operation to merge all the NDS orders.

Experiments on SD-NDS

Experimental Setup

The default setup for our experimental analysis on SD-NDS is using *five optimization objectives* with *mixed correlations* between them, a sample size of 4096. We also want to ensure the variance of the per-variate average correlations Equation 5.2 is high.

$$\mathrm{Var}\big(\mathrm{Corr}^{\text{avg}}_{\text{obj}_1}, \mathrm{Corr}^{\text{avg}}_{\text{obj}_2}, \ldots, \mathrm{Corr}^{\text{avg}}_{\text{obj}_5}\big) \tag{5.2}$$

where

$$\mathrm{Corr}^{\text{avg}}_{\text{obj}_j} = \frac{1}{4}\sum_{\substack{i=1 \\ i\neq j}}^{5} \mathrm{Corr}(obj_j, obj_i) \tag{5.3}$$

For five variates, we run experiments on all possible (120) variate orderings and analyze the performance of each of the variate orderings and how our three proposed heuristics rank each of them. For 10 variate experiments, we sample 100 random variate orderings, and obtain the top 25 and bottom 25 from each of our proposed heuristics. This will give us a total of *200* variate orderings. With the experimental setup established, we now report the results and examine the comparative performance of the heuristics.

Results

In order to evaluate the performance of SD-NDS on our datasets, we consider three metrics:

1. $Rank\%$ - Rank of the selected permutation relative to all the permutations considered. This value is 120 for 5 variate datasets and 200 for 10 variate datasets.

2. $AvgGain$ - This is the percentage difference between the average time of all the variate orders compared to the time of the selected variate order and relative to the average time of all variate orders($\frac{avgTime-t}{avgTime}$ x 100).

3. $DistToOptimal$ - This is the percentage difference between the run-time of the selected variate order and the best variate order relative to the difference between the best and worst performing variate order: $\frac{t-minTime}{maxTime-minTime}$x100). The closer this value is to zero, the better our heuristic is to predicting the best variate order.

The computational time complexity of computing correlation is linear in the population size (that is of the order $O(N)$), this value is comparatively lower than the worst case time complexity of NDS algorithm which is of quadratic order in population size, N, as shown in Table 5.7.

The time taken to compute each heuristic value - MinCorr, MaxCorr and MinAbsCorr - on 5 objective datasets was in the order of $milli - seconds$. The exact value was around 2 milliseconds. This was the same for 10 objective case. However, computing all the permutations of variates was expensive compared to the variate rank computation. But we have to keep in mind that we do not need *all* the variate permutations, we only need the *best* variate permutations. This heuristic computation time is negligible and has therefore been excluded from the result metric computation in all the results tables in this section.

Table 5.3 gives a high-level overview of the results obtained upon experimental analysis of SD-NDS.

The results for this default experiment setup, [5 objectives, mixed correlations, high variance between average correlations (given by Eq. 5.2), 4096 sample size] are presented in Table 5.4.

From the results of our default experimental setup, we can observe a clear performance gain on the pre-sorting based NDS algorithms. We use the same synthetic data as we used

Table 5.3: SD-NDS Results Overview using MinCorr Heuristic

Number of Objectives	Correlation between Objectives	Avg Correlation Spread between Objectives	Sample Size	AvgGain using MinCorr Heuristic							
				ENS-SS	ENS-BS	DCNS-SS	DCNS-SS-WS	FastNDS	DDA-NS	DDA-ENS	T-ENS
2	Positive and Negative	-	4096	-0.17%	0.29%	-2.53%	-0.73%	-0.13%	-0.68%	-0.40%	0.25%
5	Mixed	High	4096	10.96%	4.24%	8.20%	8.50%	-6.00%	-4.54%	-6.40%	11.82%
10	Mixed	High	4096	12.46%	11.91%	13.05%	10.47%	2.48%	-0.71%	0.80%	30.53%
5	Positive	High	4096	10.98%	4.77%	6.11%	6.44%	0.13%	3.35%	-3.21%	22.21%
5	Negative	High	4096	2.55%	2.73%	2.84%	1.88%	2.66%	-1.57%	-1.21%	0.64%
5	Mixed	Low	4096	11.63%	2.24%	4.95%	3.02%	-11.64%	-0.10%	-1.26%	17.97%
5	Mixed	High	2048	15.61%	3.00%	7.99%	8.24%	-2.78%	-2.69%	-10.04%	16.22%
5	Mixed	High	16384	12.55%	7.95%	8.87%	8.26%	0.93%	-0.04%	-3.89%	7.66%

Table 5.4: Results for our default experiment case

Timing Results (5 Dims, 4K Data, mixed Correlations)								
RANK %	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	27.3%	35.8%	22.5%	22.5%	55.3%	57.0%	66.2%	47.0%
minAbsCorr	78.0%	65.8%	75.8%	75.0%	49.0%	76.2%	69.3%	41.8%
maxCorr	88.2%	85.8%	88.8%	88.7%	41.0%	67.2%	62.8%	27.3%
AVG.GAIN	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	11.0%	4.2%	8.2%	8.5%	-6.0%	-4.5%	-6.4%	11.8%
minAbsCorr	-14.3%	-6.0%	-7.7%	-6.7%	2.1%	-6.4%	-3.6%	16.8%
maxCorr	-25.6%	-12.9%	-13.8%	-13.2%	3.8%	-2.9%	-3.6%	29.7%
DIST to OPTIMAL	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	16.7%	28.0%	22.0%	27.2%	44.3%	52.1%	62.8%	35.5%
minAbsCorr	63.4%	55.0%	62.8%	61.7%	33.3%	59.6%	54.4%	30.7%
maxCorr	75.9%	71.5%	78.9%	77.1%	29.3%	51.4%	57.2%	18.8%

in D-NDS experiments presented in Table 5.2.

We now perform a critical analysis of the performance of our proposed method by varying different experimental parameters as specified in Table 5.1.

Varying the correlation between objectives

In the default experiment case, we used objective variables which were positively correlated with some objectives and negatively correlated with other objectives. We now examine how the variate ordering heuristics perform when we use objectives that are either all conflicting (all negative correlations) and all non-conflicting (all positive correlations). The results for this experiment are presented in Table 5.5. In real-world scenarios, if two objectives are positively correlated, then if both of them are being minimized (or maximized), then

they do not conflict with one another and make the optimization problem easy, but, when the correlation between them is negative while the objective remains the same, then the optimization problem becomes much more difficult. This is because the two objectives are now conflicting with one another, this is because as one of the objectives is minimized, the other is automatically maximized (because they are negatively correlated with one another).

From Table 5.5 we can see that the performance of our proposed SD-NDS method obtained on the base case in Table 5.4 is maintained when we make the correlations positive. We also see that both the heuristics $minAbsCorr$ and $minCorr$ perform similarly in the positive correlations dataset. This is because of the fact that when all the values in the correlation matrix are positive, the correlation matrix is the same as the absolute value of the correlation matrix, making the $minCorr$ and $minAbsCorr$ heuristic give out the same ranking for variate orders.

Varying the Spread of Average Correlation

We observe in the results presented in Table 5.4 that minCorr heuristic gives a consistent gain in run-time of NDS algorithms. But in that scenario, we stated that the variance of the average correlations given by Equation 5.2 is higher, that is the average correlations for each variate are more spread apart. On another extreme, we have cases when the average correlations of variates given by Equation 5.3 are close together. In this case we therefore hypothesize that the time gain when we use the optimal vs suboptimal variate ordering will not be as significant as we saw in Table 5.4.

The experimental results for the case we described above are presented in Table 5.6. We can clearly see that the average performance gain by using SD-NDS decreased in ENS-BS, DCNS-SS, DCNS-SS-WS algorithms, whereas for ENS-SS NDS algorithm, the performance remained almost the same, but surprisingly for the tree based method, T-ENS, the performance gain under closer average correlations situation was higher when compared

Table 5.5: Analysis of the impact of variate ordering as we change the types of correlation between objectives

Timing Results (5 Dims, 4K Data, Positive Correlations)								
RANK %	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	12.0%	16.4%	14.0%	15.4%	30.0%	16.9%	46.3%	16.8%
minAbsCorr	12.0%	16.4%	14.0%	15.4%	30.0%	16.9%	46.3%	16.8%
maxCorr	56.0%	42.0%	44.0%	32.5%	22.4%	38.0%	25.3%	16.1%
AVG.GAIN	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	11.0%	4.8%	6.1%	6.4%	0.1%	3.4%	-3.2%	22.2%
minAbsCorr	11.0%	4.8%	6.1%	6.4%	0.1%	3.4%	-3.2%	22.2%
maxCorr	-25.7%	-5.2%	-4.1%	-1.1%	2.8%	-1.2%	1.3%	20.3%
DIST to OPTIMAL	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	12.0%	24.1%	21.3%	23.0%	38.3%	18.1%	55.6%	13.3%
minAbsCorr	12.0%	24.1%	21.3%	23.0%	38.3%	18.1%	55.6%	13.3%
maxCorr	87.3%	57.7%	56.3%	44.2%	28.7%	37.2%	34.5%	17.2%
Timing Results (5 Dims, 4K Data, Negative Correlations)								
RANK %	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	12.2%	6.3%	4.2%	16.6%	23.4%	37.7%	37.8%	28.5%
minAbsCorr	40.1%	53.7%	53.3%	54.0%	22.1%	34.3%	28.8%	35.9%
maxCorr	40.1%	53.7%	53.3%	54.0%	22.1%	34.3%	28.8%	35.9%
AVG.GAIN	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	2.5%	2.7%	2.8%	1.9%	2.7%	-1.6%	-1.2%	0.6%
minAbsCorr	-1.2%	-2.4%	-2.1%	-3.7%	3.2%	-0.4%	0.3%	-7.2%
maxCorr	-1.2%	-2.4%	-2.1%	-3.7%	3.2%	-0.4%	0.3%	-7.2%
DIST to OPTIMAL	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	26.5%	19.5%	12.8%	26.7%	37.7%	41.8%	46.6%	40.2%
minAbsCorr	56.4%	74.5%	78.3%	80.5%	32.4%	35.7%	40.4%	55.9%
maxCorr	56.4%	74.5%	78.3%	80.5%	32.4%	35.7%	40.4%	55.9%

to our base case results we presented in Table 5.4.

Varying the sample size

With Pareto-dominance based genetic algorithms like NSGA family, NPGA, SPEA, SPEA2, one of the most time taking parts of a run of those algorithms is the NDS part of the optimization run. And this often gets worse as the number of samples in the population increases. We show this empirically in Figure 1.1. Theoretically, this observation becomes obvious when we look at the time worst-case complexities of some algorithms presented in Table 5.7.

We can clearly observe that the time complexity of Non-Dominated Sorting algorithms is dependent on two things: the number of objectives, M and the number of samples, N.

Table 5.6: Experimental results on synthetic data when objectives have close average correlations

Timing Results (5 Dims, 4K Data, Mixed Correlations)								
RANK %	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	8.7%	25.8%	8.6%	19.0%	30.5%	34.9%	33.4%	18.5%
minAbsCorr	50.6%	45.2%	45.7%	26.7%	31.2%	14.8%	13.1%	21.8%
maxCorr	52.0%	46.2%	43.1%	40.3%	22.8%	33.9%	23.2%	26.5%
AVG.GAIN	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	11.6%	2.2%	5.0%	3.0%	-11.6%	-0.1%	-1.3%	18.0%
minAbsCorr	-26.7%	-7.8%	-3.3%	1.2%	1.4%	2.5%	4.0%	9.7%
maxCorr	-27.8%	-7.1%	-3.9%	-1.7%	3.4%	0.4%	1.7%	7.7%
DIST to OPTIMAL	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	11.2%	34.0%	16.3%	28.5%	41.7%	32.2%	42.1%	26.2%
minAbsCorr	71.9%	71.9%	64.8%	42.2%	29.1%	17.4%	20.7%	33.6%
maxCorr	74.1%	70.0%	64.0%	56.1%	27.7%	28.9%	28.6%	36.3%

Table 5.7: Time and Space complexities of some well-known NDS algorithms.

Algorithm	**Space Complexity**	**Time Complexity**	
		Best Case	*Worst Case*
Naive (Srinivas and Deb, 1994)	$O(N)$	$O(MN^2)$	$O(MN^3)$
ENS-SS (Zhang *et al.*, 2015)	$O(1)$	$O(MN\sqrt{N})$	$O(MN^2)$
ENS-BS (Zhang *et al.*, 2015)	$O(1)$	$O(MNlogN)$	$O(MN^2)$
DCNS-SS (Mishra *et al.*, 2019)	$O(1)$	$O(MNlogN)$	$O(MN^2)$
DCNS-SS-WS (Mishra *et al.*, 2019)	$O(N)$	$O(MNlogN)$	$O(MN^2)$
FastNDS((Deb *et al.*, 2002))	$O(N^2)$	$O(MN^2)$	$O(MN^2)$
DDA-NS (Zhou *et al.*, 2016)	$O(N^2)$	$O(MNlogN)$	$O(MN^2 + N^3)$
DDA-ENS (Mishra and Senwar, 2020)	$O(N^2)$	$O(MNlogN)$	$O(MN^2)$
T-ENS (Zhang *et al.*, 2018)	$O(MN)$	$O(\frac{MNlogN}{logM})$	$O(MN^2)$

And in all cases, the time complexity has a higher degree in the number of samples, N. To study how our algorithm performs on small and large sample size data, we also run experiments on 2,048 and 16,384 samples on our default experimental setting. Results are presented in Table 5.8.

