# Limiting Index Sort

*A New Non-Dominated Sorting Algorithm and its Comparison to the State-of-the-Art*

Michael Mazurek
*Canadian Forces Aerospace Warfare Centre OR Team Co-op Student*

Slawomir Wesolkowski, Ph.D.
*Canadian Forces Aerospace Warfare Centre OR Team*

DRDC CORA TM 2010-097
May 2010

**Defence R&D Canada**
**Centre for Operational Research and Analysis**

National Defence Défense nationale

Canada

# Limiting Index Sort

*A New Non-Dominated Sorting Algorithm and its Comparison to the State-of-the-Art*

Michael Mazurek
Canadian Forces Aerospace Warfare Centre OR Team Co-op Student

Slawomir Wesolkowski, Ph.D.
Canadian Forces Aerospace Warfare Centre OR Team

**Defence R&D Canada – CORA**

Technical Memorandum
DRDC CORA TM 2010-097
May 2010

Principal Author

*Original signed by Michael Mazurek* *Slawomir Wesolkowski*

Michael Mazurek Slawomir Wesolkowski

Defence Research Assistant Defence Scientist

Approved by

*Original signed by D.M. Bergeron, Ph.D.*

D.M. Bergeron, Ph.D.

Head Air Section

Approved for release by

*Original signed by Paul Comeau*

Paul Comeau, Chief Scientist

The information contained herein has been derived and determined through best practice and adherence to the highest levels of ethical, scientific and engineering investigative principles. The reported results, their interpretation, and any opinions expressed therein, remain those of the author and do not represent, or otherwise reflect, any official opinion or position of DND or the Government of Canada.

This research was sponsored by the Canadian Forces Aerospace Warfare Centre (CFAWC) as part of the multi-objective optimization work done for Project Yukon.

Defence R&D Canada – Centre for Operational Research and Analysis (CORA)

# Abstract

An important problem in the realm of evolutionary multi-objective optimization (MOO) is that of finding all non-dominated fronts (NDFs). We specifically address the computational efficiency of the non-dominated sorting algorithm for finding the non-dominated fronts for the non-dominated sorting genetic algorithm II (NSGA-II) algorithm. We introduce the Limiting Index Sort (LIS) algorithm which improves on the existing state-of-the-art algorithm for datasets which are positively correlated or uncorrelated. LIS indexes individuals based on their scores in each objective, and intelligently iterates through the index to limit the number of comparisons required between individuals. LIS also makes use of a stopping condition, allowing it to exclude some individuals from the skyline without comparing them to any other individuals. LIS was compared to two other state-of-the-art non-dominated sorting algorithms, the Sort and Limit Skyline Algorithm (SaLSa), and the Divide-and-Conquer (D&C) approach. LIS outperformed SaLSa in all tests, and it outperformed D&C when sorting individuals with positively correlated or uncorrelated objective scores, except for sorting large, uncorrelated, datasets on many objectives. LIS outperformed D&C for sorting small, negatively correlated datasets on three or five objectives. By understanding the appropriate non-dominated sorting algorithm to use in different situations, NSGA-II can be utilized more efficiently for MOO. This will allow optimizations to be run with larger population sizes, more objectives, or for more generations, which should ultimately increase the quality of solutions returned by the algorithm.



# Résumé

Un problème important dans le domaine d'optimisation multi-objective (MOO) évolutive est de trouver tous les fronts non-dominés (NDFs). Nous abordons spécifiquement l'efficacité de l'algorithme de tri non-dominé pour trouver les fronts non-dominés pour l'algorithme génétique de triage non-dominé (NSGA-II). Nous introduisons l'algorithme Limiting Index Sort (LIS) et démontrons sa supériorité pour le tri des données positivement corrélés ou décorrélés. LIS indexe les solutions individuelles sur la base de leurs scores pour chaque objectif, et itère intelligemment à travers les indices pour limiter le nombre de comparaisons nécessaires entre les solutions individuelles. LIS également fait usage d'une condition d'arrêt, ce qui permet d'exclure certaines solutions de la ligne d'horizon, sans les comparer à d'autres solutions. LIS a été comparée à deux autres algorithmes de tri non-dominé de pointe, le Sort and Limit Skyline Algorithm (SaLSa) et l'algorithme Divide-and-Conquer (D&C). LIS a surclassé SaLSa dans tous les tests, et il surpassé D&C dans le tri des individus pour des objectifs à corrélation positive ou étant décorrélé, sauf pour le tri d'un grand nombre d'individus avec un grand nombre d'objectifs decorrélées. LIS a surclassé D&C pour le tri d'un petit nombre d'individus sur trois ou cinq objectifs anti-corrélées. En comprenant les circonstances pendant lesquelles chacun des algorithmes pour le tri de vecteurs non-dominé est meilleur, NSGA-II peut être utilisé plus efficacement pour MOO. Cela permettra de faire des optimisations sur de plus grandes populations, sur plus objectifs, ou pour plus de générations, qui devrait, augmenter la qualité des solutions retournées par l'algorithme.

This page intentionally left blank.

# Executive Summary

## Limiting Index Sort: A New Non-Dominated Sorting Algorithm and its Comparison to the State-of-the-Art

**Michael Mazurek; Slawomir Wesolkowski; DRDC CORA TM 2010-097; Defence R&D Canada – CORA; May 2010.**

**Introduction or background:** Multi-objective optimization (MOO) is concerned with finding solutions that are optimal with respect to multiple and sometimes opposing objectives. An important problem in the realm of evolutionary multi-objective optimization is that of finding all non-dominated fronts (NDFs). We specifically address the computational efficiency of the non-dominated sorting algorithm for finding the non-dominated fronts for the non-dominated sorting genetic algorithm II (NSGA-II) algorithm.

We introduce the Limiting Index Sort (LIS) algorithm which improves on the existing state-of-the-art algorithm for datasets which are positively correlated or uncorrelated. LIS indexes individuals based on their scores in each objective, and intelligently iterates through the index to limit the number of comparisons required between individuals. LIS also makes use of a stopping condition, allowing it to exclude some individuals from the skyline without comparing them to any other individuals.

**Results:** LIS was compared to two other state-of-the-art non-dominated sorting algorithms, the Sort and Limit Skyline Algorithm (SaLSa), and the Divide-and-Conquer (D&C) approach. LIS outperformed SaLSa in all tests. LIS outperformed D&C on all tests when sorting individuals with positively correlated or uncorrelated objective scores, except for sorting uncorrelated datasets with more than 4000 individuals on nine objectives. D&C was the most efficient algorithm for sorting negatively correlated datasets, except when sorting datasets of up to about 4000 individuals on three or five objectives.

**Significance:** By understanding the appropriate non-dominated sorting algorithm to use in different situations, NSGA-II can be utilized more efficiently for MOO. This will allow optimizations to be run with larger population sizes, more objectives, or for more generations, which should ultimately increase the quality of solutions returned by the algorithm.

A variation on LIS, $LIS_{half}$, only sorts until the first few fronts, which contain at least half of the individuals in the dataset, are found. This is a faster variation which can be used with NSGA-II, because only the half of the combined parent and children population is chosen as parents for the next generation of the algorithm.

**Future plans:** In the future, a more precise assessment of the cases in which it is appropriate to use either LIS or D&C should be performed. In particular, if there are problems where it is desired to sort highly correlated datasets, LIS will be a very efficient choice of algorithm. D&C's performance worsens as the number of objectives increases, and also as the correlation between the objective scores increases. If the correlation between a large number of objectives in a dataset is high enough, there may be some cases where it would be more efficient to use LIS instead of

D&C, no matter what the population size of the dataset is. An empirical (or perhaps analytical) study to determine if these cases exist could be useful.

# Sommaire

## Limiting Index Sort: A New Non-Dominated Sorting Algorithm and its Comparison to the State-of-the-Art

**Michael Mazurek; Slawomir Wesolkowski; DRDC CORA TM 2010-097; R & D pour la défense Canada – CARO; Mai 2010.**

**Introduction ou contexte:** Le but de l'optimisation multi-objective (MOO) est la recherche de solutions optimales à l'égard de plusieurs objectifs qui sont parfois opposés. Un problème important dans le domaine d'optimisation multi-objective (MOO) évolutive est de trouver tous les fronts non-dominés (NDFs). Nous abordons spécifiquement l'efficacité de l'algorithme de tri non-dominé pour trouver les fronts non-dominés pour l'algorithme génétique de triage non-dominé (NSGA-II).

Nous introduisons l'algorithme Limiting Index Sort (LIS) et démontrons sa supériorité pour le tri des données positivement corrélées ou décorrélées. LIS indexe les solutions individuelles sur la base de leurs scores pour chaque objectif, et itère intelligemment à travers les indices pour limiter le nombre de comparaisons nécessaires entre les solutions individuelles. LIS également fait usage d'une condition d'arrêt, ce qui permet d'exclure certaines solutions de la ligne d'horizon, sans les comparer à d'autres solutions.

**Résultats:** LIS a été comparée à deux autres algorithmes de tri non-dominé de pointe, le Sort and Limit Skyline Algorithm (SaLSa) et l'algorithme Divide-and-Conquer (D&C). LIS a surclassé SaLSa dans tous les tests, et il surpassé D&C dans le tri des individus pour des objectifs à corrélation positive ou étant décorrélé, sauf pour le tri des données non corrélées avec plus de 4000 individus sur neuf objectifs. D&C est l'algorithme le plus efficace pour le tri de données négativement corrélées, sauf pour le tri de données sur 4000 ou moins d'individus pour trois ou cinq objectifs.

**Importance:** En comprenant les circonstances pendant lesquelles chacun des algorithmes pour le tri de vecteurs non-dominé est meilleur, NSGA-II peut être utilisé plus efficacement pour MOO. Cela permettra de faire des optimisations sur de plus grandes populations, sur plus d'objectifs, ou pour plus de générations, qui devrait, augmenter la qualité des solutions retournées par l'algorithme.

Une variation sur LIS, $LIS_{half}$, tri seulement les premiers quelques fronts qui comportent au moins la moitié des individus dans la base de données. Il s'agit d'une variation plus rapide qui peut être utilisé avec NSGA-II, parce que seule la moitié de l'ensemble des parents et des enfants de la population est choisi en tant que parents pour la prochaine génération de l'algorithme.

**Perspectives:** À l'avenir, une évaluation plus précise des cas dans lesquels il est approprié d'utiliser la LIS ou la D&C doit être effectuée. En particulier, s'il y a des problèmes où il est souhaitable de trier des données avec une forte corrélation, LIS sera le choix le plus efficace. La performance de D&C se dégrade au fur et à mesure que le nombre d'objectifs augmente, et que la corrélation entre les objectifs augmente. Si la corrélation entre un grand nombre d'objectifs dans un ensemble de données est suffisamment élevée, ca se peut qu'il serait plus efficace d'utiliser

LIS au lieu de D & C, quelle que soit la taille de la population. Une analyse empirique (ou peut-être analytique) pourrait être utile afin de déterminer si ces cas existent.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgements

The authors thank Irene Collin for improving the readability and presentation of this technical memorandum, as well as the reviewer, Bohdan Kaluzny, for providing helpful comments.