From the results obtained in the 2K and 16K sample size experiments, we can conclude that as we increase the sample size, the improvement by variate ordering through our proposed method, SD-NDS remains consistent. Although the percentage of run-time decreased remains the same, the absolute amount of run-time reduced increases as the total run-time of NDS itself increases as the population size increases.

Table 5.8: Analysis of the impact of variate ordering as we change the sample size

Timing Results (5 Dims, 2K Data, mixed Correlations)								
RANK %	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	12.3%	41.7%	19.7%	23.7%	53.0%	40.3%	69.0%	43.0%
minAbsCorr	66.0%	75.5%	61.8%	66.8%	30.0%	54.2%	44.2%	31.0%
maxCorr	93.0%	84.3%	75.3%	77.5%	44.8%	56.2%	59.5%	26.2%
AVG.GAIN	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	15.6%	3.0%	8.0%	8.2%	-2.8%	-2.7%	-10.0%	16.2%
minAbsCorr	-4.8%	-8.1%	-2.3%	-4.3%	4.8%	-2.4%	1.3%	26.4%
maxCorr	-28.5%	-11.1%	-7.1%	-8.6%	1.8%	0.1%	-0.5%	27.1%
DIST to OPTIMAL	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	10.0%	30.5%	18.3%	22.4%	35.2%	23.8%	47.5%	27.4%
minAbsCorr	45.9%	59.7%	45.4%	55.1%	17.2%	31.2%	26.5%	20.0%
maxCorr	76.6%	66.5%	57.4%	66.6%	25.3%	24.7%	34.4%	16.9%
Timing Results (5 Dims, 16K Data, mixed Correlations)								
RANK %	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	18.9%	18.9%	18.9%	23.3%	30.0%	53.3%	58.9%	58.9%
minAbsCorr	80.0%	70.0%	73.3%	75.6%	80.0%	64.4%	51.1%	32.2%
maxCorr	95.6%	78.9%	93.3%	85.6%	76.7%	60.0%	48.9%	31.1%
AVG.GAIN	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	12.5%	8.0%	8.9%	8.3%	0.9%	0.0%	-3.9%	7.7%
minAbsCorr	-9.3%	-1.8%	3.8%	-4.2%	-4.1%	-3.9%	0.7%	15.3%
maxCorr	-34.9%	-8.8%	-14.2%	-9.2%	0.0%	0.0%	1.2%	26.5%
DIST to OPTIMAL	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	10.7%	9.8%	12.9%	12.7%	19.5%	44.3%	53.4%	37.1%
minAbsCorr	76.2%	47.6%	56.4%	59.6%	69.9%	66.6%	26.1%	41.8%
maxCorr	98.5%	73.0%	91.3%	78.0%	55.2%	45.6%	28.2%	20.9%

Varying the number of objectives

The second factor impacting the time complexities of the NDS algorithms presented in Table 5.7 is the number of objectives in the optimization problem, M. We study how the increase/decrease of the number of objectives impacts the gain in time produced by using variate ordering proposed by SD-NDS. We run experiments on synthetic data containing 2 and 10 objectives and then compare the results of these two sets of experiments with our base case which contains 5 objectives. Results are presented in Table 5.9.

From Table 5.9, it is clear that 2 objective optimization problems do not get any advantage when using variate ordering. This is because in 2 variate optimization problems in continuous objective spaces, it is highly probable that both the objectives are required for determining the dominance relationship between two tuples.

Table 5.9: Comparison of the performance gain obtained as we change the number of objectives

Timing Results (2 Dims, 4K Data, Positive and Negative Correlations)								
RANK %	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	90.0%	50.0%	90.0%	90.0%	70.0%	60.0%	70.0%	70.0%
minAbsCorr	90.0%	50.0%	90.0%	90.0%	70.0%	60.0%	70.0%	70.0%
maxCorr	77.5%	72.5%	77.5%	85.0%	72.5%	65.0%	80.0%	70.0%
AVG.GAIN	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	-0.2%	0.3%	-2.5%	-0.7%	-0.1%	-0.7%	-0.4%	0.2%
minAbsCorr	-0.2%	0.3%	-2.5%	-0.7%	-0.1%	-0.7%	-0.4%	0.2%
maxCorr	0.1%	-0.1%	-0.5%	-0.5%	0.2%	0.4%	-1.1%	0.3%
DIST to OPTIMAL	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	80.0%	0.0%	80.0%	80.0%	40.0%	20.0%	40.0%	40.0%
minAbsCorr	80.0%	0.0%	80.0%	80.0%	40.0%	20.0%	40.0%	40.0%
maxCorr	55.0%	45.0%	55.0%	70.0%	45.0%	30.0%	60.0%	40.0%
Timing Results (10 Dims, 4K Data, mixed Correlations)								
RANK %	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	19.0%	19.9%	14.0%	12.4%	43.8%	54.5%	40.0%	24.4%
minAbsCorr	50.2%	58.7%	55.1%	60.8%	44.7%	53.0%	62.3%	64.8%
maxCorr	92.0%	97.0%	91.2%	95.6%	46.6%	55.4%	48.1%	33.7%
AVG.GAIN	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	12.5%	11.9%	13.1%	10.5%	2.5%	-0.7%	0.8%	30.5%
minAbsCorr	-0.8%	-2.4%	-1.1%	-2.0%	2.2%	-0.4%	-1.4%	-13.3%
maxCorr	-21.5%	-24.1%	-14.5%	-18.7%	2.7%	-0.6%	0.3%	15.7%
DIST to OPTIMAL	**ENS-SS**	**ENS-BS**	**DCNS-SS**	**DCNS-SS-WS**	**FastNDS**	**DDA-NS**	**DDA-ENS**	**T-ENS**
minCorr	14.4%	16.8%	15.6%	15.8%	21.6%	38.9%	29.9%	17.4%
minAbsCorr	43.4%	48.7%	48.4%	48.3%	21.9%	36.7%	41.4%	44.9%
maxCorr	85.9%	92.9%	81.5%	90.2%	24.4%	39.4%	34.9%	27.0%

Experimental evaluation of SD-NDS on NSGA-II

Synthetic Multi-Objective Optimization Problems

To evaluate the performance of our optimized NDS methods on the overall NSGA-II runtime, we use the set of DTLZ test-problems formulated in (Deb *et al.*, 2005). The DTLZ problem suite was originally proposed by (Deb *et al.*, 2005) to test the effectiveness of genetic algorithms both in terms of convergence time and the quality of solutions they converge to. Each problem in this problem suite has varying kinds of pareto fronts. For example, the solutions of DTLZ1 are on a flat, planar pareto front, but the solutions of DTLZ2 are on a convex surface and the solutions of DTLZ7 are on a segmented pareto

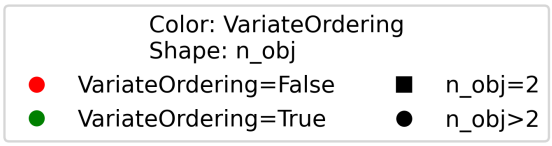


Figure 5.1: Legend for the experiment result graphs presented in this section.

front.

The DTLZ synthetic problem suite makes it easy to configure different MOO problems. Specifically we vary both the number of objectives (n_obj) and number of decision variables (n_var). Adding higher number of objectives makes the pareto-optimal frontier larger and hence warrants for a larger population size, on the other hand, as we increase the number of decision variables, it makes it easier for the genetic algorithm to better tune the parameters to arrive the the optimal pareto frontier of solutions in the MOO.

We run experiments on DTLZ1 through DTLZ7 proposed by (Deb *et al.*, 2005) varying the parameters as mentioned in Table 5.10.

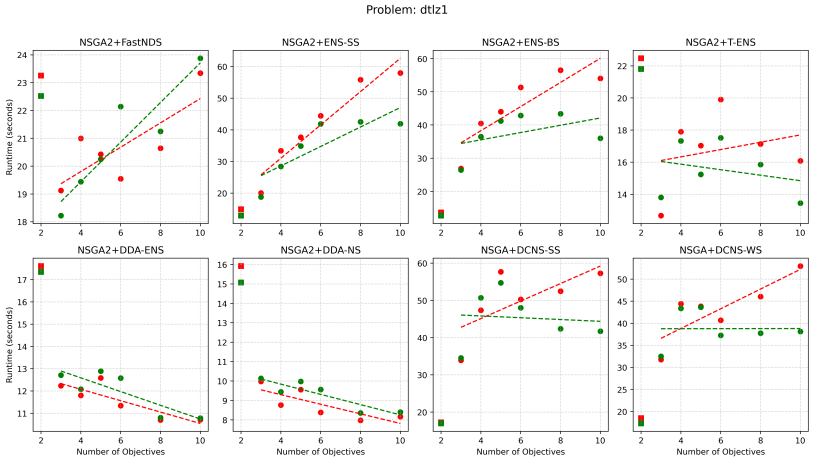
Parameter	**Values**
n_obj	[2,3,4,5,6,8,10]
n_var	[10,50,100]
pop_size	[**256**, 512, 1024, 2048]
n_function_evals	[30K, **60K**, 120K]

Table 5.10: Experiment Parameter settings for experiments using synthetic MOO problems

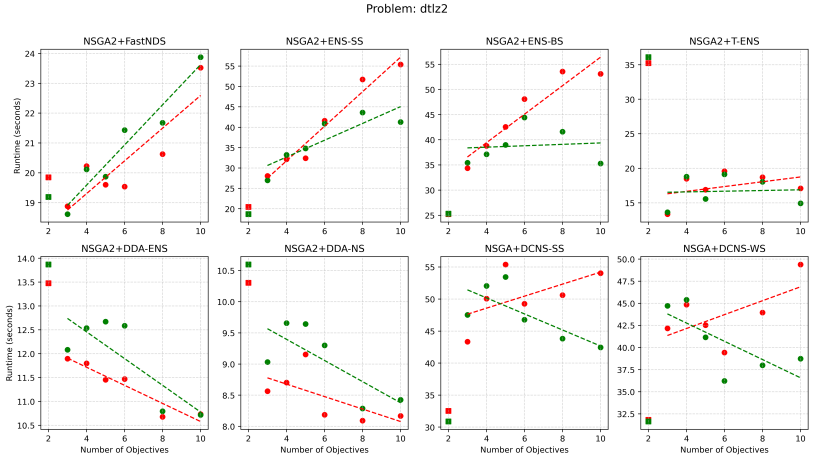
For all the experiments on synthetic MOO problems from (Deb *et al.*, 2005) our default experimental setup uses a sample size of 256 and each optimization run is run for 60,000 function evaluations.The results for this default case are shown in Figure 5.2.

The legend for all the experiment results presented in this section is shown in Figure 5.1.

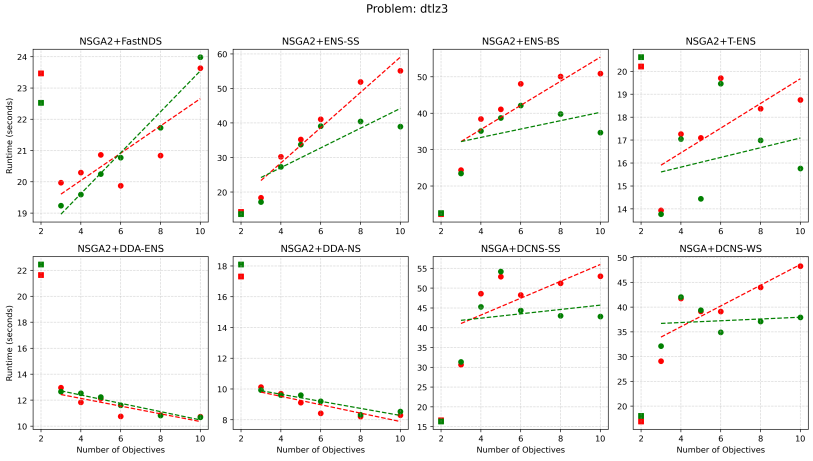
To see the impact of increasing population size on the performance improvement we see when using SD-NDS in Figure 5.2, we run experiments varying the population size