# 1 Introduction

Multi-objective optimization (MOO) is concerned with finding solutions that are optimal with respect to multiple and sometimes opposing objectives [1]. An important problem in the realm of evolutionary multi-objective optimization [1], [2] is that of finding all non-dominated fronts (NDFs). We specifically address the computational efficiency of the non-dominated sorting algorithm for finding the non-dominated fronts for the NSGA-II algorithm. A similar problem to performing a non-dominated sort is to find just the first non-dominated front, or *skyline* of a dataset [3]. This problem has many applications with database searching applications, and multiple approaches have been taken to solve this problem [3], [4], [5], [6]. One easy way to sort a dataset into NDFs is to find the set's skyline, save these points as the first NDF, remove these points from the dataset, and then find the reduced dataset's skyline, which is then the second NDF of the dataset. This process is repeated for subsequent fronts, until the reduced dataset is empty, and every point in the original dataset has been assigned to a front. The non-dominated sorting algorithm suggested for use with the non-dominated sorting genetic algorithm II (NSGA-II) [2] uses exactly this method, although with a suboptimal skyline algorithm. Three skyline algorithms, block-nested loops [3], sort-filter skyline [4], and the sort-and-limit skyline algorithm [5], [6], are presented and discussed below.

An important concept in MOO is that of solution dominance. In a set of solutions with any number of objectives, a solution $A$ is said to dominate solution $B$ (written $A \succ B$) if and only if $A$ performs equally as well or better than $B$ in all objectives, and $A$ performs strictly better than $B$ in at least one objective. Two solutions $A$ and $B$ are incomparable (written $A \sim B$) if neither one dominates the other (i.e. $A$ performs better than $B$ in some objectives, and $B$ performs better than $A$ in others). The non-dominated front of a dataset of individuals is the subset of individuals which are not dominated by any others in the dataset. If individuals $A$ and $B$ are evaluated over M objectives, and have objective scores $(a_1,\ldots,a_M)$ and $(b_1,\ldots,b_M)$, respectively, and if a maximum objective score is optimal, then the definition of dominance is as follows:

$$A \succ B \Leftrightarrow \left(\forall\, i \in (1,\ldots,M): a_i \geq b_i\right) \wedge \left(\exists\, i \in (1,\ldots,M): a_i > b_i\right) \tag{1}$$

A very well known evolutionary multi-objective algorithm, NSGA-II [2], uses the concept of non-dominated sorting of data into multiple fronts. We will focus our comparisons to NSGA-II, which is an elitist, evolutionary, and multi-objective algorithm [2]. It introduces the idea of choosing individuals for successive generations based on the NDF they belong too. The first non-dominated front of a population of individuals is simply the skyline of the group. The second front contains individuals which are only dominated by members of the first front, and, in general, the $k^{th}$ front of a dataset is the set of individuals which are only dominated by members of the first k-1 fronts. Improving the non-dominated sorting speed for NSGA-II is crucial for many applications which use a large population and a large number of fronts such as military fleet mix problems [7].

The complexity of the sorting algorithm in NSGA-II, is $O(M \cdot N^2)$[1] where N is the number of solutions (individuals) to be sorted. This complexity can be improved by using the Divide and Conquer (D&C) algorithm [8], [9] to obtain a complexity of $O(N \cdot \log^{M-1} N)$. We introduce the Limiting Index Sort (LIS) algorithm which is shown to be experimentally better than the D&C algorithm for datasets which are positively correlated or uncorrelated (D&C excels at sorting anti-correlated data). LIS is a skyline algorithm which indexes individuals based on their scores in each objective, and intelligently iterates through the index to limit the number of comparisons required between individuals. LIS also makes use of a stopping condition [5], [6], so it can exclude some individuals from the skyline without comparing them to any other individuals. While LIS has a worst case complexity of $O(M \cdot N^2)$, we show that it outperforms on average all other non-dominated sorting algorithms on positively correlated or uncorrelated datasets. We have also found LIS to be superior or comparable to D&C in a few special cases of sorting solutions from anti-correlated objectives. In this paper, we present the new algorithm and detailed experimental results which validate our findings.

The paper is organized as follows. The second section summarizes the prior art. The third section introduces and describes the Limiting Index Sort algorithm. The results are described in the fourth section. The technical memorandum concludes with the fifth section. Annex A presents figures for sorting results on 3, 7, and 9 objectives. Annex B and C contain respectively the mean and standard deviation numbers of the required number of comparisons for all tests.

[1] Big *O* notation allows us to characterize the execution parameters of an algorithm in theory (in this case the time needed but it can also refer to memory requirements or specific numbers of additions or multiplications, etc.), given the problem size. Therefore, in the case of $O(M \cdot N^2)$ the complexity depends on the number of objectives M and the square of the number of solutions N.

# 2 Previous Work

A similar problem to performing a non-dominated sort is to find just the first non-dominated front, or *skyline* of a dataset [3]. This problem has many applications with database searching applications, and multiple approaches have been taken to solve this problem [3], [4], [5], [6]. While an optimal non-dominated sorting algorithm exists for sorting a dataset on two objectives [9], finding an optimal algorithm for sorting on three or more objectives remains an open problem.

One easy way to sort a dataset into NDFs is to find the set's skyline, save these points as the first NDF, remove these points from the dataset, and then find the reduced dataset's skyline, which is then the second NDF of the dataset. This process is repeated for subsequent fronts, until the reduced dataset is empty, and every point in the original dataset has been assigned to a front. The non-dominated sorting algorithm suggested for use with NSGA-II [2] uses exactly this method, although with a suboptimal skyline algorithm. Three skyline algorithms, *block-nested loops* [1], *sort-filter skyline* [4], and the *sort-and-limit skyline algorithm* [5], [6], are presented and discussed in section 2.2.

## 2.1 Two-objective Case

An optimal non-dominated sorting algorithm which sorts a set of $N$ individuals on $M$=2 objectives is described by Jensen [9]. As a first step, the individuals are sorted by their first objective scores, in decreasing order, and to break any ties, the second objective is considered. This results in an order in which no individual can be dominated by another one that comes after it in the order. As a result, the first individual in this list (i.e. the individual with the highest first objective score) is the first member of the first non-dominated front. The second individual is compared to the first; if the two points are incomparable, the second point is added to the first front, and otherwise it becomes the first member of the second front. Subsequent individuals are compared to all fronts, and assigned to the first front they are not dominated by. For example, if an individual $p$ is dominated by a member of the third front, but not the fourth, $p$ will be assigned to the fourth front.

Since the set is only being sorted on two objectives, the individual with the highest first-objective score in any front must also have the lowest second-objective score among all members of the front [9]. Because of this, and the fact that each new individual being compared to the fronts has a higher first-objective score than every individual which has already been assigned a front, new individuals only need to be compared to the newest member of a front (and only in the second objective) to determine whether or not they are dominated by the front. By implementing a binary search to find the front a new individual belongs to, as opposed to comparing to each front in order, this algorithm requires no more than $O(N \cdot \log N)$ comparisons to sort the entire dataset [9] and thus it should be used in the two-objective sorting case (no better algorithm is known).

## 2.2 Skyline Algorithms

### 2.2.1 Block-Nested Loops

A simple way to find the skyline is to use the *block-nested loops* (BNL) algorithm [3], which is the algorithm suggested for use with NSGA-II [3]. In a dataset with $N$-tuples[2], or individuals, a tuple is chosen and assigned as the only member of a *non-dominated set* (NDS). Each remaining tuple is then compared with each of the tuples in the NDS, which is updated after every comparison. Any NDS member which is dominated by a tuple is removed from the set, while tuples not dominated by any NDS member are added to the NDS, and tuples which are dominated by an NDS member are discarded. After every individual has been compared with the NDS, the NDS is the dataset's skyline.

In the best case for BNL, the skyline will consist of a single point, and it will be the first point to be added to the NDS. Every subsequent tuple will be compared to the single NDS member (and subsequently discarded), for a total of $N$ comparisons. If the tuples are being compared on $M$ objectives, each dominance test requires $M$ objective-value comparisons, for a total best-case complexity of $O(MN)$. In the worst case, when every dataset member is also a skyline member, $N(N-1)/2$ dominance tests are required, for a worst-case computational complexity of $O(MN^2)$.

### 2.2.2 Sort Filter Skyline

The *sort filter skyline* (SFS) algorithm [4] is a variation on BNL that first introduces the idea of initially ordering the individuals by a monotonically increasing scoring function, then considering them for skyline inclusion in this order. The result is an order in which no tuple can dominate any other one which comes before it in the order, because the order *"is a topological sort with respect to the skyline dominance partial relation"* [4].

Using this information, the first individual in the order must be part of the skyline, since it cannot be dominated by any other individual. The second individual only has to be compared to the first to determine its inclusion in the skyline; if the first does not dominate the second, then the second is a skyline member, otherwise, it is not. The remaining individuals only need to be compared with any skyline members that occur before them in the order. Thus, an individual is added to the skyline if and only if it is not dominated by these skyline members.

When using BNL, it is possible for two non-skyline tuples to be compared with each other. However, since both of those tuples will eventually be discarded, the comparison between them does not offer any useful information, and therefore is a wasted computation. By first sorting the dataset by an appropriate scoring function, SFS guarantees that every comparison between tuples involves at least one skyline member. This eliminates the wasted comparisons which may occur with BNL.

SFS has the same best- and worst-case scenarios as BNL, however there is an overhead of $O(N \cdot \log N)$ required to initially sort the dataset. So the computational complexities of SFS in the best and worst case are $O(N \cdot \log N + MN)$ and $O(N \cdot \log N + MN^2)$, respectively. Although SFS

[2] In mathematics, a tuple is a sequence (or ordered list) of finite length. An $N$-tuple is a sequence of $N$ elements.

requires this $O(N \cdot \log N)$ overhead, making it appear to have worse performance than BNL, empirical studies show that SFS significantly outperforms BNL [4]. This is because the time saved by eliminating the wasted comparisons which occur in BNL outweighs the cost of the overhead in SFS.

### 2.2.3 Sort and Limit Skyline algorithm

The latest skyline algorithm to be developed is the *sort and limit skyline algorithm* (SaLSa) [5], [6]. SaLSa works by first sorting the individuals by a monotonic scoring function (as in the case of SFS), then considering them for inclusion in the skyline in this order. The main improvement over SFS is that a limiting condition is defined, where the algorithm can be terminated without necessarily considering every individual for inclusion in the skyline. If the limiting condition is chosen appropriately, then a large number of individuals could potentially be excluded from the non-dominated front without having to perform any dominance tests between these individuals and the skyline members. By considering a limiting condition, SaLSa will never need to perform more comparisons than SFS if they are both applied to sort the same dataset.