(a) DTLZ1

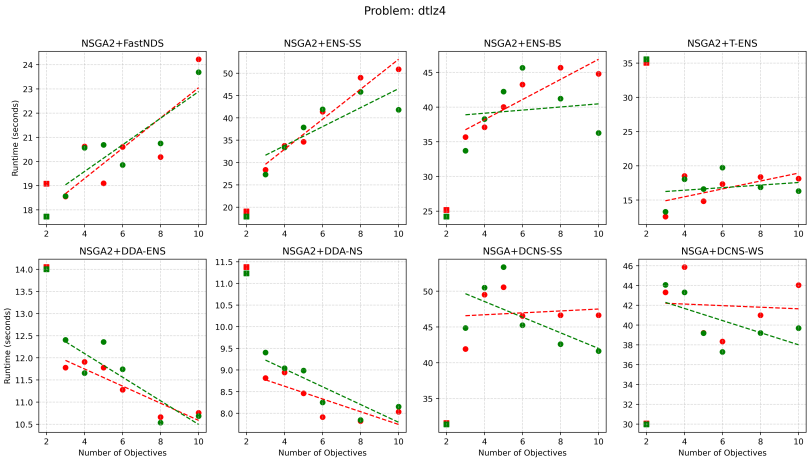


(b) DTLZ2

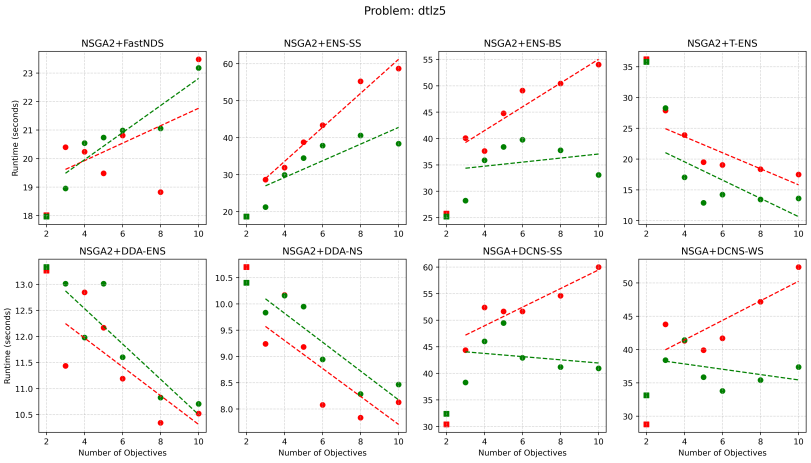


(c) DTLZ3

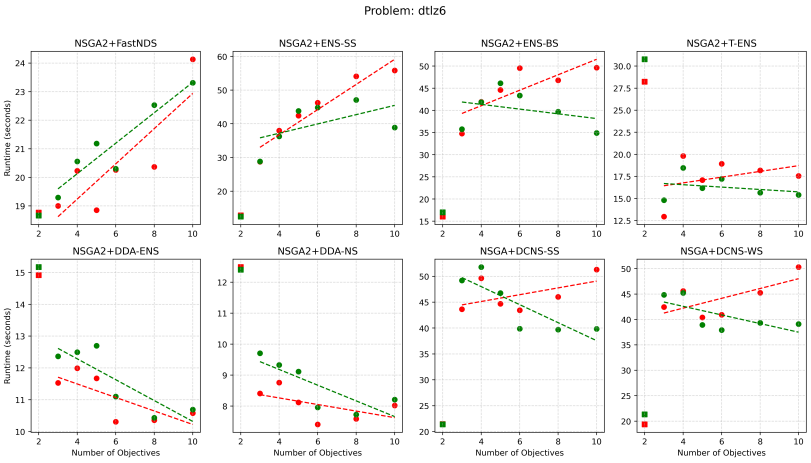
Figure 5.2: NSGA-II + SD-NDS results for [pop_size, $n\_function\_evals$] = [256, 60K] (DTLZ1–7, $n_{var} = 50$).



(d) DTLZ4

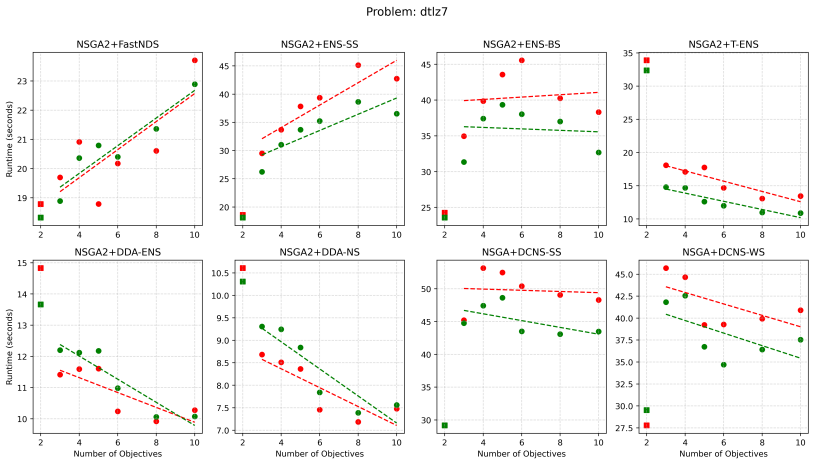


(e) DTLZ5



(f) DTLZ6

Figure 5.2: NSGA-II + SD-NDS results (continued).



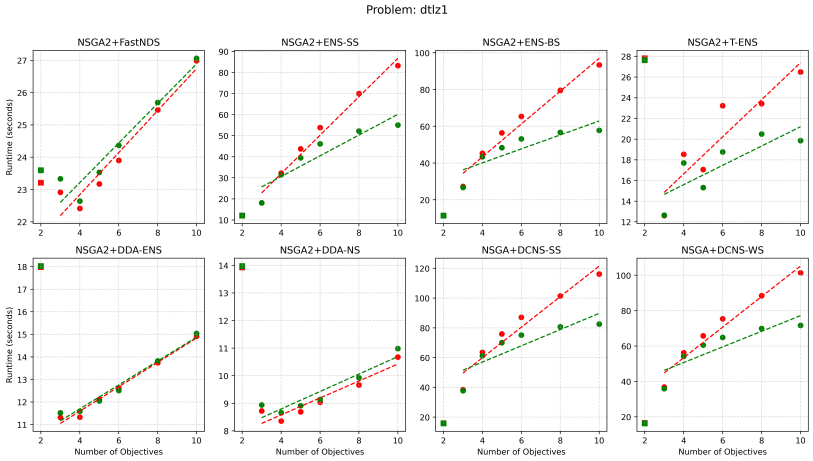
(g) DTLZ7

Figure 5.2: NSGA-II + SD-NDS results (continued).

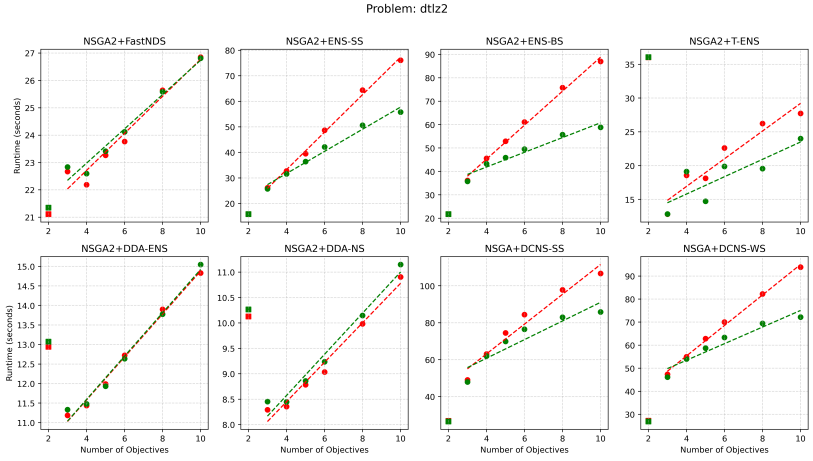
(pop_size) while keeping the number of function evaluations ($n\_function\_evals$) constant at $60K$. The results for this analysis is shown in Figures 5.3, 5.4, 5.5.

As seen from our results in Table 5.8 and Table 5.4, the time gain remains stable as we increase the population size but as we have empirically seen in Figure 1.1 and theoretically in Table 5.7, we can clearly see an increase in the run-time of NSGA-II as we increase the population size.

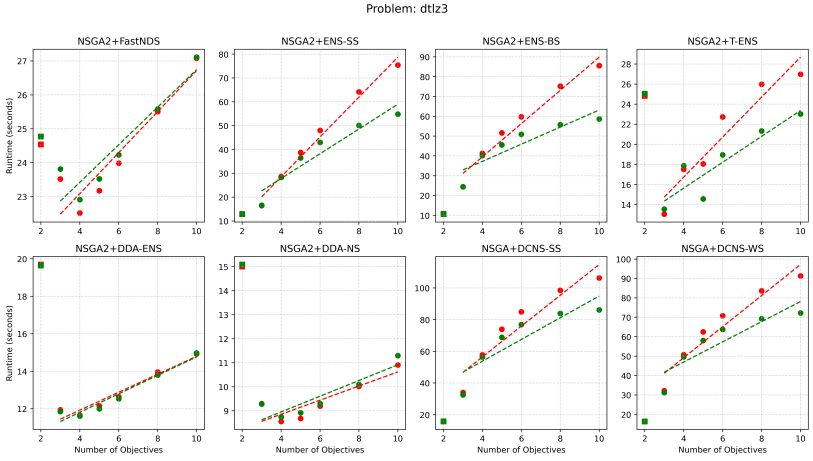
Another factor that determines the number of times NDS operator is used in a single optimization run and therefore, the run-time of an optimization run using NSGA-II is the number of generations that the genetic algorithm is run for. In our setup, we set this by using the "$n\_function\_evals$" (the number of total objective function evaluations). As mentioned earlier, we use the default value of 60K for this parameter, but we also run experiments for 30K and 120K function evaluations to see the impact of increasing the number of generations in an NSGA-II run using our proposed optimization method. Results for this analysis are showcased in Figures 5.6 and 5.7.



(a) DTLZ1

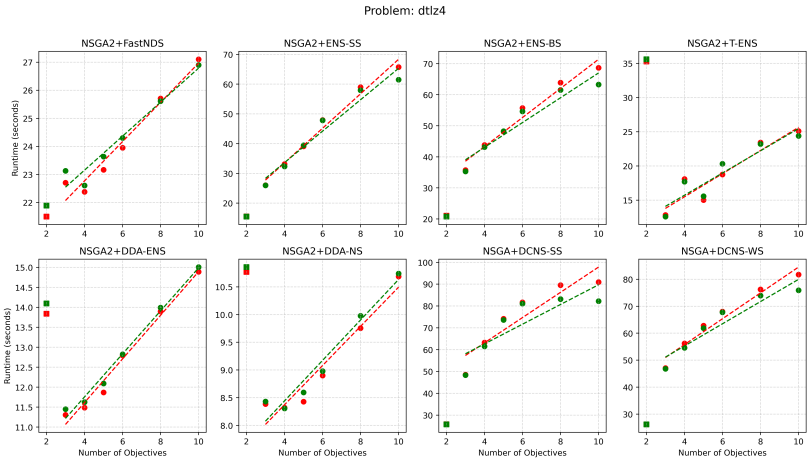


(b) DTLZ2

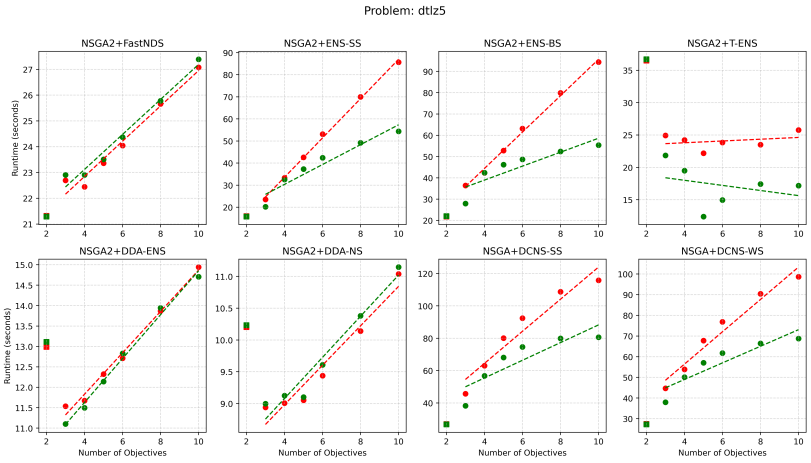


(c) DTLZ3

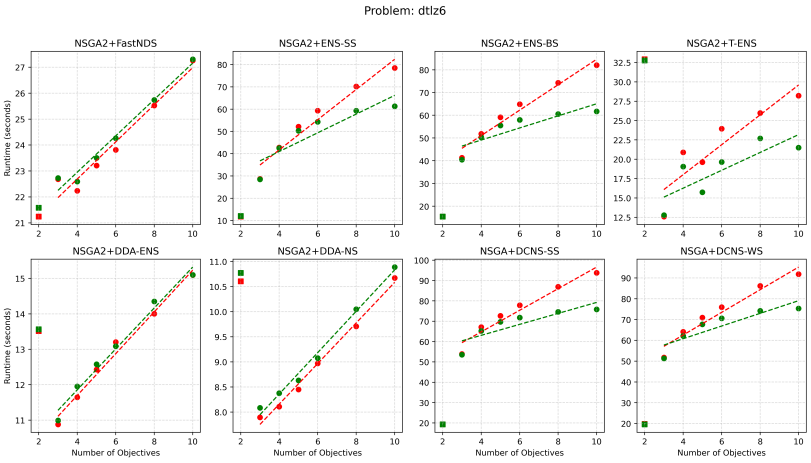
Figure 5.3: NSGA-II + SD-NDS results for [pop_size, $n\_function\_evals$] = [512, 60K] (DTLZ1–7, $n_{var} = 50$).