Bartolini *et al.* propose three symmetric monotonic scoring functions (and three corresponding limiting conditions) to be used to pre-sort the individuals [5]. The individuals are each assigned a score based on the scoring function chosen, and then they are ordered by these scores, from highest to lowest. The individuals are considered for inclusion in the skyline in this order, since for these three functions, no individual may dominate any individual that comes before it in these orders.

The three scoring functions score the individuals by the sum of their objective scores, the product of their objective scores, or by the maximum of their objective scores. For brevity, only the function which scores individuals by their maximum objective score, $F_{max}$, will be discussed in detail in this paper:

$$F_{\max}(A_i) = \max_{j \in (1,\ldots,M)} a_{ij}. \tag{2}$$

The reader is referred to Bartolini *et al.* [5] for a description of the other two functions.

$F_{max}$ scores each individual by its maximum objective value. If the tuples are ordered in this way, no tuple can dominate any which comes before it in the order; as a consequence, tuples only need to be compared to those which come before them in the order to determine non-dominance.

The other aspect of SaLSa is that it maintains a *stopping value*, $v_{stop}$, when searching for the skyline. If the algorithm reaches a point where it can be proven that every unread individual is dominated by at least one skyline member, then SaLSa can stop without explicitly considering every individual for inclusion in the skyline. This limiting feature of SaLSa can lead to a major decrease in the number of comparisons needed to find the skyline. The stopping value is simply the maximum value of the skyline members' minimum objective scores. It is easy to update the stopping value; each time a skyline member is added to the front, its minimum objective score is compared to $v_{stop}$, and the maximum of the two numbers becomes the new $v_{stop}$. This is an important concept that is also used in our limiting index sort algorithm.

The SaLSa algorithm is summarized as follows. Let $M = F_{max}(A) = d_{max}$, where $M$ is *A's* maximum objective score. While iterating through the tuples (to consider them for inclusion in the skyline), if at any point the maximum objective score, $M_i$, of the *i-th* individual, $X_i$, is less than $v_{stop}$, then the stopping individual, $X_{stop}$, (the tuple which corresponds to $v_{stop}$), must dominate $X_i$, and every other tuple which comes after $X_i$ in the order. Since $M_i < v_{stop}$, the minimum objective of $X_{stop}$ must be greater than the maximum possible objective of $X_i$, therefore, every objective of $X_{stop}$ is greater than every objective of $X_i$, and $X_{stop}$ dominates $X_i$. If $X_{stop} \succ X_{stop+1}$, then $X_{stop}$ must dominate every tuple in the dataset that has not yet been considered for inclusion in the skyline; when this condition is satisfied, SaLSa has found the entire skyline, and it can stop.

In order to illustrate how SaLSa functions, consider the example dataset of five tuples $p_1,\ldots p_5$ in Table 1. The tuples each have scores in objective functions $f_1$, $f_2$, and $f_3$. The tuples marked with an asterisk (*), are part of the skyline.

Table 2 displays how the tuples in Table 1 would be sorted by using SaLSa with the max limiting function. First, the tuples are sorted by their maximum objective scores. Point $p_5$ is added to the skyline, as it is the first individual in the list. Point $p_2$ is then compared with $p_5$, and added to the skyline, for a total of one dominance test at this point. After two more comparisons, it is known that $p_3$ is incomparable with $p_5$, but dominated by $p_2$, so $p_3$ is not a skyline member. $p_4$ is incomparable with both $p_5$ and $p_2$, so it is added to the skyline. Since $p_4$'s minimum objective function score is 0.5, and $p_1$'s maximum objective score is 0.3, $p_4$ must dominate every remaining tuple in the sorted list, so the algorithm can stop, and no dominance tests involving $p_1$ are performed. SaLSa has effectively found the skyline with a total of five dominance tests.

*Table 1: Example Set of Tuples*

| *Tuple* | $f_1$ | $f_2$ | $f_3$ |
|---|---|---|---|
| $p_1$ | *0.1* | *0.2* | *0.3* |
| $p_2$* | *0.8* | *0.4* | *0.7* |
| $p_3$ | *0.5* | *0.2* | *0.6* |
| $p_4$* | *0.5* | *0.5* | *0.5* |
| $p_5$* | *0.4* | *0.3* | *0.9* |

*Table 2: Sorting with SaLSa (max)*

| *Tuple* | $f_1$ | $f_2$ | $f_3$ |
|---|---|---|---|
| $p_5$* | *0.4* | *0.3* | *0.9* |
| $p_2$* | *0.8* | *0.4* | *0.7* |
| $p_3$ | *0.5* | *0.2* | *0.6* |
| $p_4$* | *0.5* | *0.5* | *0.5* |
| $p_1$ | *0.1* | *0.2* | *0.3* |

By utilizing a monotonic scoring function to pre-sort the individuals, as well as a stopping value, SaLSa can efficiently find a skyline from a dataset, without necessarily performing dominance tests on every individual in the set. SaLSa is similar to SFS, the main difference between the two algorithms being SaLSa's consideration of a stopping condition. Hence, the two algorithms have the same computational complexity of $O(N \cdot \log N + MN^2)$ in the worst-case scenario. In the best case, the skyline will only contain one point, and the algorithm will stop after this point is found. Because of the overhead of the initial sort, SaLSa's best-case computational complexity is

$O(N \cdot \log N + MN)$. Because of the stopping condition, the number of dominance tests SaLSa requires to find a skyline must always be less than or equal to the number performed by SFS, so it will always outperform SFS [5].

## 2.3 Divide and Conquer Algorithms

The three previously discussed algorithms, BNL, SFS, and SaLSa are fairly similar to each other, and all have a worst-case computational complexity of $O(MN^2)$. An improved worst-case complexity of $O(N \cdot \log^{M-2} N)$ can be achieved by using a Skyline Divide and Conquer (SD&C) method developed by Kung, Luccio, and Preparata [8]. For brevity, their algorithm is not discussed in detail in this paper. However, an algorithm presented by Jensen [9] based on the SD&C is discussed below.

### 2.3.1 Divide & Conquer Non-Dominated Sorting Algorithm

A divide-and-conquer non-dominated sorting algorithm (D&C) is described by Jensen [9]. This algorithm is based on SD&C; however, it assigns every individual to the proper front after one pass, instead of just finding the skyline. The basic idea of D&C is as follows: a dataset of individuals, $S$, to be sorted on $M$ objectives is divided into sets $L$ and $H$, such that every individual in $L$ has a lower $M$-th objective score than all individuals in $H$. Initially, every individual in $S$ is assigned to the first non-dominated front. The set $H$ is then sorted recursively, so that every member of $H$ is assigned the correct front number. The members of $L$ are then compared to $H$, and the front members of $L$ are updated based on which members of $H$ they are dominated by. Finally, after $L$ has been sorted according to the front numbers in $H$, the members of $L$ need to be compared to each other, and this is done by recursively calling the D&C algorithm on $L$, while respecting the front numbers of $L$ as they have been affected by $H$.

D&C calls two functions recursively, nd-helperA (ndhA) and nd-helperB (ndhB), which will be detailed below. ndhA recursively divides the set of tuples into much smaller sets, and sorts these smaller sets into non-dominated fronts. ndhB is then used to "merge" these sets back together, until the entire dataset has been merged, and every individual has been assigned to the correct NDF.

Initially, ndhA is called on $S$, over the first $M$ objectives. If $S$ only contains 2 individuals, (i.e. if $|S| = 2$), then they can just be directly compared, and if one dominates the other, their front numbers can be updated. For example, let $S$ contain the individuals $s_1$ and $s_2$, with front numbers $f[s_1]$ and $f[s_2]$. Say $s_1$ dominates $s_2$, then $f[s_2]$ will be set to $\max(f[s_1]+1, f[s_2])$. If the two individuals are incomparable, their front numbers are not changed. If $|S| > 2$, $S$ is split into sets $L$ and $H$, as described above. ndhA is called recursively on $H$, so that the individuals in $H$ are all assigned the correct front numbers.

Next, ndhB is called on $L$ and $H$, which updates the front numbers in $L$ depending on the individuals in $H$ that dominate them. For example, if $l \in L$ is dominated by individuals $h_1$, $h_2$, $h_3 \in H$, then, after the completion of ndhB, $f[l]$ will be set to $\max(f[h_1]+1, f[h_2]+1, f[h_3]+1, f[l])$. Since individuals in $L$ have lower $M$-th objective scores than those in $H$, when ndhB is called on $L$ and $H$, only the first $M$-1 objectives need to be considered. When ndhB is called, if $|L|=1$ or $|H|=1$, the set with the single element is directly compared to every element of the other set. The front numbers of members of $L$ which are dominated by members of $H$ are then updated. If both sets

contain more than one individual, and if the two sets are only being compared on $M$=2 objectives, then $L$ is sorted with respect to $H$ in a way similar to the optimal two-objective sweep-line algorithm described in Section 2.1.

If $M$>2, other properties of $L$ and $H$ are checked. If ndhB is comparing $L$ and $H$ on $M$ objectives, and if $\max(x_M(l)) \leq \min(x_M(h)),\ \forall\, l \in L,\ h \in H$, then objective $M$ can be ignored, since the $M$-th objective scores of all elements of $L$ are less than the scores of all elements of $H$, and ndhB can be recursively called on $L$ and $H$, but this time on $M$-1 objectives. Similarly, if $\min(x_M(l)) > \max(x_M(h)),\ \forall\, l \in L,\ h \in H$ nothing else needs to be done, since no elements of $L$ can be dominated by any points in $H$. If none of the previous conditions hold, the element which has the median $M$-th objective score of the larger of $L$ and $H$ is found. $L$ and $H$ are then split into $L_1$, $L_2$, $H_1$, and $H_2$, such that elements in $L_1$ and $H_1$ have lower $M$-th objective scores than elements in $L_2$ and $H_2$. ndhB is then called recursively on these sets. $L_1$ and $H_1$ are compared over $M$ objectives, $L_1$ and $H_2$ only need to be compared over $M$-1 objectives, and $L_2$ and $H_2$ need to be compared in all $M$ objectives. Elements in $L_2$ are guaranteed to be incomparable with members of $H_1$, so ndhB does not have to be called on these two sets.

Jensen shows that the running time of the D&C algorithm is bounded above by $O(N \cdot \log^{M-1} N)$ time [9]. This is a factor of log$N$ slower than SD&C, however, Jensen's algorithm has the advantage of sorting the entire dataset at once, as opposed to finding just the skyline. If SD&C was repeatedly applied to find each front of a dataset, this could lead to $O(N^2 \cdot \log^{M-2} N)$ time, if each individual in the dataset was in its own front [9].

# 3 Limiting Index Sort

Like SaLSa, the *limiting index sort* (LIS) method uses the ideas of pre-arranging the individuals before sorting, as well as including a stopping condition. However, instead of pre-sorting the individuals only once, as do SFS and SaLSa, LIS creates $M$ separate orders of the tuples. With a small variation, LIS can be implemented as either a skyline algorithm, or a non-dominated sorting algorithm. The skyline version is presented first.

## 3.1 Limiting Index Sort - Skyline

If the skyline of a group of $N$ individuals, each with $M$ objectives, needs to be found, then as a first step the tuples are sorted into $M$ lists, $l_1,\ldots,l_M$, based on their scores in objectives $f_1,\ldots,f_M$, respectively. In the case where there is a tie between two or more individuals' scores in an objective, the individuals can just be compared in a different objective (i.e. this is a lexicographic sort). It is easy to see that if the individuals are sorted by their scores in any one objective (from highest to lowest), then a tuple cannot be dominated by any ones which come after it in this list. The $M$ orders, $l_1,\ldots,l_M$, will (in general) be unique.

After the $M$ orders have been created, they can be iterated through in an intelligent way to reduce the total number of comparisons needed to determine the non-dominated front. Clearly, the individuals at the beginning of every order (denoted $l_1(1),\ldots,l_M(1)$) must be part of the skyline, since they cannot be dominated by any other individual. In this way, up to $M$ (if $l_1(1),\ldots,l_M(1)$ are unique) individuals can be added to the skyline, without doing a single comparison between any individuals. Next, the second individual in each order is considered. Starting with $l_1$, it must be determined whether or not $l_1(2)$ is part of the skyline. If it has already been considered for inclusion (i.e. if $l_1(2) = l_2(2) \wedge \ldots \wedge l_M(2)$), then it's already known whether or not it is part of the skyline, and no dominance tests are necessary. However, if it has not yet been considered, then it needs to be determined through dominance tests whether or not any member of the skyline dominates it. The naïve way to determine this would be to simply compare $l_1(2)$ with every skyline member. However, since it is only the second point in $l_1$ which is being considered, the only point that could possibly dominate it is $l_1(1)$, hence only one comparison needs to be made. Next, $l_2(2)$ is considered, and, if this individual has not yet been considered, it must be compared to $l_2(1)$. The tuples are iterated through in this way, and each tuple which has not been previously considered is only compared to skyline members which come before it in the current list.

Like for SaLSa, there is also a stopping condition in LIS which can allow the skyline search to stop before performing comparisons on every individual. Each individual has a position in each of the $M$ orders, and it is easy to determine an individual's maximum position, $p_{max}$, of these $M$ orders (i.e. the individual's position in the order where it is furthest from the beginning of the list). A minimum order-position, $p_{min}$ can be defined similarly; $p_{min}$ is simply the minimum position of the individual over $l_1,\ldots,l_M$. Since each order is based on the individuals' scores in one of the $M$ objectives, if an individual $A$'s maximum order-position is $k$, then clearly any individuals with $p_{min} > k$ are dominated by $A$. When searching for the skyline, if the minimum maximum order-position of all current skyline members is $p_{minimax}$, then only the individuals with $p_{min} \leq p_{minimax}$ need to be explicitly compared with skyline members to determine their inclusion in the skyline, as all individuals with $p_{min} > p_{minimax}$ are definitely not in the front. Each time an individual is added to the front, it is easy to update $p_{minimax}$, by simply assigning the minimum of

$p_{minimax}$ and the newly added individual's $p_{max}$ as the new $p_{minimax}$. When iterating through the individuals, once the order-position greater than $p_{minimax}$ is reached, the algorithm can stop with the guarantee that the entire skyline has been found, since the individual corresponding to $p_{minimax}$ dominates every unconsidered tuple in the dataset.

*Table 3: Sorting with LIS*

| $l_1$ | $l_2$ | $l_3$ |
|---|---|---|
| $p_2$* | $p_4$* | $p_5$* |
| $p_4$* | $p_2$* | $p_2$* |
| $p_3$ | $p_5$* | $p_3$ |
| $p_5$* | $p_3$ | $p_4$* |
| $p_1$ | $p_1$ | $p_1$ |

In Table 3, the skyline of the tuples from Table 1 are found using LIS. As a first step, the individuals are sorted into three different lists, $l_1$, $l_2$, and $l_3$, based on their scores in objectives $f_1$, $f_2$, and $f_3$, respectively. These lists are then iterated through from left-to-right, and top-to-bottom. The tuples in the first row, $p_2$, $p_4$, and $p_5$ are immediately added to the skyline without performing any dominance tests, since they are at the beginning of the three lists. The second row is now considered; points $p_4$ and $p_2$ fill the second row, however, since they are both also in the first row, they have already been considered for skyline inclusion, and no action is taken. Since point $p_2$ is entirely contained in the first two positions of the three lists, it must dominate all remaining tuples which have not yet been considered for skyline inclusion (i.e. $p_2$ dominates $p_1$ and $p_3$). The algorithm can stop at this point, and the entire skyline has been found without performing any dominance tests, and the limiting condition prevents any dominance tests involving tuples $p_1$ and $p_3$.

To perform the $M$ initial sorts, the proposed algorithm requires an overhead of $O(MN \cdot \log N)$. In the best case, there will be one point which is optimal in every objective, in which case the algorithm adds the first point to the skyline, then terminates due to the stopping condition. The worst case is slightly more complicated. In the worst case, the scores in $M-1$ objectives are perfectly positively correlated (i.e. all have a correlation of 1 with respect to each other), and the scores in the $M$-th objective have a correlation of -1 with every other objective. Since two of the objectives are perfectly anti-correlated, the algorithm will compare the first $N/2$ individuals in the first objective list, and the remaining $N/2$ individuals in the anti-correlated objective's list, for a total of $[(N/2)(N/2-1)]/2 + [(N/2)(N/2-1)]/2 \approx N^2/4$ dominance tests. Therefore, in the best-case scenario, the proposed algorithm has computational complexity $O(MN \cdot \log N)$, and in the worst-case, it has complexity $O(MN \cdot \log N + MN^2/4)$. The overhead cost can be significant if one is only looking for a dataset's skyline, but when sorting an entire set of tuples into non-dominated fronts, the original sorted orders can be preserved, reducing the importance of the initial sorting.

By keeping track of the skyline's minimum-maximum order-position, a stopping condition is defined which can rule out a potentially large number of individuals from the skyline, without having to perform any comparisons on the ruled-out individuals. To sort an entire dataset into non-dominated fronts, skylines can be repeatedly found and then removed from the dataset, until the dataset is empty.

## 3.2 Variations

There are a couple of variations on LIS which might make it more efficient at sorting larger datasets. The first is to implement LIS as a non-dominated sorting algorithm (as opposed to a skyline algorithm), to allow it to sort a dataset in a single pass, instead of repeatedly finding and removing the skyline. Consider the following: if $l_1(2)$ is dominated by $l_1(1)$, then $l_1(2)$ is not in the skyline, but since it cannot be dominated by $l_1(3),\ldots,l_1(N)$, it must be part of the second non-dominated front, and it can be added to the front at this point, instead of ignoring it until the entire skyline has been found. This logic can be used for any number of fronts. If any individual is dominated by members of the first $k$ fronts, then it can be compared to the $(k+1)$-th front. Individual stopping criteria can be maintained for each front, so any individual will have a range of fronts that it is potentially a member of. If an individual is compared to each front in this range, in order, then this variation is equivalent to repeatedly applying LIS to find the skyline. However, if a binary search among the possible fronts is implemented, individuals might have to be compared to a fewer number of fronts to determine in which front they belong. This variation is known as $\text{LIS}_{\text{bin}}$.

The second variation is applicable for the specific application of non-dominated sorting within NSGA-II, although it could be used in any problem where only a portion of a dataset needs to be sorted. In NSGA-II, the parent and child populations, both of size $N$, are combined into a set of size $2N$, sorted into non-dominated fronts, and then the top $N$ individuals are chosen as parents for the next generation. Clearly, it is not necessary to sort the entire dataset into fronts, and only the first fronts that collectively contain at least $N$ individuals are needed. With the repeated application of a skyline algorithm, it is easy to keep track of the total number of individuals assigned to a front after each new front is found, and stop the sort when this total number is at least $N$. By stopping the sorting earlier, this variation should always provide some improvement. This variation can only be applied to non-dominated sorting methods which use a skyline algorithm (meaning it cannot be used with Jensen's D&C method). In this paper, this variation will be referred to as $\text{LIS}_{\text{half}}$.

# 4 Experiments and Results

Since D&C and LIS are very different algorithms, LIS is expected to perform better than D&C in some cases, and worse in others. LIS and SaLSa are fairly similar, so they are both expected to realize their best and worst performance in similar cases. A variety of experiments were performed on these three algorithms in an attempt to determine in which cases each performs the best.

## 4.1 Experimental Setup

Three different types of datasets were randomly generated for sorting into non-dominated fronts: correlated, independent, and anti-correlated. Correlated datasets were generated such that all the objective scores had correlations of approximately 0.8 with each other. Objective scores in an independent dataset were drawn randomly and independently of each other. For anti-correlated datasets, lists of objective scores with correlations of about -0.8 were generated in pairs, and the pairs of lists were combined to create a dataset with the desired number of objectives. If an odd number of objectives is needed, the scores for each tuple's final objective are drawn independently of the other scores.

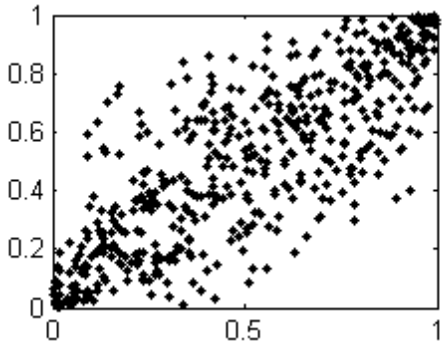


*Figure 1: Correlated Dataset*

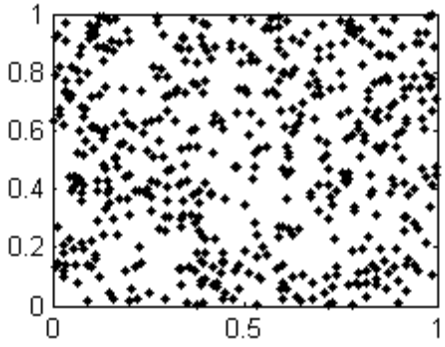


*Figure 2: Independent Dataset*

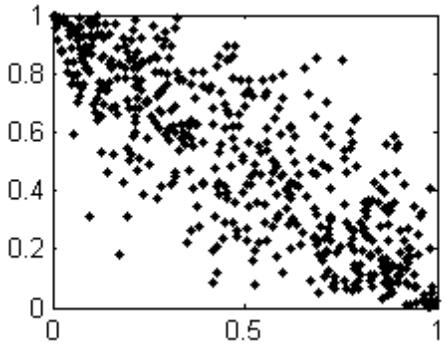


*Figure 3: Anti-correlated Dataset*

Figure 1, Figure 2, and Figure 3 provide examples of correlated, independent, and anti-correlated datasets, respectively, all with two objectives.