(d) DTLZ4

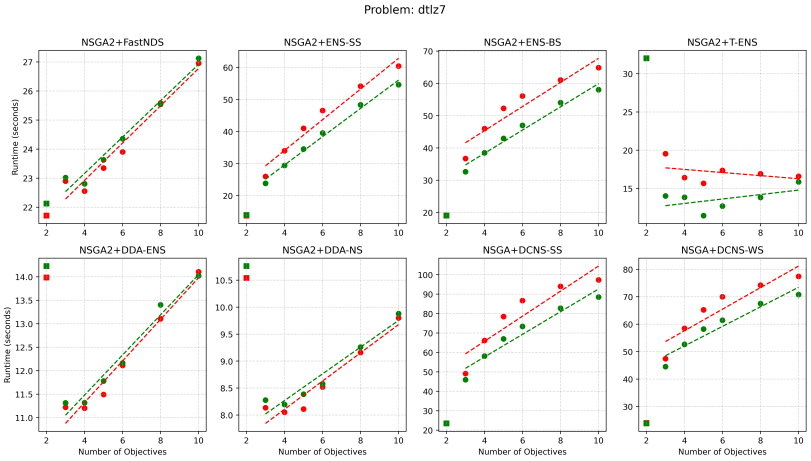


(e) DTLZ5



(f) DTLZ6

Figure 5.3: NSGA-II + SD-NDS results (continued).



(g) DTLZ7

Figure 5.3: NSGA-II + SD-NDS results (continued).

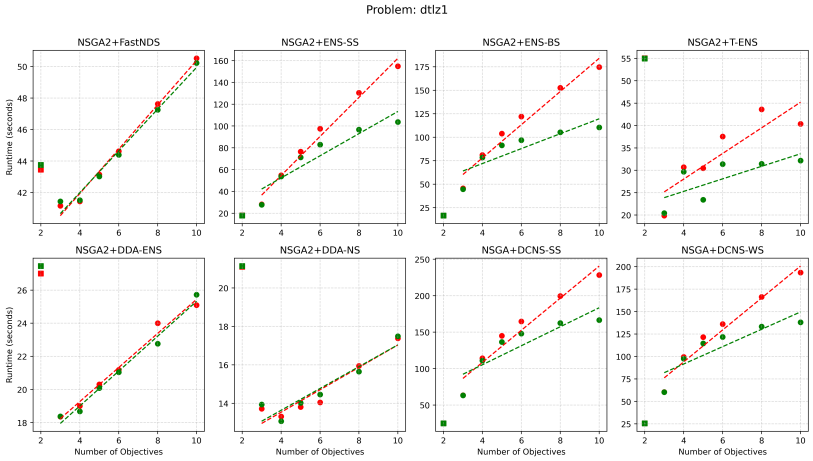
Real-World MOO Problems

For testing our proposed speed-up on NSGA-II using SD-NDS, on real-world optimization problems, we refer to (Tanabe and Ishibuchi, 2020) for real-world problem definitions. We used the problem definition provided by authors of (Tanabe and Ishibuchi, 2020) in their code-base on Github [1] and converted them to be compatible with pymoo (Blank and Deb, 2020). We used these new problem classes in our experiments.

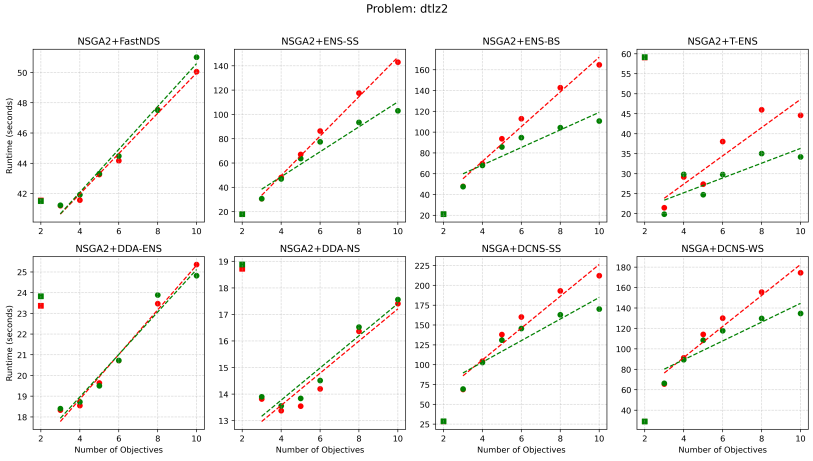
An overview of the results on the RE Problem set (Tanabe and Ishibuchi, 2020) is presented in Table 5.11. The results presented in Table 5.11 are for the experiment set when the parameters [pop_size, n_function_evals] are $[512, 60\text{K}]$.

In this section we present our experiment results on integrating our proposed SD-NDS method in NSGA-II. As specified in Table 5.10, we analyze the performance of our proposed SD-NDS method varying number of objectives (n_obj), number of decision variables (n_var), population size (pop_size) and number of function evaluations ($n\_function\_evals$).

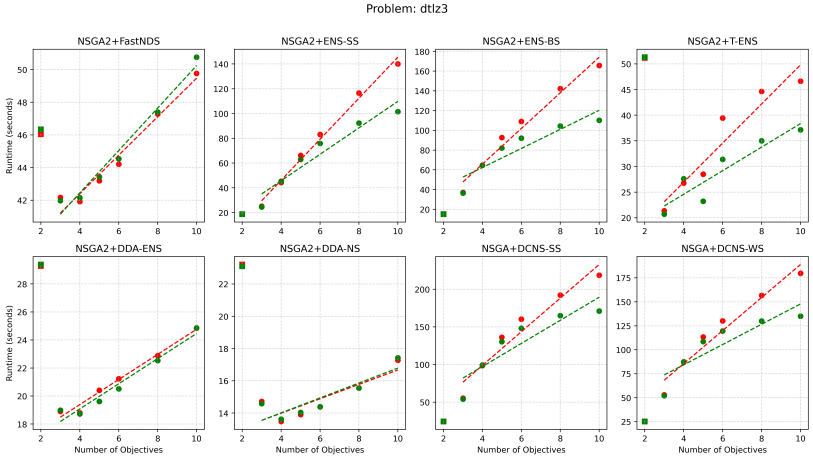
[1] https://github.com/ryojitanabe/reproblems



(a) DTLZ1

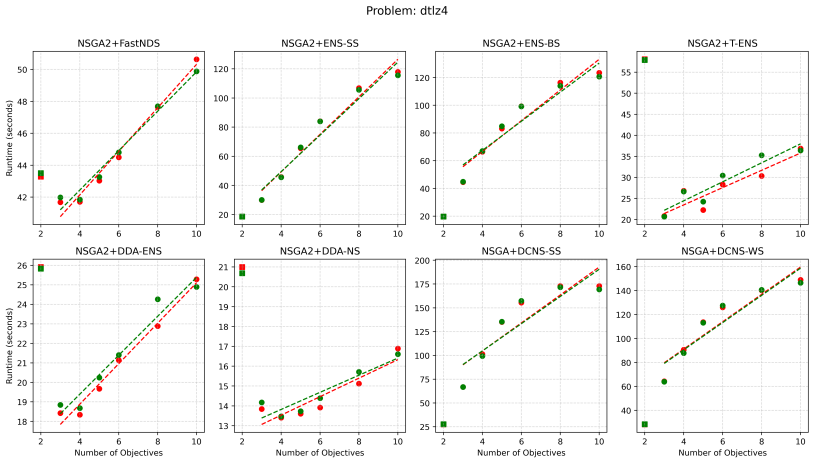


(b) DTLZ2

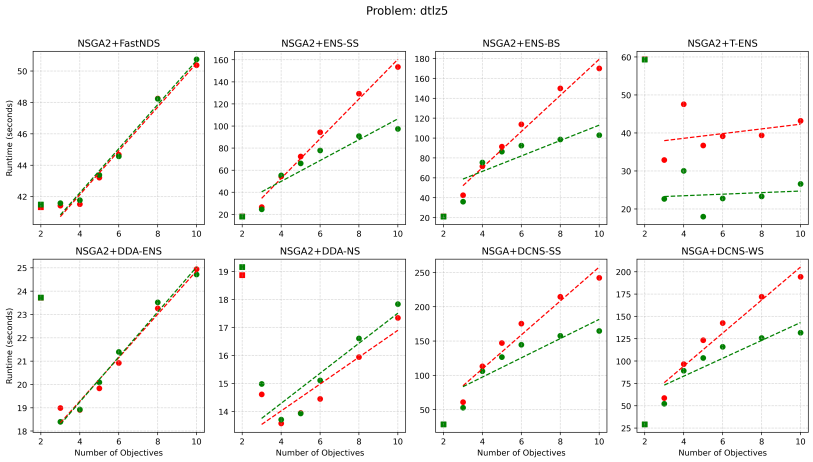


(c) DTLZ3

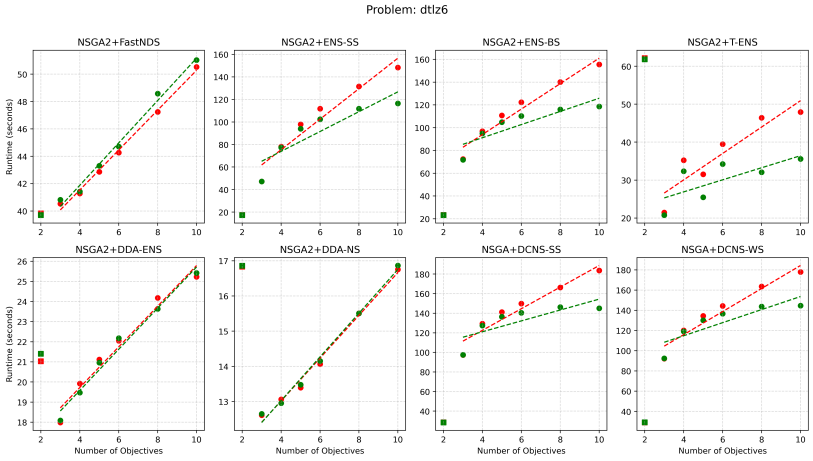
Figure 5.4: NSGA-II + SD-NDS results for [pop_size, $n\_function\_evals$] = [1024, 60K] (DTLZ 1-7, $n_{var} = 50$).



(d) DTLZ4

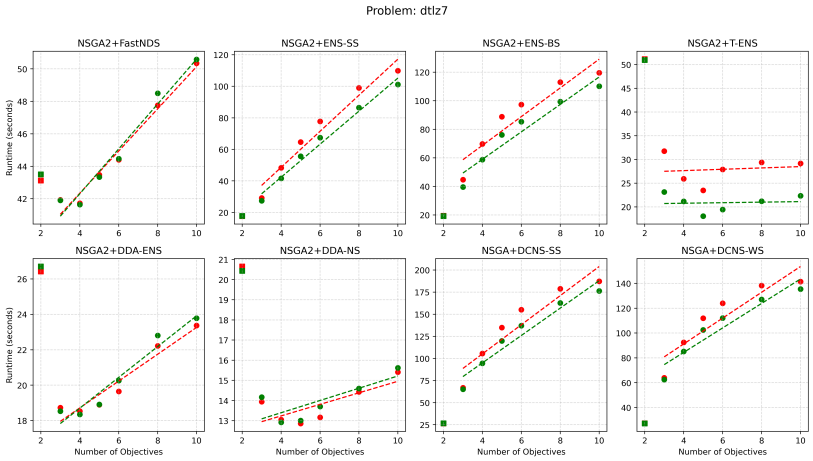


(e) DTLZ5



(f) DTLZ6

Figure 5.4: NSGA-II + SD-NDS results (continued).

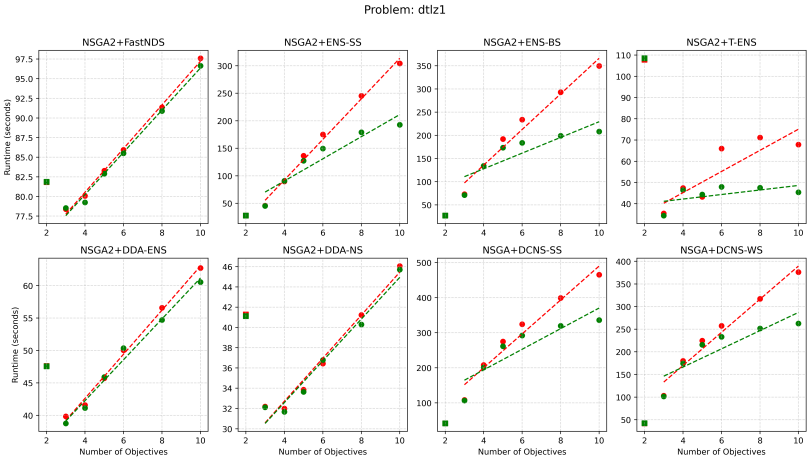


(g) DTLZ7

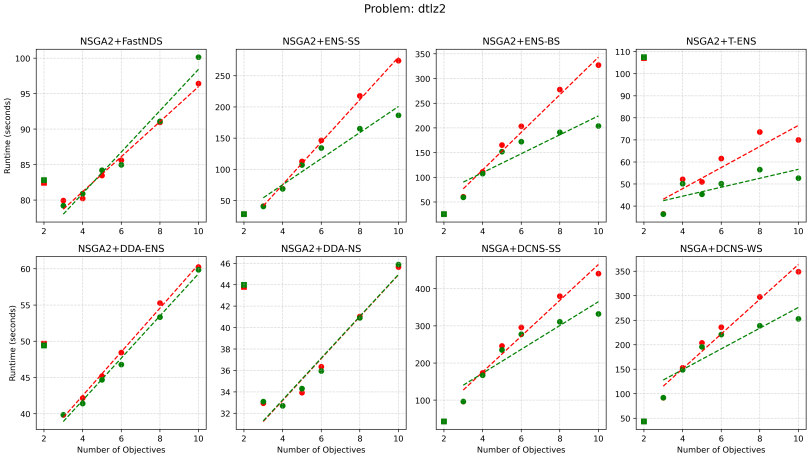
Figure 5.4: NSGA-II + SD-NDS results (continued).

Table 5.11: Overview of results obtained on RE Problem Set (Tanabe and Ishibuchi, 2020) using NSGA-II + SD-NDS

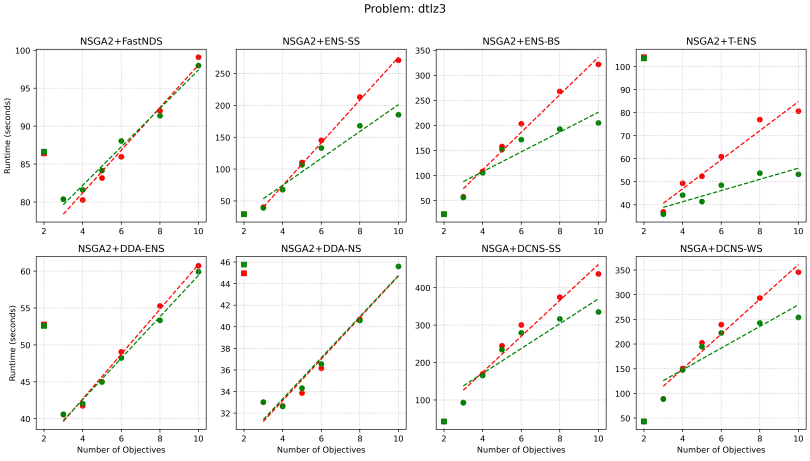
Problem	Number of Objectives	Average Time Gain (RE Problems)					
		ENS-SS	ENS-BS	FastNDS	DDA-NS	DDA-ENS	T-ENS
RE21	2	-1%	-1%	-2%	-3%	-4%	-1%
RE22	2	-1%	-1%	-2%	-2%	-3%	0%
RE23	2	0%	0%	-2%	-3%	-2%	0%
RE24	2	-1%	0%	-2%	-1%	-2%	0%
RE25	2	-1%	0%	-3%	-1%	-3%	0%
RE31	3	1%	-1%	-3%	-3%	-4%	3%
RE32	3	-4%	-4%	-2%	-4%	-5%	-1%
RE33	3	2%	1%	-1%	-4%	-2%	3%
RE34	3	14%	9%	-3%	-7%	2%	9%
RE35	3	0%	-1%	-1%	-2%	-4%	-1%
RE36	3	-2%	1%	-1%	-2%	-4%	7%
RE37	3	0%	-1%	-1%	-4%	-3%	-2%
RE41	4	0%	0%	-1%	-4%	-3%	-2%
RE42	4	2%	-2%	-2%	-2%	-4%	33%
RE61	6	5%	-2%	-3%	-2%	-2%	-64%
RE91	9	2%	1%	-3%	-3%	-2%	-42%



(a) DTLZ1

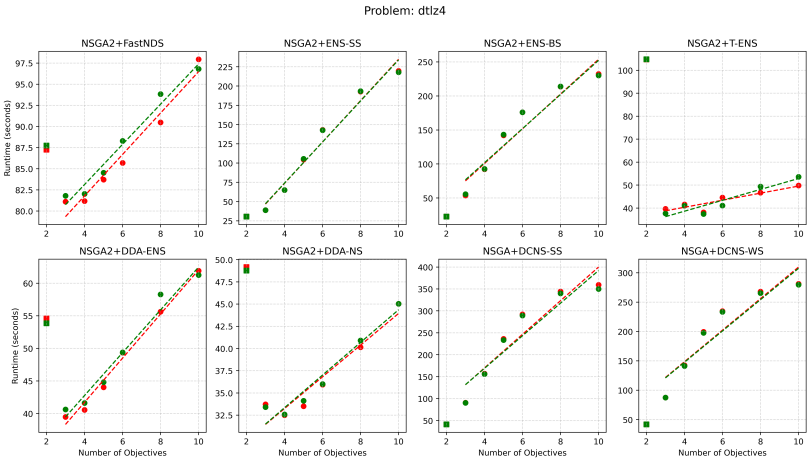


(b) DTLZ2

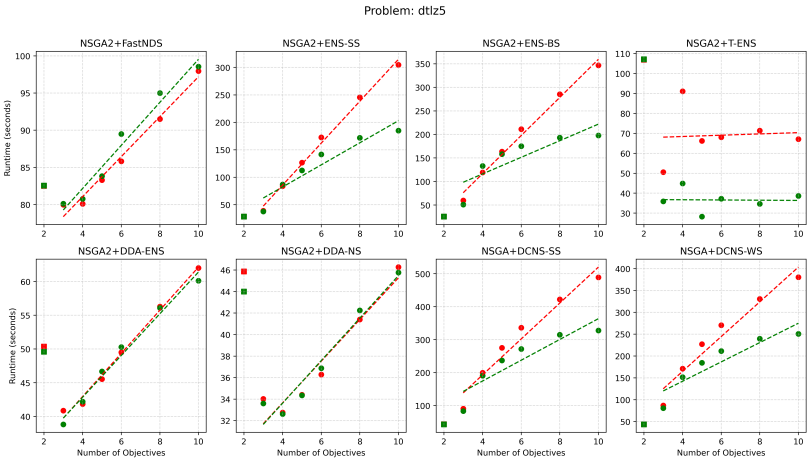


(c) DTLZ3

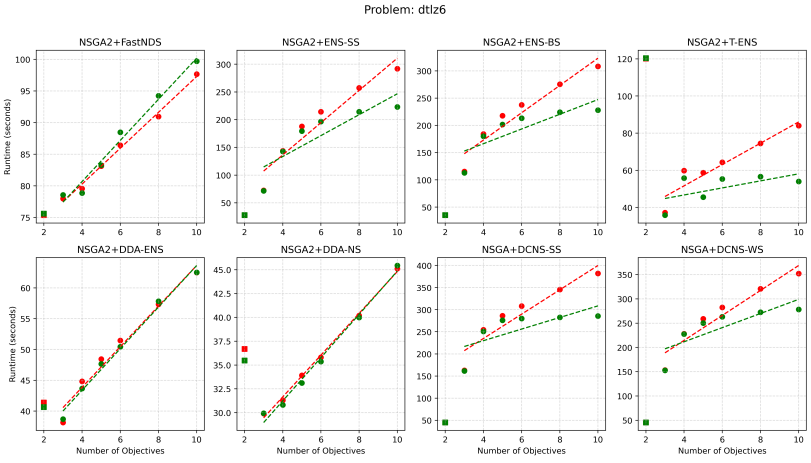
Figure 5.5: NSGA-II + SD-NDS results for [pop_size, $n\_function\_evals$] = [2048, 60K] (DTLZ 1-7, $n_{var} = 50$).



(d) DTLZ4

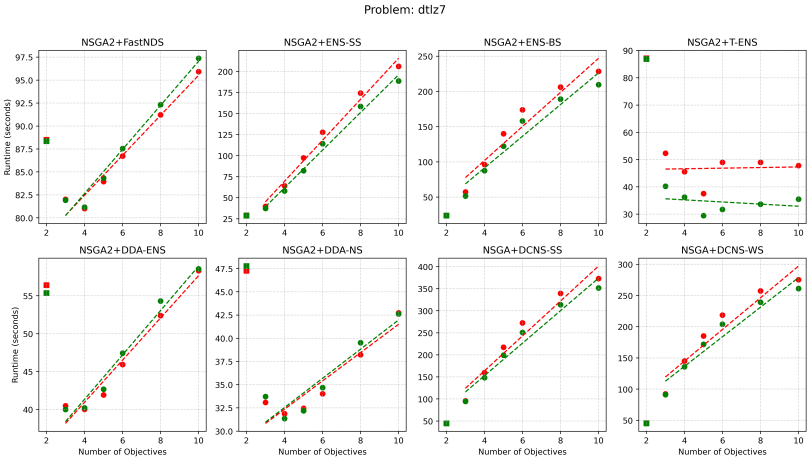


(e) DTLZ5



(f) DTLZ6

Figure 5.5: NSGA-II + SD-NDS results (continued).



(g) DTLZ7

Figure 5.5: NSGA-II + SD-NDS results (continued).

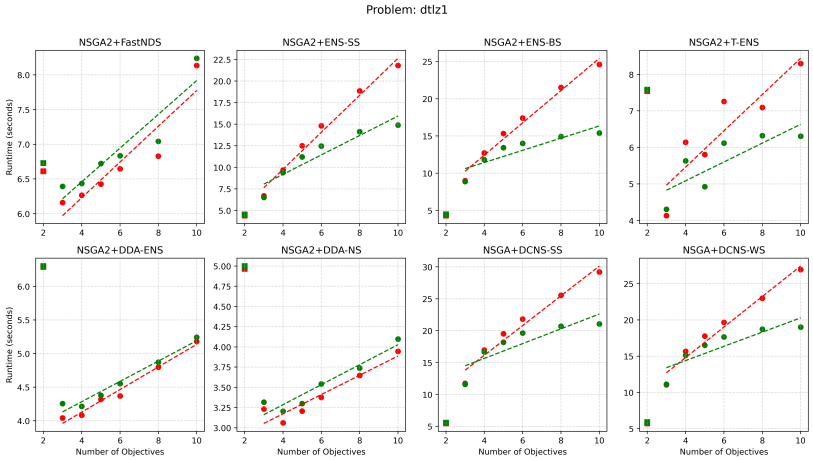
Parameter	**Values**
pop_size	[**256**, 512, 1024, 2048]
n_function_evals	[30K, **60K**, 120K]

Table 5.12: Experiment Parameter settings for experiments on Real World problems from (Tanabe and Ishibuchi, 2020)

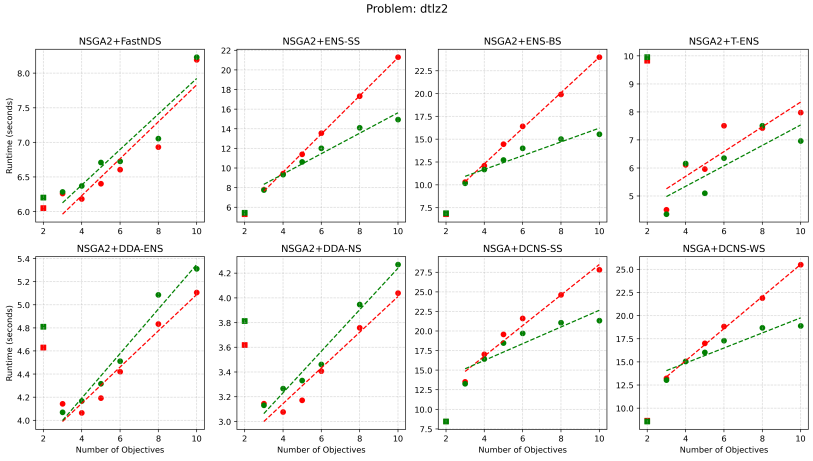
We now present the results on using our proposed SD-NDS algorithm in NSGA-II to solve real-world MOO problems (Tanabe and Ishibuchi, 2020). The experiments parameters for this set of experiments is detailed in Table 5.12. The base-case results for the RE problem set is presented in Figure 5.8.

As we can observe from Figure 5.8, we do not see significant benefits we observed in the DTLZ experiment set. This can be attributed to the fact that some of these test problems have low variance between the average correlation of their objectives, this makes it all the possible variates to be as good as one another and therefore applying variate ordering wont produce any benefit in run-time.