Experiments were run on datasets with 3, 5, 7 and 9 objectives. For each number of objectives, 100 of each type of dataset was randomly generated, each containing $2^{13}$ (8192) tuples. Subsets of each dataset were sorted into non-dominated fronts; the populations of each of these subsets were 256, 512, 1024, 2048, 4096, and 8192. Every set was sorted with SaLSa, using three different sorting functions, Jensen's D&C method, and three variants of LIS (LIS, $LIS_{bin}$, and $LIS_{half}$). The overhead sorting required by the SaLSa and LIS variants was performed using Hoare's quicksort [10], and the split operator used by D&C was achieved using Hoare's find algorithm [11]. The numbers of comparisons required by each algorithm to sort each dataset were counted, including comparisons performed in any overhead sorting, D&C's split operator, and dominance tests. One dominance test between two individuals on *M* objectives would be counted as *M* comparisons.

## 4.2 Results

The numbers of comparisons required by each algorithm to sort each different dataset type on five objectives were averaged over the 100 tests. Figure 4, Figure 5, and Figure 6 plot the average number of comparisons required for sorting on five objectives vs. the size of the dataset for anti-correlated, independent, and correlated datasets, respectively. Similar figures displaying the algorithms' performances when sorting on three, seven, and nine objectives can be seen in Annex A. Tables with the average number of comparisons when sorting on each number of dimensions are in Annex B, and their standard deviations are in Annex C. Finally, Table 4, Table 5, and Table 6 list respectively the best performing algorithm on anti-correlated, independent and positively correlated data with respect to the population size and the number of objectives based on the data in Annex B.

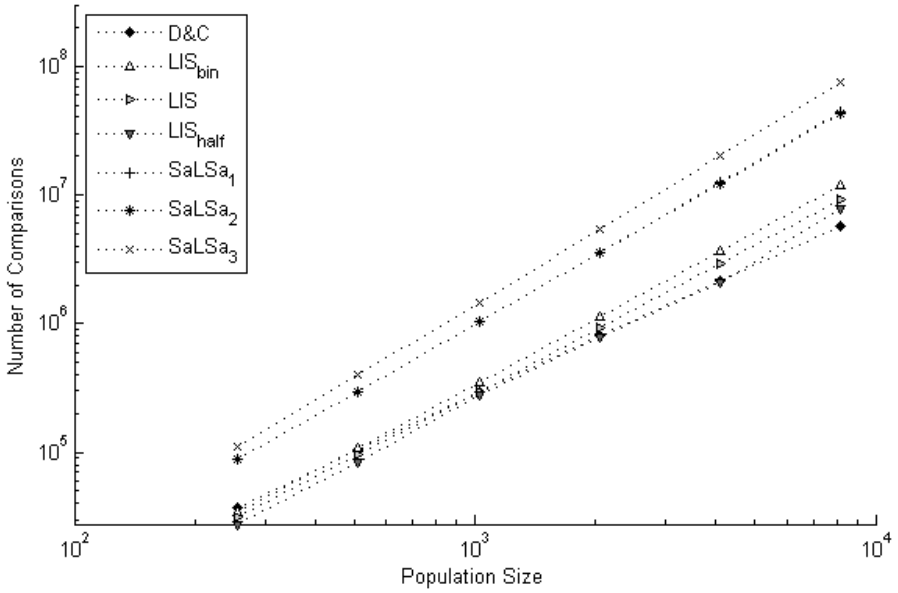


*Figure 4: Number of comparisons required for sorting anti-correlated data on 5 objectives*

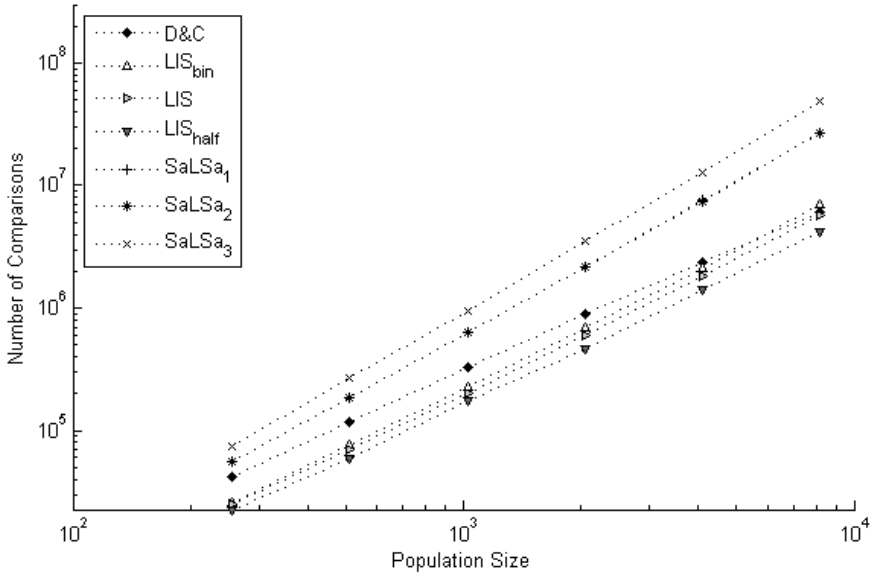


*Figure 5: Number of comparisons required for sorting independent data on 5 objectives*

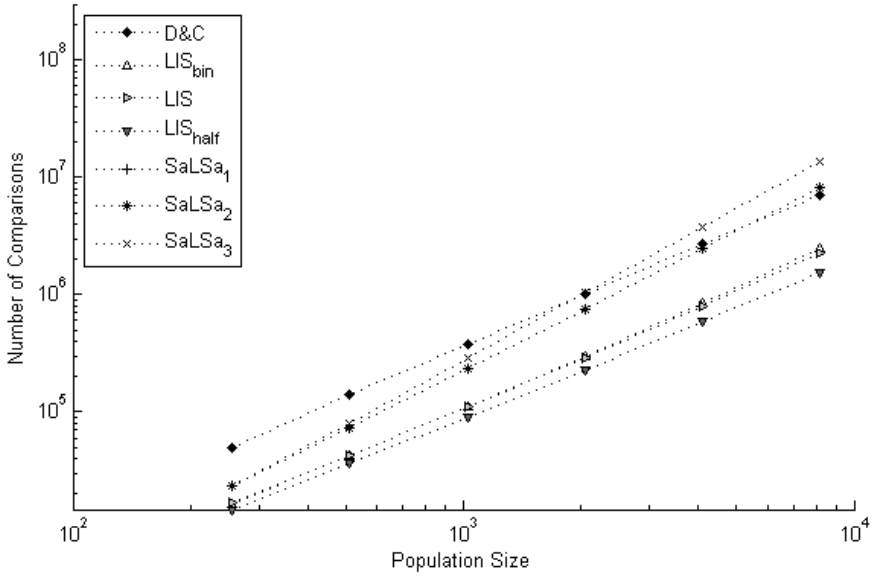


*Figure 6: Number of comparisons required for sorting positively correlated data on 5 objectives*

*Table 4: Highest performing algorithm (mean number of comparisons) on the anti-correlated data set*

| | **Number of Objectives** | | | |
|---|---|---|---|---|
| **Population** | **3** | **5** | **7** | **9** |
| 256 | **$LIS_{half}$** | **$LIS_{half}$** | **D&C** | **D&C** |
| 512 | **$LIS_{half}$** | **$LIS_{half}$** | **D&C** | **D&C** |
| 1024 | **$LIS_{half}$** | **$LIS_{half}$** | **D&C** | **D&C** |
| 2048 | **$LIS_{half}$** | **$LIS_{half}$** | **D&C** | **D&C** |
| 4096 | **D&C** | **$LIS_{half}$** | **D&C** | **D&C** |
| 8192 | **D&C** | **D&C** | **D&C** | **D&C** |

*Table 5: Highest performing algorithm (mean number of comparisons) on the independent data set*

| | Number of Objectives | | | |
|---|---|---|---|---|
| **Population** | **3** | **5** | **7** | **9** |
| 256 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 512 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 1024 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 2048 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 4096 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **D&C** |
| 8192 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **D&C** |

*Table 6: Highest performing algorithm (mean number of comparisons) on the positively correlated data set*

| | Number of Objectives | | | |
|---|---|---|---|---|
| **Population** | **3** | **5** | **7** | **9** |
| 256 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 512 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 1024 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 2048 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 4096 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |
| 8192 | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** | **$LIS_{half}$** |

In all cases, the three LIS variants are clearly better than all SaLSa variants. Furthermore, somewhat surprisingly, $LIS_{bin}$ is slower than LIS in all cases. Since comparing an individual to fronts in order is faster than doing a binary search among the range of fronts it might belong to, most individuals likely belong to one of the first few fronts in their range of possible fronts. This is due to the efficiency of the stopping condition in LIS. $LIS_{half}$ outperformed LIS in all cases, which is obviously due to the fact that $LIS_{half}$ sorts less of the dataset than LIS.

On anti-correlated data, D&C is the winner in most cases (see Table 4). When sorting the anti-correlated data on three objectives for populations up to 2048, and on five objectives for populations up to 4096, $LIS_{half}$ is more efficient than D&C, however, for larger populations or when sorting on more objectives, D&C is the better choice. On the independent and correlated datasets (Table 5 and Table 6 respectively), $LIS_{half}$ was the most efficient algorithm in all tests, except when sorting independent datasets on nine objectives, for populations of 4096 and 8192.

LIS's worst-case scenario is when there is only one front in the dataset (i.e. no individuals are dominated by any others), and its best case scenario is when each individual is in a different front. As individuals' objective scores become more anti-correlated, or if the number of objectives increases, the number of fronts will decrease, as it will become less likely for any one individual to dominate another. Therefore, it is not surprising that LIS requires the fewest comparisons when sorting positively correlated data on three objectives, and it is least efficient when sorting the anti-correlated data on nine objectives. D&C, on the other hand, requires the fewest comparisons to sort anti-correlated data, and the most to sort positively correlated datasets.

In the context of NSGA-II, it is likely that when sorting the combined parent and child populations the objective scores will be anti-correlated. If a very large population and/or number of objectives is being used, D&C should be used. However, when sorting on five objectives or fewer, for datasets of up to 4000 tuples, $LIS_{half}$ may be the better choice. Deb [1] suggested that for a problem with five objectives, a population size of about 250 is sufficient, and for three or four objectives, a population of about 100 is large enough. When running NSGA-II with parameters similar to these, $LIS_{half}$ should be used for non-dominated sorting. In fact, $LIS_{half}$ may outperform D&C in the context of NSGA-II for population sizes of up to 2000 (remember that if the population is $N$, the combined parents and children create a dataset of size $2N$), for optimizations on up to five objectives.

# 5 Conclusions

Non-dominated sorting is an important application of multi-objective optimization problems. Various non-dominated sorting algorithms (for sorting on three or more objectives) were discussed, and their performance was analyzed. An efficient skyline algorithm, SaLSa, was applied as a non-dominated sorting algorithm, however, it was outperformed by a more efficient divide-and-conquer approach.

A new non-dominated sorting algorithm, LIS, was presented, which uses multiple sorted lists to effectively limit the number of dominance tests required to sort a set of individuals into non-dominated fronts. LIS performs the best on datasets which are highly correlated, but outperforms D&C in some situations where the objective scores in a dataset are anti-correlated. When using NSGA-II to solve an optimization problem with up to five objectives, with populations of up to about 2000 individuals (realistic parameters for use with NSGA-II), a variation on LIS, $LIS_{half}$, should be used for non-dominated sorting.

In the future, a more precise assessment of the cases in which it is appropriate to use either LIS or D&C should be performed. In particular, if there are problems where it is desired to sort highly correlated datasets, LIS will be a very efficient choice of algorithm. D&C's performance worsens as the number of objectives increases, and also as the correlation between the objective scores increases. If the correlation between a large number of objectives in a dataset is high enough, there may be some cases where it would be more efficient to use LIS instead of D&C, no matter what the population size of the dataset is. An empirical (or perhaps analytical) study to determine if these cases exist could be useful.

# References


[1] Deb, K., (2001) *Multi-objective Optimization Using Evolutionary Algorithms*. New York: Wiley, 2001.

[2] Deb, K., Pratab, A., Agarwal, S., and Meyarivan, T., (2002) "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 182-197, Apr. 2002.

[3] Borzsony, S., Kossmann, D., Stocker, K., (2001) "The Skyline operator," *17th International Conference on Data Engineering*, pp.421-430, 2001.

[4] Chomicki, J., Godfrey, P., Gryz, J., Liang, D., (2003) "Skyline with presorting," *19th International Conference on Data Engineering*, pp. 717-719, 5-8 March 2003.

[5] Bartolini, I., Ciaccia, P., and Patella, M. (2006), "SaLSa: computing the skyline without scanning the whole sky," *15th ACM international Conference on information and Knowledge Management*, pp. 405-414, Arlington, Virginia, USA, November 06 - 11, 2006.

[6] Bartolini, I., Ciaccia, P., and Patella, M., (2008) "Efficient sort-based skyline evaluation," *ACM Trans. Database Systems*, vol. 33, no. 4.

[7] Mazurek, M., Wesolkowski, S., (2009) "Fleet Mix Computation Using Evolutionary Multiobjective Optimization," *IEEE Computational Intelligence Symposium on Multi-criteria Decision Making*, Nashville, TN, 2009.

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

[9] Jensen, M.T., (2003) "Reducing the run-time complexity of multi-objective EAs: The NSGA-II and other algorithms," *IEEE Transactions on Evolutionary Computation*, vol.7, no.5, pp. 503-515, Oct. 2003.

[10] Hoare, C.A., "Algorithm 64: Quicksort," *Commun. ACM* 4, 7 (Jul. 1961), 321.

[11] Hoare, C.A., "Algorithm 65: find," *Commun. ACM* 4, 7 (Jul. 1961), 321-322.

This page intentionally left blank.

# Annex A Sorting on 3, 7, and 9 Objectives

## A.1 Anti-Correlated Data

Figures 7, 8 and 9 compare the performance of all the algorithms on 3, 7, and 9 objective anti-correlated data sorting problems.

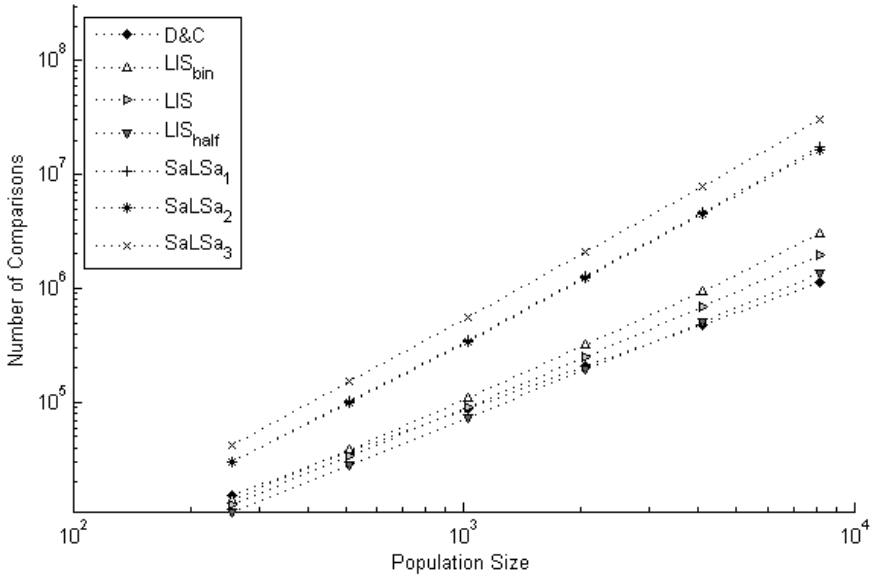


*Figure 7: Number of comparisons required for sorting anti-correlated data on 3 objectives*

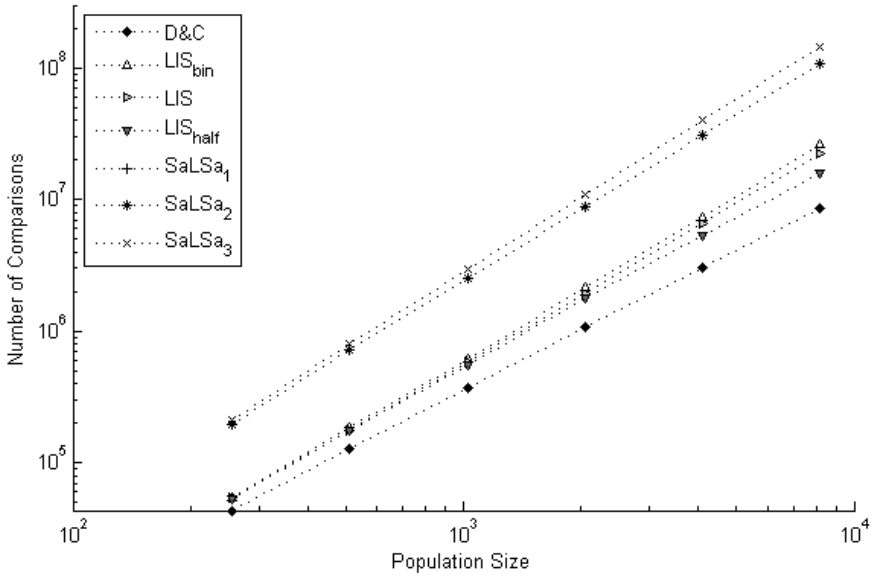


*Figure 8: Number of comparisons required for sorting anti-correlated data on 7 objectives*

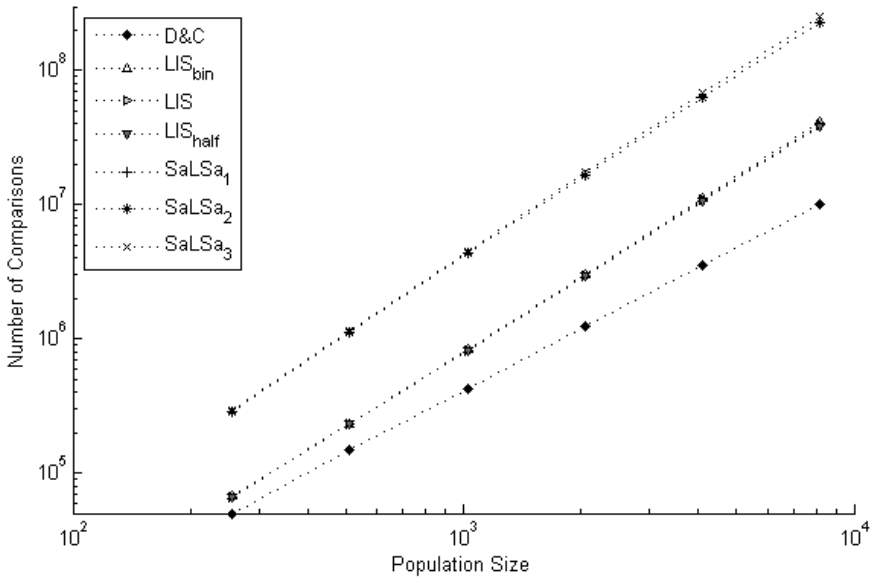


*Figure 9: Number of comparisons required for sorting anti-correlated data on 9 objectives*

## A.2 Independent Data

Figures 10, 11, and 12 compare the performance of all the algorithms on 3, 7, and 9 objective independent (non-correlated) data sorting problems.

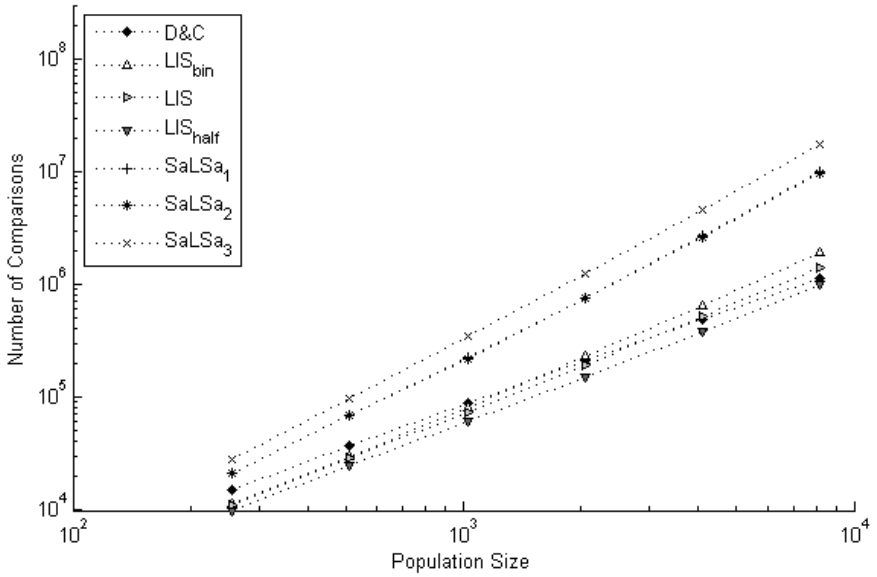


*Figure 10: Number of comparisons required for sorting independent data on 3 objectives*