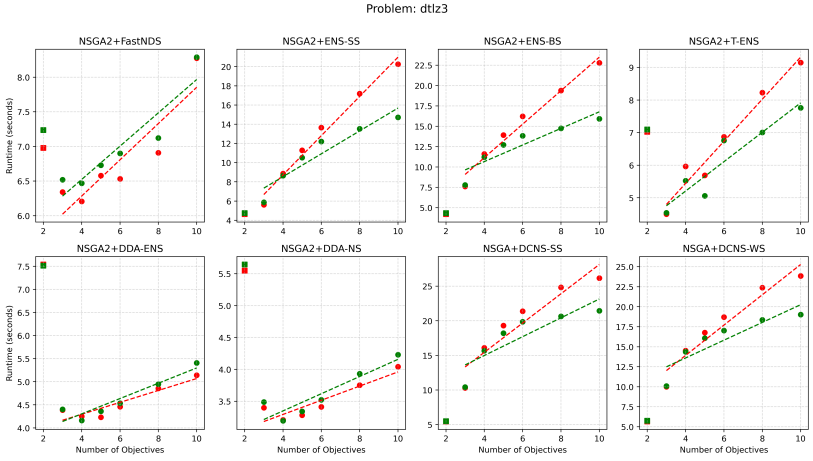
There are clearly some cases in which the problems in the RE set take advantage of our variate ordering. We can see that ENS-BS achieves a maximum time reduction of 9%, ENS-SS achieves a maximum time reduction of 14.3%, whereas, the tree based method,



(a) DTLZ1

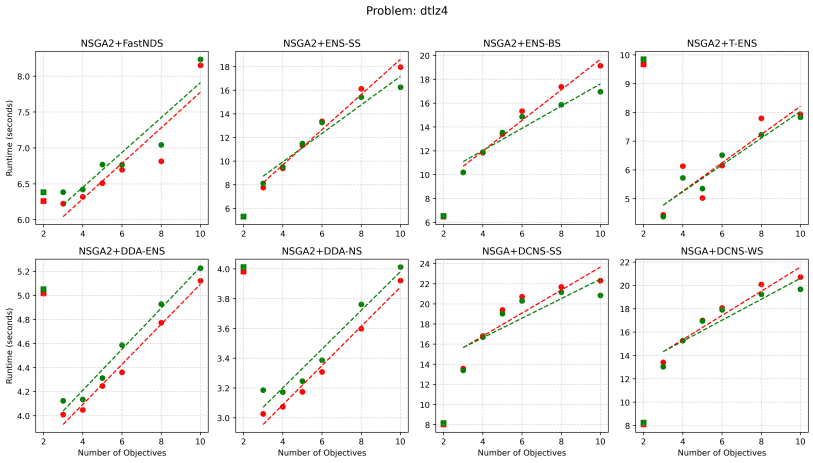


(b) DTLZ2

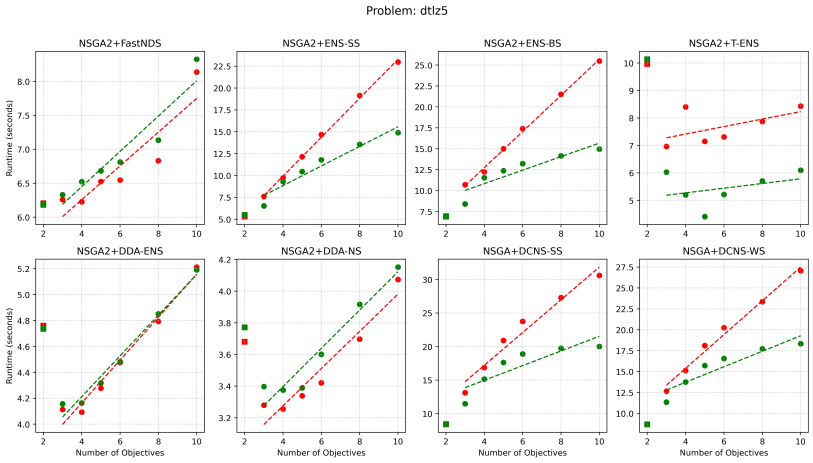


(c) DTLZ3

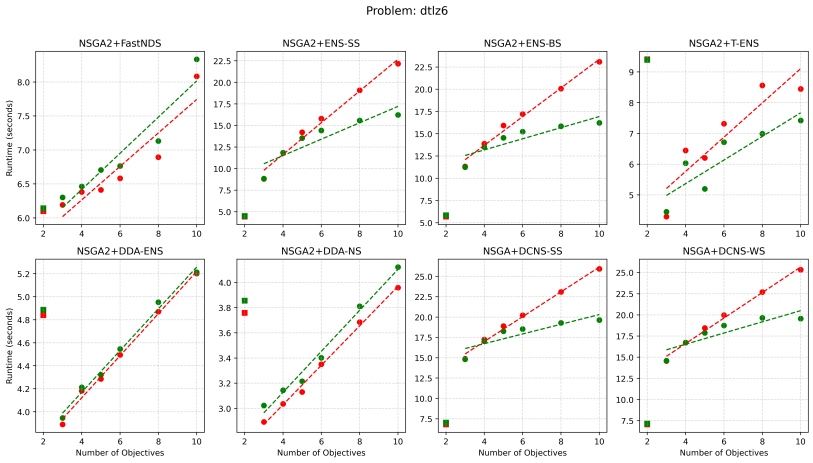
Figure 5.6: NSGA-II + SD-NDS results for [pop_size, $n\_function\_evals$] = [256, 30K] (DTLZ1–7, $n_{var} = 50$).



(d) DTLZ4

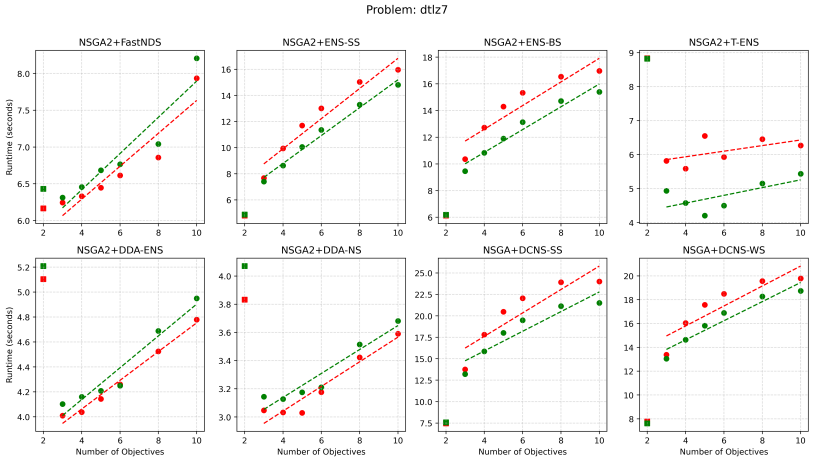


(e) DTLZ5



(f) DTLZ6

Figure 5.6: NSGA-II + SD-NDS results (continued).



(g) DTLZ7

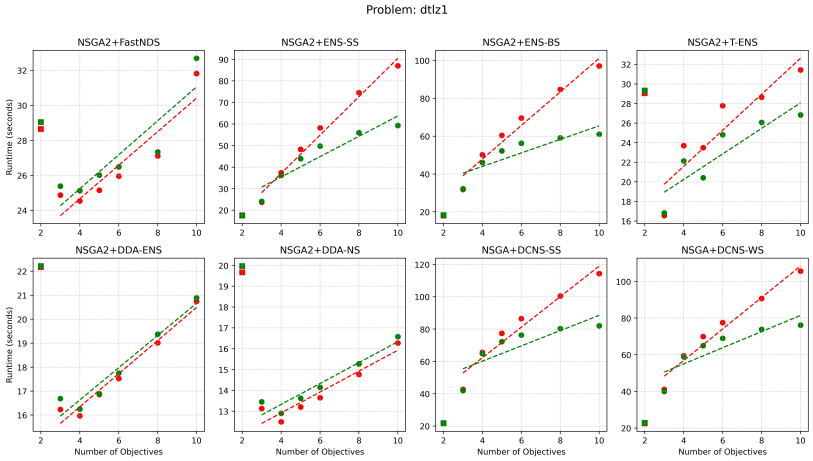
Figure 5.6: NSGA-II + SD-NDS results (continued).

T-ENS achieves a best time reduction of 33%.

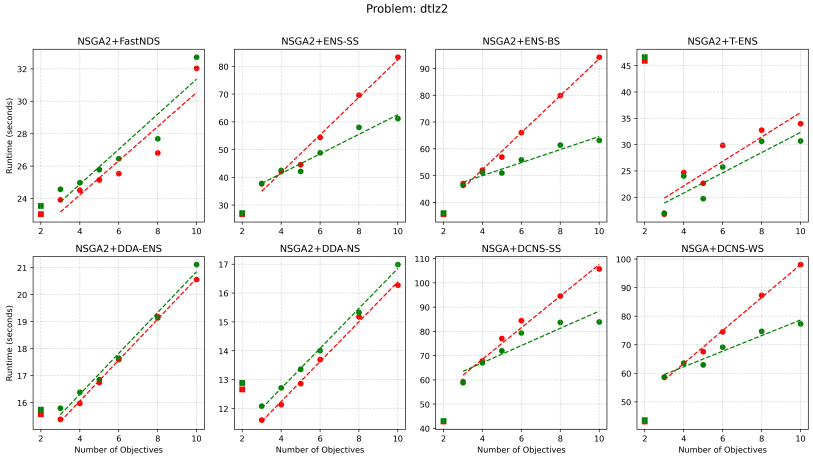
With the base case established, we now go over the impact of population size and the number of function evaluations on the performance of SD-NDS.

Looking at the results presented in Figures 5.9, 5.10 and 5.11, we can clearly see that as we increase the population size from 256 to 512 and 1024, the performance of our proposed SD-NDS method remains fairly consistent with the baseline, however, when we increase it further to 2048, there are some inconsistencies in the results of DDA-NS and DDA-ENS algorithms.

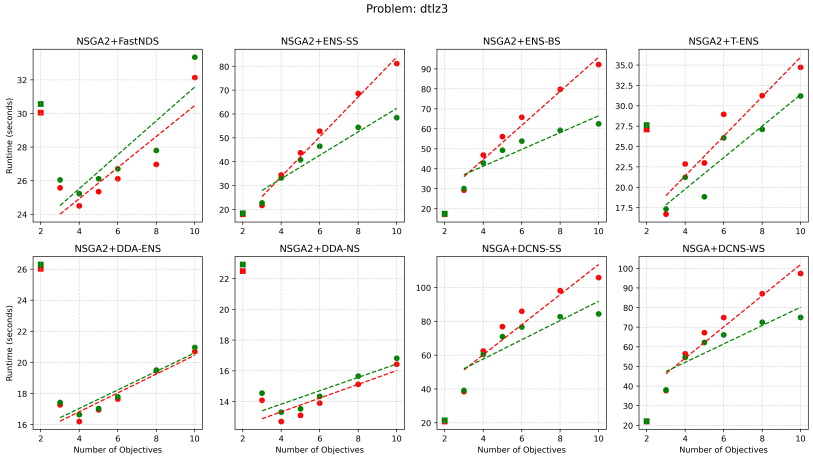
Looking at the results presented in Figures 5.12 and 5.13, as we decrease the number of function evaluations from 60K to 30K, the percentage reduction in run-time of NSGA-II remains fairly consistent, the same pattern is also seen when we increase the number of function evaluations from 60K to 120K.



(a) DTLZ1

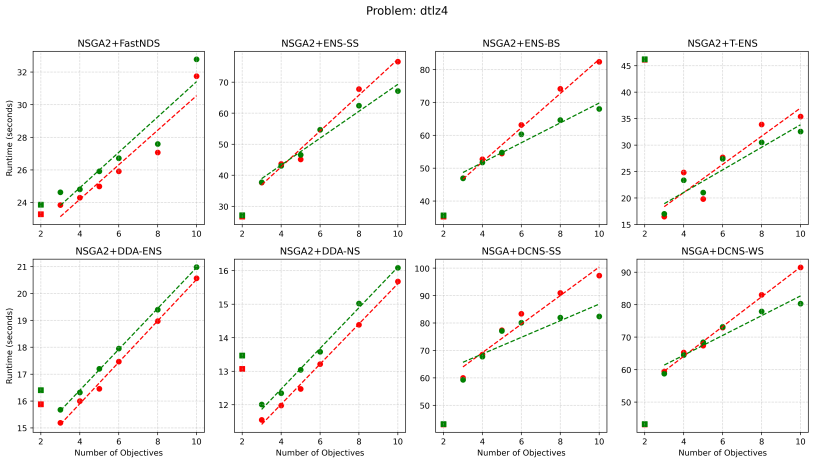


(b) DTLZ2

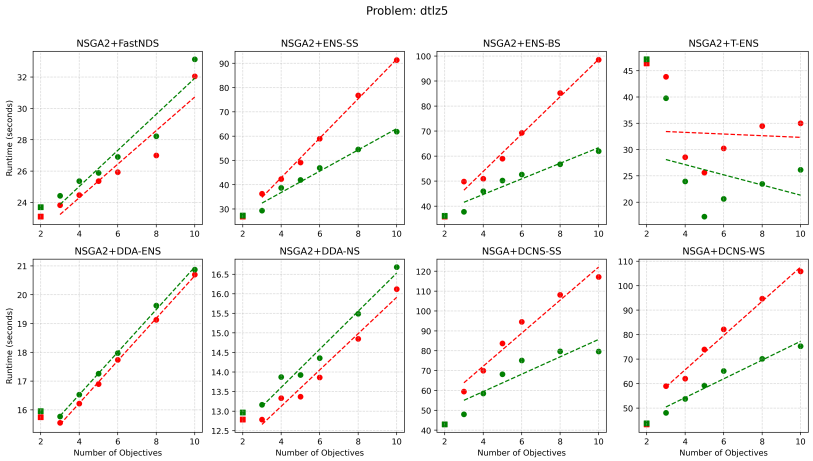


(c) DTLZ3

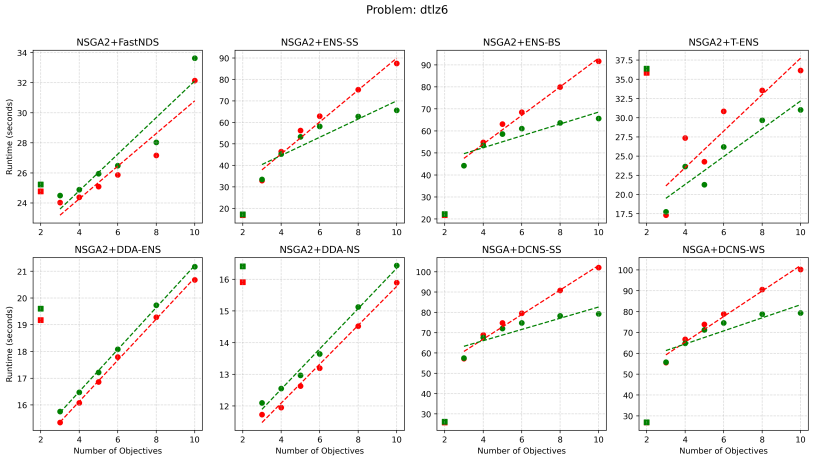
Figure 5.7: NSGA-II + SD-NDS results for [pop_size, $n\_function\_evals$] = [256, 120K] (DTLZ 1-7, $n_{var} = 50$).