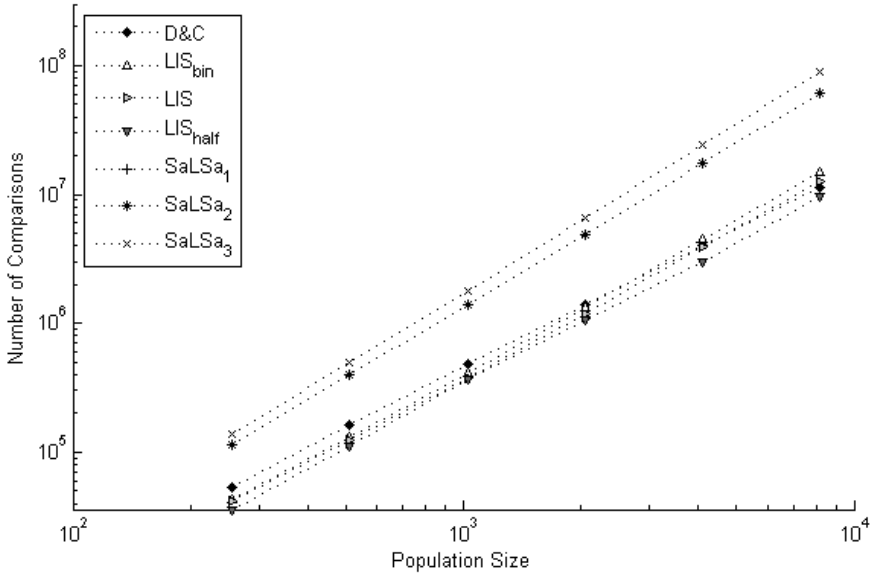


*Figure 11: Number of comparisons required for sorting independent data on 7 objectives*

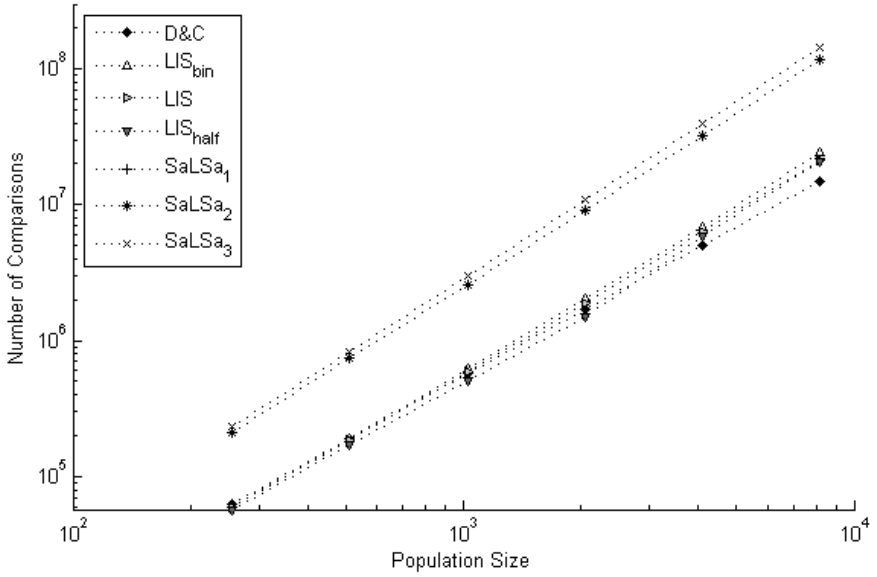


*Figure 12: Number of comparisons required for sorting independent data on 9 objectives*

## A.3 Positively Correlated Data

Figures 13, 14, and 15 compare the performance of all the algorithms on 3, 7, and 9 objective positively correlated data sorting problems.

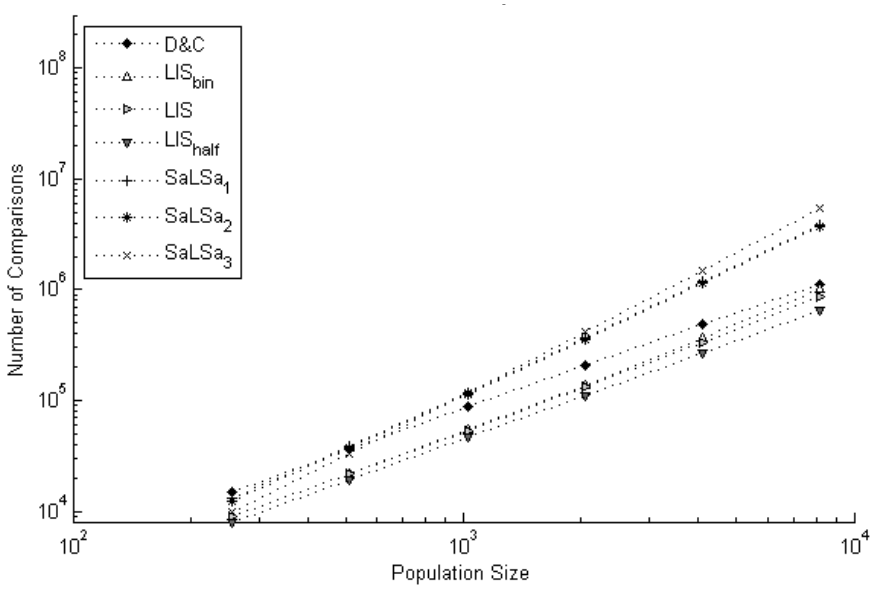


*Figure 13: Number of comparisons required for sorting positively correlated data on 3 objectives*

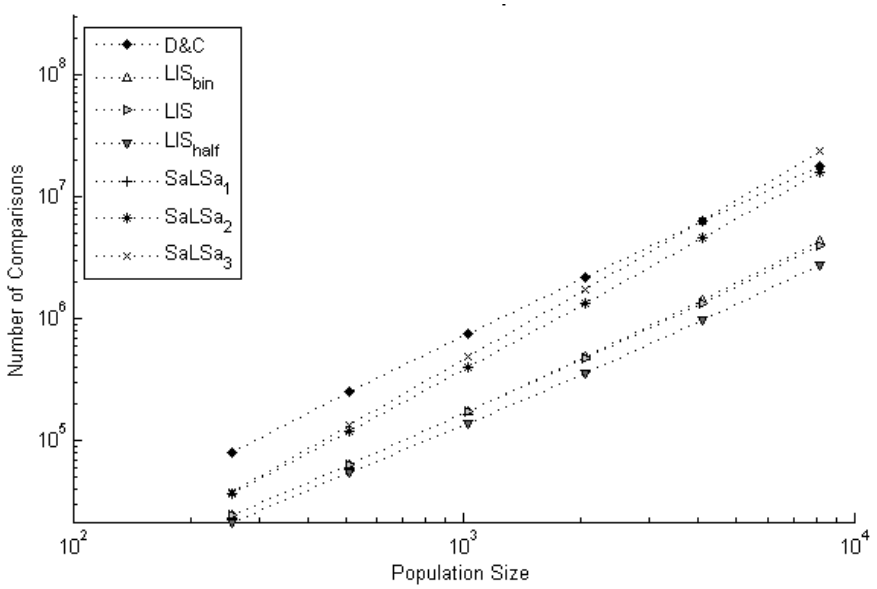


*Figure 14: Number of comparisons required for sorting positively correlated data on 7 objectives*

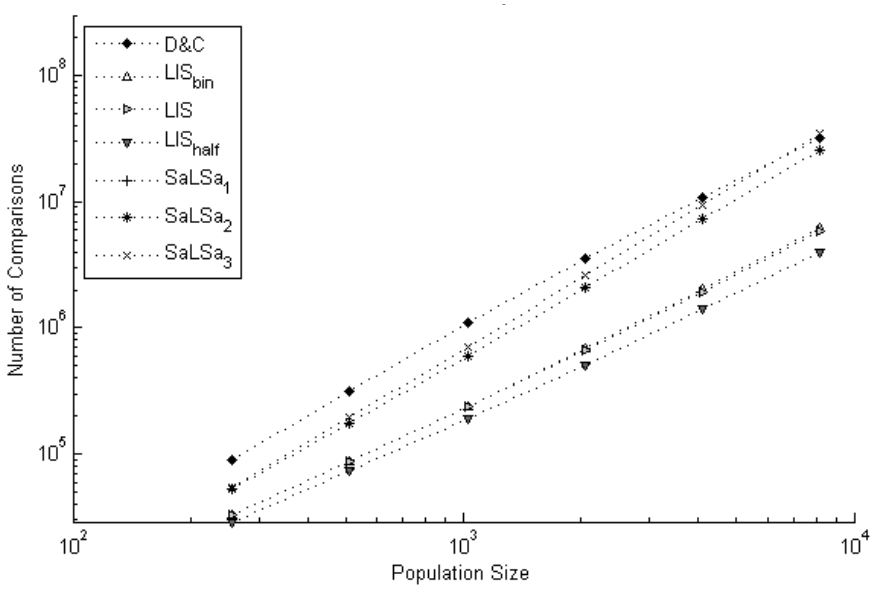


*Figure 15: Number of comparisons required for sorting positively correlated data on 9 objectives*

This page intentionally left blank.

# Annex B Mean Number of Comparisons Required for All Tests

## B.1 Anti-correlated Data

*Table 7: Mean, Anti-correlated, 3 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 15095.63 | 13801.43 | 12788.67 | 11068.56 | 30076.73 | 29920.8 | 42208.13 |
| 512 | 36957.53 | 38250.5 | 34061.11 | 28679.64 | 101459 | 100253.1 | 152104.4 |
| 1024 | 88148.31 | 109588.7 | 91002.78 | 73323.2 | 352888.5 | 345593.2 | 553951.9 |
| 2048 | 208521 | 320217 | 249499.3 | 194377.4 | 1267846 | 1227993 | 2078930 |
| 4096 | 484887.4 | 940734.1 | 687741.7 | 507618.7 | 4644427 | 4451664 | 7867875 |
| 8192 | 1117731 | 2989010 | 1941968 | 1374394 | 17427899 | 16604696 | 29981751 |

*Table 8: Mean, Anti-correlated, 5 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 36703.48 | 34205.52 | 31783.39 | 27692.06 | 87492.01 | 87539.92 | 111278.5 |
| 512 | 105448.9 | 106854.8 | 95727.61 | 84142.28 | 297423.2 | 296789.7 | 401818.1 |
| 1024 | 294379.1 | 347697.3 | 294651.9 | 274611.7 | 1027767 | 1023097 | 1468526 |
| 2048 | 803675.2 | 1126017 | 916635.7 | 773757.7 | 3552324 | 3522361 | 5399405 |
| 4096 | 2149750 | 3612074 | 2898802 | 2089395 | 12466442 | 12294477 | 20066218 |
| 8192 | 5644448 | 11773364 | 9325866 | 7779537 | 44389173 | 43483844 | 75083085 |

*Table 9: Mean, Anti-correlated, 7 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 43506.7 | 55049.11 | 53979.97 | 53710.03 | 197510.2 | 197732.4 | 210954.3 |
| 512 | 127672.7 | 184020 | 177950.2 | 174813.3 | 722809 | 723474.1 | 796895.2 |
| 1024 | 369915.5 | 621403.3 | 586021.1 | 559982.8 | 2545642 | 2546921 | 2952255 |
| 2048 | 1064995 | 2130295 | 1953207 | 1761743 | 8869564 | 8863940 | 10852466 |
| 4096 | 3044211 | 7394572 | 6550984 | 5355010 | 30880919 | 30819282 | 39807217 |
| 8192 | 8641864 | 26130603 | 22181337 | 15728741 | 1.08E+08 | 1.08E+08 | 1.46E+08 |