(d) DTLZ4

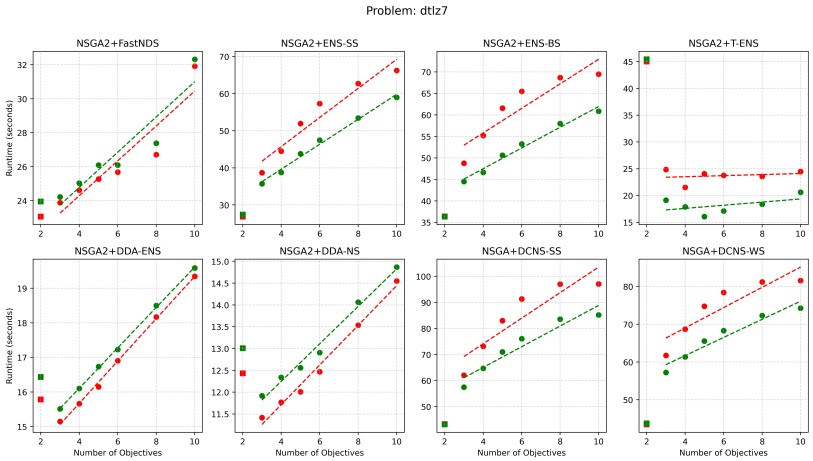


(e) DTLZ5



(f) DTLZ6

Figure 5.7: NSGA-II + SD-NDS results (continued).



(g) DTLZ7

Figure 5.7: NSGA-II + SD-NDS results (continued).

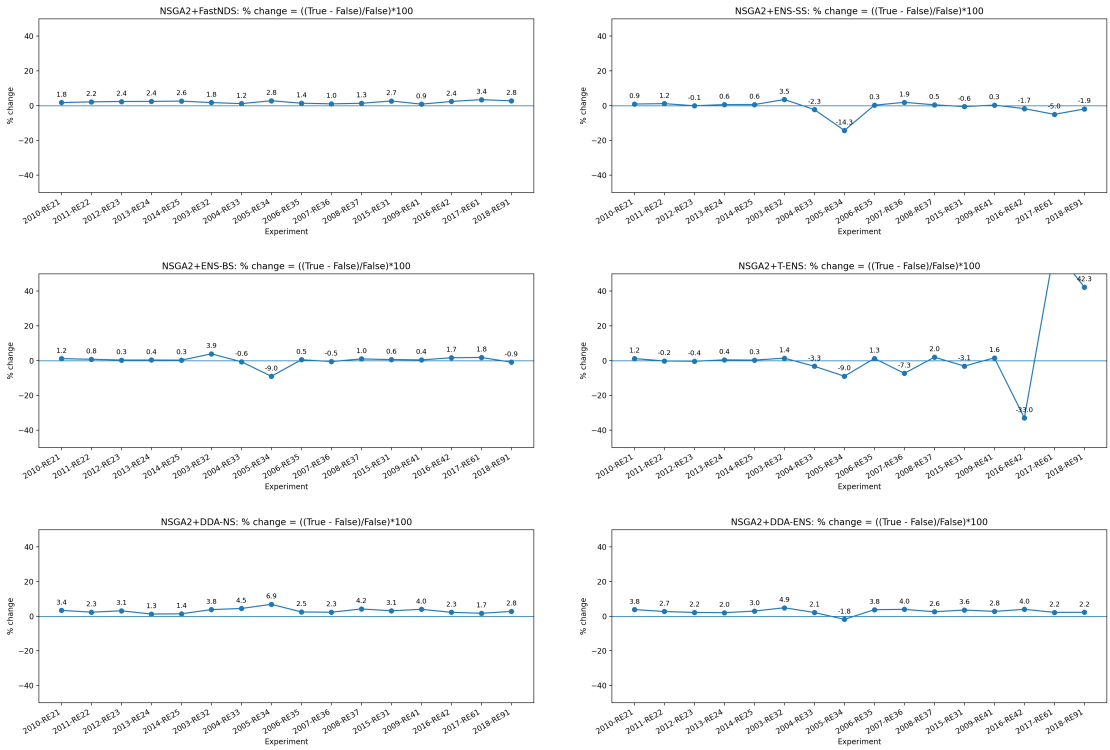


Figure 5.8: Comparison of NSGA-II experiment results on Real World Multi-Objective Optimization Problems using various NDS algorithms in conjunction with our SD-NDS optimization. Experiment Settings, [pop_size, n_func_evals] = [256, 60K]

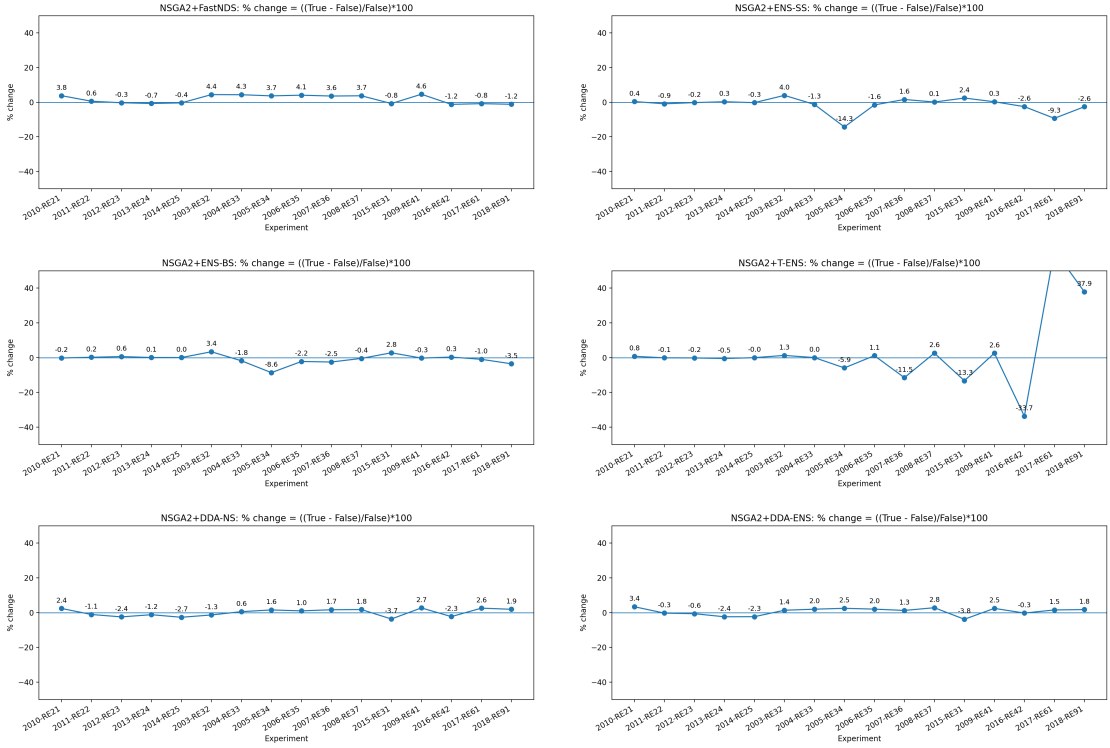


Figure 5.9: Comparison of NSGA-II experiment results on Real World Multi-Objective Optimization Problems using various NDS algorithms in conjunction with our SD-NDS optimization. Experiment Settings, [pop_size, n_func_evals] = [512, 60K]

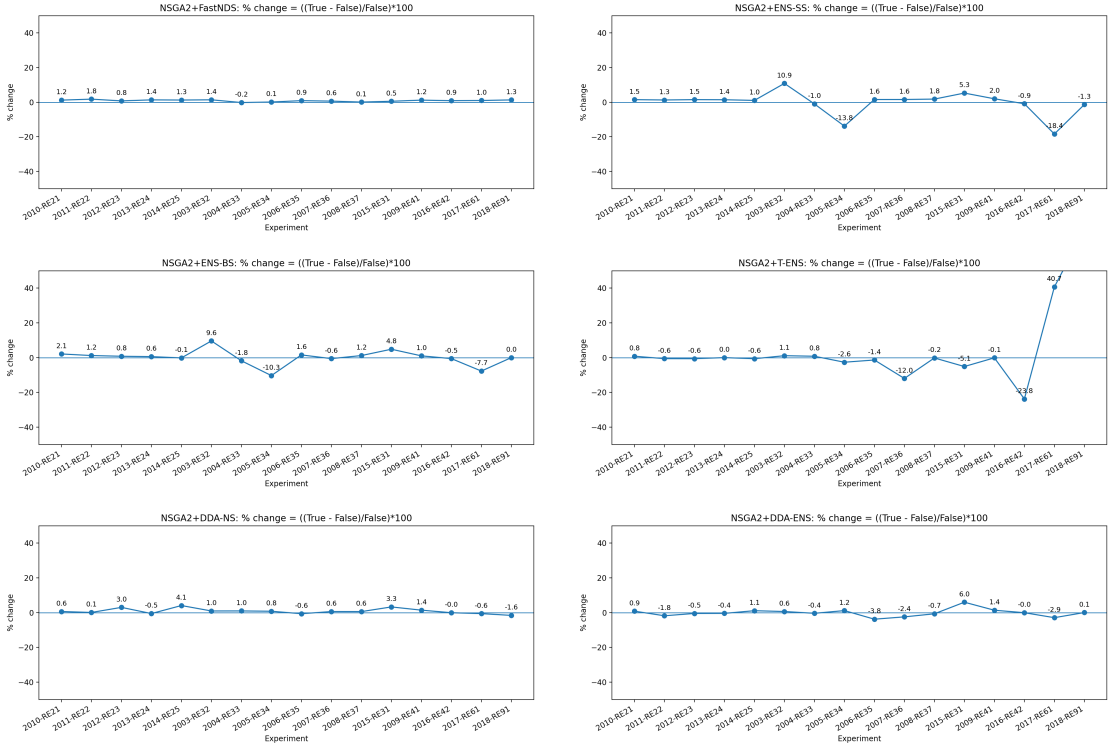


Figure 5.10: Comparison of NSGA-II experiment results on Real World Multi-Objective Optimization Problems using various NDS algorithms in conjunction with our SD-NDS optimization. Experiment Settings, [pop_size, n_func_evals] = [1024, 60K]

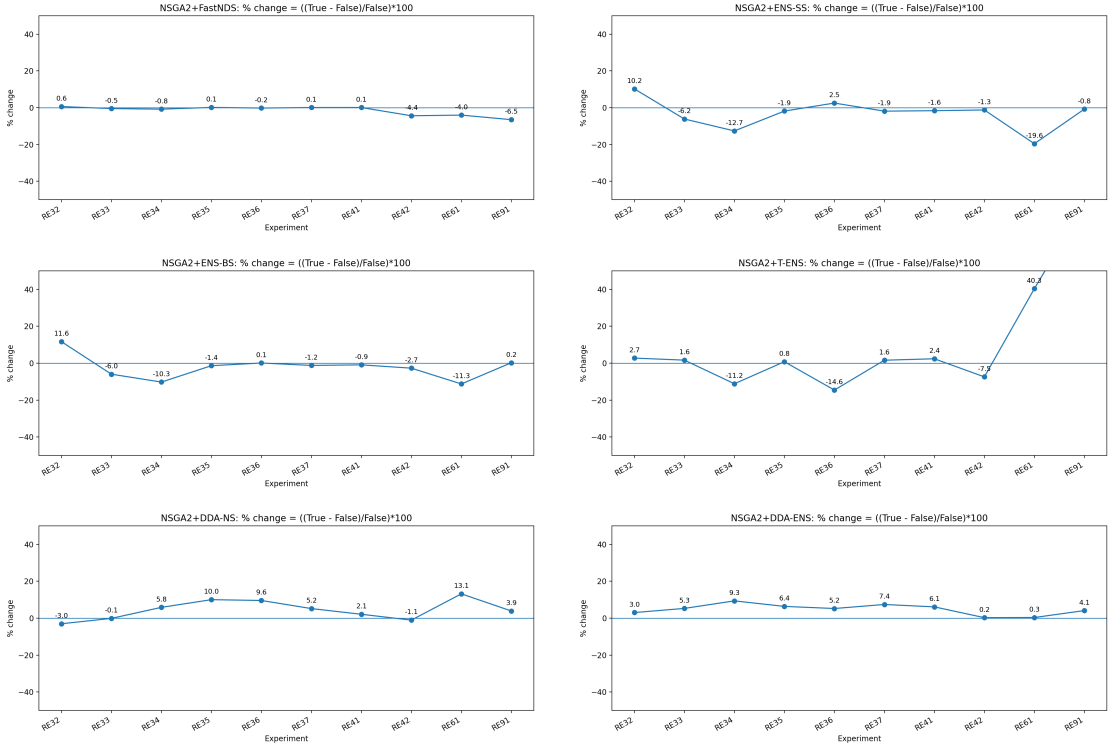


Figure 5.11: Comparison of NSGA-II experiment results on Real World Multi-Objective Optimization Problems using various NDS algorithms in conjunction with our SD-NDS optimization. Experiment Settings, [pop_size, n_func_evals] = [2048, 60K]

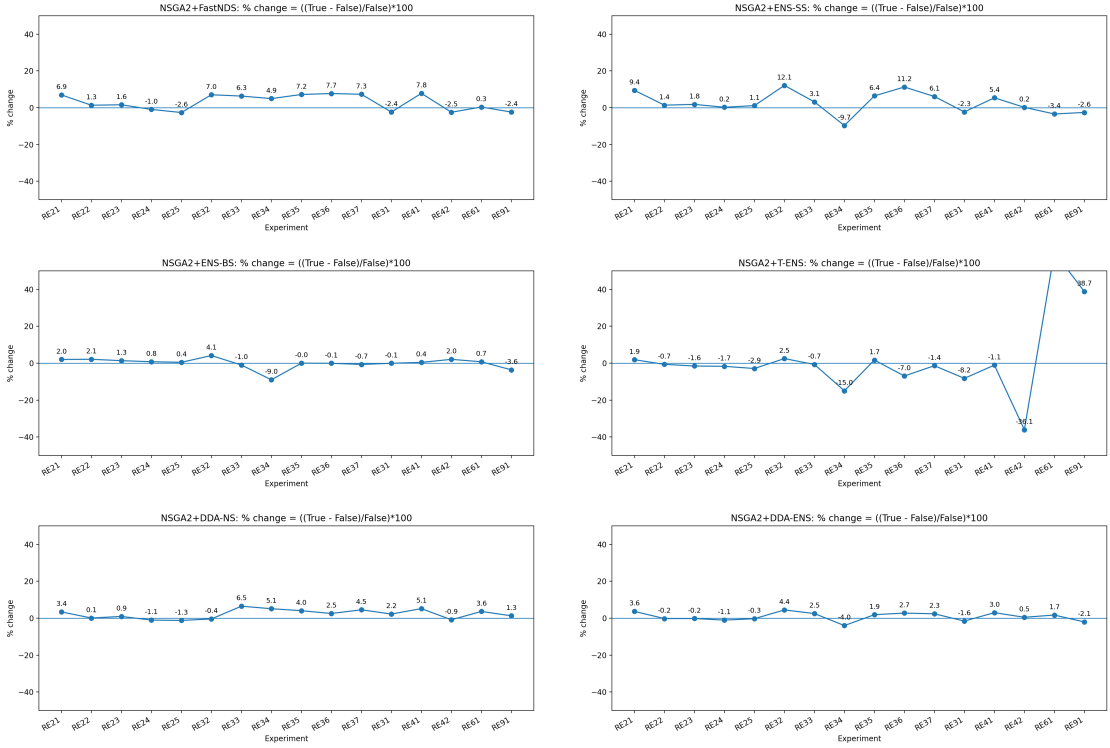


Figure 5.12: Comparison of NSGA-II experiment results on Real World Multi-Objective Optimization Problems using various NDS algorithms in conjunction with our SD-NDS optimization. Experiment Settings, [pop_size, n_func_evals] = [512, 30K]

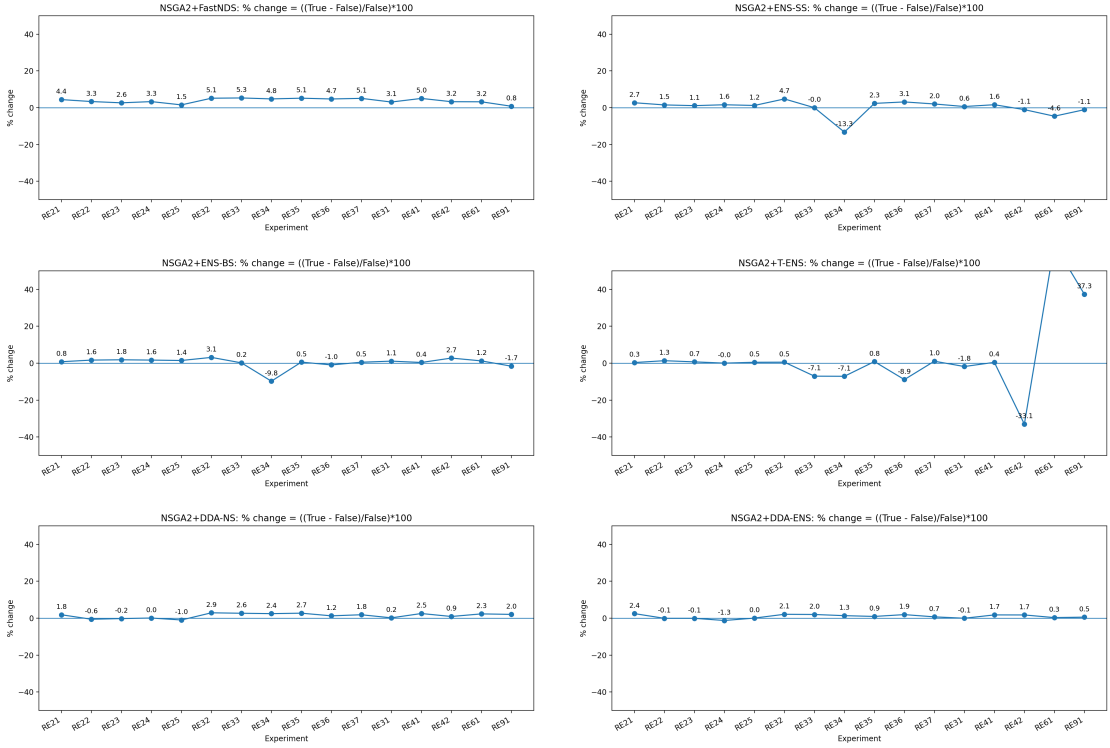


Figure 5.13: Comparison of NSGA-II experiment results on Real World Multi-Objective Optimization Problems using various NDS algorithms in conjunction with our SD-NDS optimization. Experiment Settings, [pop_size, n_func_evals] = [512, 120K]

Chapter 6

CONCLUSION

The research presented in this thesis proposed de-correlation based methods that improved the algorithmic time-complexity of NSGA-II by optimizing the NDS algorithm it uses. Based on our prior work in Causal Search for Skylines(CSS), we first presented the D-NDS approach which completely de-correlates the population before performing NDS on subsets of data. On experimental analysis, we found this method to be ineffective and counter-productive when performing NDS on synthetic datasets. We then presented our second method to optimize NDS, Selective De-Correlation NDS (SD-NDS) which *selectively de-correlates* the population before performing NDS. Selective de-correlation in this context is done through ordering objectives in the objective vector. To determine the best variate ordering, we introduced three heuristics:$minCorrRank$, $minAbsCorrRank$, $maxCorrRank$. Through experimental analysis on NDS performed on synthetic data, we show that $minCorrRank$ has the best performance. Following these observations, we run experiments to evaluate the effectiveness of our proposed SD-NDS approach on NSGA-II by using our most-effective heuristic, $minCorrRank$.

Upon analysis of the results obtained in our experiments integrating SD-NDS and NSGA-II, we observe a consistent improvement in run-time as we use SD-NDS. Although in some cases like in DTLZ-4 test-problem and some real-world problems, we do not see significant improvement in NSGA-II run-time. We hypothesize that this could be due to lower variance in the average correlatinos of the objective variables as given by Equation 5.2.

Limitations and Future Work

An interesting future research direction could be aimed at determining a ”cut-off” value for minCorr heuristic below which SD-NDS (variate ordering) won't be applied. Another line of work would be applying variate ordering to other pareto-dominance reliant algorithms and operators like the Skyline operator (Borzsony *et al.*, 2001).

REFERENCES

Ahmadianshalchi, A., S. Belakaria and J. R. Doppa, "Pareto front-diverse batch multi-objective bayesian optimization", in "Proceedings of the AAAI Conference on Artificial Intelligence", vol. 38, pp. 10784–10794 (2024).

Altinoz, T., "Solving many-objective constraint real-world optimization problems with multi-objective optimization algorithms", Avrupa Bilim ve Teknoloji Dergisi , 34, 234–238 (2022).

Blank, J. and K. Deb, "Pymoo: Multi-objective optimization in python", IEEE Access **8**, 89497–89509 (2020).

Borzsony, S., D. Kossmann and K. Stocker, "The skyline operator", in "Proceedings 17th International Conference on Data Engineering", pp. 421–430 (2001).

Burlacu, B., "Rank-based non-dominated sorting", URL `https://arxiv.org/abs/2203.13654` (2022).

Carroll, R. and G. Johnson, "Optimal design of compact spur gear sets", Journal of Mechanisms, Transmissions, and Automation in Design **106**, 1, 95–101 (1984).

Deb, A., "Introduction to soft computing techniques: artificial neural networks, fuzzy logic and genetic algorithms", in "Soft computing in textile engineering", pp. 3–24 (Elsevier, 2011).

Deb, K., "Multi-Objective Optimization", in "Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques", edited by E. K. Burke and G. Kendall, pp. 273–316 (Springer US, Boston, MA, 2005), URL `https://doi.org/10.1007/0-387-28356-0`$_1 0$.

Deb, K. 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 **18**, 4, 577–601 (2014).

Deb, K. and S. Jain, "Multi-speed gearbox design using multi-objective evolutionary algorithms", J. Mech. Des. **125**, 3, 609–619 (2003).

Deb, K., A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii", IEEE Transactions on Evolutionary Computation **6**, 2, 182–197 (2002).

Deb, K., K. Sindhya and J. Hakanen, "Multi-objective optimization", in "Decision sciences", pp. 161–200 (CRC Press, 2016).

Deb, K., L. Thiele, M. Laumanns and E. Zitzler, "Scalable test problems for evolutionary multiobjective optimization", in "Evolutionary multiobjective optimization: theoretical advances and applications", pp. 105–145 (Springer, 2005).

Gustavsson, P. and A. Syberfeldt, "A new algorithm using the non-dominated tree to improve non-dominated sorting", Evolutionary computation **26**, 1, 89–116 (2018).

Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, "Array programming with NumPy", Nature **585**, 7825, 357–362, URL https://doi.org/10.1038/s41586-020-2649-2 (2020).

Hashemi, A., M.-R. Pajoohan and M. B. Dowlatshahi, "Nsofs: a non-dominated sorting-based online feature selection algorithm", Neural Comput. Appl. **36**, 3, 1181–1197, URL https://doi-org.ezproxy1.lib.asu.edu/10.1007/s00521-023-09089-5 (2023).

Horn, J., N. Nafpliotis and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization", in "Proceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence", pp. 82–87 (Ieee, 1994).

Jensen, M., "Reducing the run-time complexity of multiobjective eas: The nsga-ii and other algorithms", IEEE Transactions on Evolutionary Computation **7**, 5, 503–515 (2003).

Keahey, K., J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha and J. Stubbs, "Lessons learned from the chameleon testbed", in "Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)", (USENIX Association, 2020).

Kohira, T., H. Kemmotsu, O. Akira and T. Tatsukawa, "Proposal of benchmark problem based on real-world car structure design optimization", in "Proceedings of the Genetic and Evolutionary Computation Conference Companion", pp. 183–184 (2018).

Mishra, S. and C. A. C. Coello, "P-ens: Parallelism in efficient non-dominated sorting", in "2018 IEEE Congress on Evolutionary Computation (CEC)", pp. 1–8 (2018).

Mishra, S., S. Saha and S. Mondal, "Divide and conquer based non-dominated sorting for parallel environment", in "2016 IEEE Congress on Evolutionary Computation (CEC)", pp. 4297–4304 (2016).

Mishra, S., S. Saha, S. Mondal and C. A. C. Coello, "A divide-and-conquer based efficient non-dominated sorting approach", Swarm and evolutionary computation **44**, 748–773 (2019).

Mishra, S. and R. Senwar, "Dda-ens: Dominance degree approach based efficient non-dominated sort", in "2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)", pp. 1075–1080 (IEEE, 2020).

Norvig, P. R. and S. A. Intelligence, "A modern approach", Prentice Hall Upper Saddle River, NJ, USA: Rani, M., Nayak, R., & Vyas, OP (2015). An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. Knowledge-Based Systems **90**, 33–48 (2002).

Rangaiah, G. P., *Multi-objective optimization: techniques and applications in chemical engineering*, vol. 5 (world scientific, 2016).

Srinivas, N. and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms", Evolutionary computation **2**, 3, 221–248 (1994).

Tanabe, R. and H. Ishibuchi, "An easy-to-use real-world multi-objective optimization problem suite", Applied Soft Computing **89**, 106078 (2020).

Vrajitoru, D., "Large population or many generations for genetic algorithms? implications in information retrieval", in "Soft computing in information retrieval: Techniques and applications", pp. 199–222 (Springer, 2000).

Zhang, X., Y. Tian, R. Cheng and Y. Jin, "An efficient approach to nondominated sorting for evolutionary multiobjective optimization", IEEE Transactions on Evolutionary Computation **19**, 2, 201–213 (2015).

Zhang, X., Y. Tian, R. Cheng and Y. Jin, "A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization", IEEE Transactions on Evolutionary Computation **22**, 1, 97–112 (2018).

Zhou, Y., Z. Chen and J. Zhang, "Ranking vectors by means of the dominance degree matrix", IEEE Transactions on Evolutionary Computation **21**, 1, 34–51 (2016).

Zitzler, E., M. Laumanns and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm", TIK report **103** (2001).

Zitzler, E. and L. Thiele, "An evolutionary algorithm for multiobjective optimization: The strength pareto approach", TIK report **43** (1998).