*Table 10: Mean, Anti-correlated, 9 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 50298.72 | 67274.4 | 67032.62 | 67100.51 | 289900.8 | 289990.3 | 292788 |
| 512 | 147294.8 | 232150.9 | 231132.2 | 230823.3 | 1134459 | 1134873 | 1155907 |
| 1024 | 427239.3 | 831770 | 823977.4 | 822991.3 | 4386825 | 4388879 | 4533957 |
| 2048 | 1230478 | 3038389 | 2986364 | 2974163 | 16721367 | 16730002 | 17614512 |
| 4096 | 3523587 | 11180424 | 10844525 | 10727977 | 62370018 | 62393298 | 67527561 |
| 8192 | 10036780 | 41075264 | 39113044 | 38110644 | 2.28E+08 | 2.28E+08 | 2.55E+08 |

## B.2 Independent Data

*Table 11: Mean, Independent, 3 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 15062.29 | 11447.93 | 11138.29 | 9841.72 | 21305.22 | 21158.21 | 28195.41 |
| 512 | 36882.5 | 30069.48 | 28652.28 | 24372.02 | 68643.3 | 67907.82 | 98241.26 |
| 1024 | 88366.08 | 80720.05 | 74080.23 | 60542.21 | 223767.9 | 220440.2 | 346426 |
| 2048 | 209363.9 | 227932.5 | 194035 | 151479 | 763925 | 750704.6 | 1250399 |
| 4096 | 486764.5 | 644801.5 | 519129.2 | 386004.8 | 2690099 | 2635631 | 4601640 |
| 8192 | 1114861 | 1938667 | 1420164 | 1008026 | 9846186 | 9634397 | 17219746 |

*Table 12: Mean, Independent, 5 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 41611.87 | 26080.56 | 25086.97 | 22691.09 | 56140.02 | 56096.59 | 75049.92 |
| 512 | 118014.8 | 75981.97 | 71098.03 | 59463.89 | 187430.2 | 187228.1 | 267601.1 |
| 1024 | 326754.4 | 223780.1 | 204620.8 | 175227.3 | 630047.9 | 628465.5 | 958921.1 |
| 2048 | 883343.5 | 681705.5 | 605165.4 | 460833.8 | 2164638 | 2155164 | 3512026 |
| 4096 | 2334598 | 2116231 | 1828034 | 1408854 | 7514123 | 7467580 | 12924191 |
| 8192 | 6068341 | 6932049 | 5729010 | 4178063 | 26705807 | 26482204 | 48241120 |

*Table 13: Mean, Independent, 7 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 53774.13 | 42826.34 | 41319.49 | 35919.08 | 114739.6 | 114850.9 | 136758.7 |
| 512 | 161761.2 | 129680.9 | 123430.5 | 109606.8 | 398043.2 | 398248.6 | 491757 |
| 1024 | 479404.3 | 410801.1 | 381392.8 | 363311.2 | 1388044 | 1388015 | 1779739 |
| 2048 | 1400131 | 1330550 | 1197326 | 1056096 | 4872585 | 4871089 | 6554657 |
| 4096 | 4037304 | 4437047 | 3877255 | 3021943 | 17319662 | 17301804 | 24264761 |
| 8192 | 11467694 | 14923529 | 12758678 | 9674789 | 61837792 | 61735908 | 90099286 |

*Table 14: Mean, Independent, 9 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 63260.47 | 59788.14 | 58677.38 | 57167.53 | 212089 | 212237.2 | 230590.9 |
| 512 | 190446.9 | 189935.8 | 183487.9 | 172563.9 | 745922.3 | 746290.4 | 833223.4 |
| 1024 | 571873 | 616198.1 | 583802.9 | 511652.7 | 2591675 | 2592556 | 2985730 |
| 2048 | 1698691 | 2044348 | 1907915 | 1502929 | 9127992 | 9129821 | 10843044 |
| 4096 | 5013593 | 6913896 | 6325377 | 5811171 | 32183892 | 32185571 | 39474446 |
| 8192 | 14722117 | 24167226 | 21529895 | 20594539 | 1.16E+08 | 1.16E+08 | 2.55E+08 |

## B.3 Positively Correlated Data

*Table 15: Mean, Positively correlated, 3 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 15076.54 | 8860.47 | 8890.5 | 8046.16 | 12976.78 | 12518.38 | 9957.04 |
| 512 | 36818.09 | 21933.18 | 21899 | 19349.15 | 38626.67 | 37364.93 | 33505.31 |
| 1024 | 88089.16 | 54426.27 | 53627.16 | 46090.17 | 117040.5 | 113398 | 116247.7 |
| 2048 | 208259 | 138984.2 | 133878.2 | 109983.4 | 365839.9 | 355568.5 | 417570.6 |
| 4096 | 483913.9 | 369042.9 | 334730.1 | 263602.5 | 1168823 | 1138613 | 1489270 |
| 8192 | 1114347 | 1008985 | 864284.1 | 645302.9 | 3867333 | 3772894 | 5490102 |

*Table 16: Mean, Positively correlated, 5 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 49310.42 | 16546.07 | 16672.27 | 14734.66 | 23615.6 | 23381.6 | 23185.1 |
| 512 | 139002 | 42577.37 | 42518.13 | 36235.8 | 73323.15 | 72673.7 | 80876.15 |
| 1024 | 382071.9 | 111608.2 | 109980.7 | 89979.77 | 232112.9 | 230415 | 290877.7 |
| 2048 | 1026001 | 300777.5 | 290355.2 | 224131.7 | 746785.3 | 741850 | 1036275 |
| 4096 | 2694175 | 853867.5 | 799490.1 | 581787.6 | 2474883 | 2460167 | 3768908 |
| 8192 | 6956555 | 2486809 | 2250321 | 1549156 | 8294801 | 8245443 | 13726610 |

*Table 17: Mean, Positively correlated, 7 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 80537.95 | 24560.06 | 24713.17 | 21711.89 | 37057.17 | 36891.9 | 37900.39 |
| 512 | 252655.7 | 64477.5 | 64450.56 | 54847.11 | 120890.6 | 120480 | 134656.9 |
| 1024 | 751849.9 | 173280 | 172433.8 | 138993.6 | 401834.9 | 400897.5 | 488915.7 |
| 2048 | 2185598 | 485599.9 | 472942.3 | 360380.2 | 1344614 | 1342085 | 1752506 |
| 4096 | 6250608 | 1410698 | 1342255 | 968615.4 | 4582514 | 4575344 | 6382065 |
| 8192 | 17557001 | 4282987 | 3963309 | 2708908 | 15911738 | 15886905 | 23612766 |

*Table 18: Mean, Positively correlated, 9 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 89030.43 | 32893.51 | 33147.36 | 29120.88 | 53036.23 | 52915.72 | 54764.23 |
| 512 | 318297.9 | 87088.19 | 87373.77 | 74532.94 | 176695.1 | 176445.8 | 194158.1 |
| 1024 | 1101836 | 236921.7 | 236689.5 | 191737.7 | 601462.5 | 600990.4 | 700400.8 |
| 2048 | 3529162 | 676692.6 | 665096.3 | 506273.8 | 2094247 | 2093259 | 2579344 |
| 4096 | 10707085 | 2007351 | 1927649 | 1397165 | 7316897 | 7314303 | 9474684 |
| 8192 | 31805284 | 6187938 | 5813843 | 3968006 | 25852528 | 25844011 | 34831803 |

# Annex C Standard Deviation of Number of Comparisons Required for All Tests

## C.1 Anti-correlated Data

*Table 19: Standard deviation, anti-correlated, 3 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 339.918 | 634.2964 | 460.7974 | 473.7056 | 2019.53 | 1998.357 | 2950.351 |
| 512 | 731.9588 | 1673.007 | 1094.186 | 1344.075 | 6187.235 | 6152.341 | 9075.243 |
| 1024 | 1376.468 | 4186.661 | 2142.741 | 2655.821 | 18592.51 | 18401.48 | 32013.25 |
| 2048 | 2377.313 | 9762.178 | 5380.751 | 6834.808 | 57206.08 | 58466.32 | 99907.63 |
| 4096 | 6339.695 | 30829.48 | 12746.41 | 15718.61 | 225218.3 | 215122.3 | 289733.7 |
| 8192 | 11719.02 | 96422.73 | 37307.06 | 39397.55 | 764041.4 | 717585.9 | 839528.7 |

*Table 20: Standard deviation, anti-correlated, 5 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 862.4452 | 1420.085 | 1278.895 | 1756.498 | 5964.796 | 6097.542 | 6286.959 |
| 512 | 1766.507 | 4732.886 | 3411.784 | 8366.699 | 16774.57 | 17051.36 | 19353.13 |
| 1024 | 3114.359 | 14934.17 | 8674.254 | 11619.57 | 46020.23 | 45724.92 | 62434.97 |
| 2048 | 5505.28 | 47834.88 | 25625.49 | 37935.53 | 147191.4 | 144939.1 | 205317.9 |
| 4096 | 10118.67 | 130161.5 | 70431.4 | 122389.6 | 384421 | 377112.5 | 654959.1 |
| 8192 | 18707.02 | 337354.2 | 197711.5 | 264189.6 | 959546.5 | 920342.7 | 2220051 |

*Table 21: Standard deviation, anti-correlated, 7 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 1326.76 | 1918.521 | 1919.832 | 2009.744 | 8831.702 | 8828.204 | 5718.296 |
| 512 | 2587.686 | 5915.159 | 6054.771 | 6838.108 | 29848.57 | 29866.62 | 21661.66 |
| 1024 | 5463.152 | 17136.15 | 16151.12 | 20209.68 | 82768.23 | 83248.4 | 67342.09 |
| 2048 | 11131.27 | 57595.83 | 51337.63 | 70346.22 | 287454 | 288662.2 | 250970.6 |
| 4096 | 22346.07 | 157346.8 | 129604.5 | 200076.6 | 752444.9 | 753739.2 | 778867.5 |
| 8192 | 49271.76 | 508350.3 | 381014.5 | 959983.7 | 2325600 | 2308190 | 2733797 |

*Table 22: Standard deviation, anti-correlated, 9 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 1399.673 | 1442.976 | 1624.45 | 1609.745 | 3968.5 | 3899.229 | 2267.548 |
| 512 | 2922.099 | 4462.149 | 4487.79 | 4544.556 | 17762.95 | 17698.56 | 11785.55 |
| 1024 | 5667.048 | 14982.52 | 14949.7 | 15380.85 | 71500.69 | 71299.26 | 48351.31 |
| 2048 | 12880.15 | 50149.91 | 50875.62 | 53501.85 | 289357.7 | 289699.2 | 204317 |
| 4096 | 29751.8 | 158395.9 | 164275.6 | 178521.1 | 961632.7 | 964516.1 | 783622.5 |
| 8192 | 50016.27 | 553874.4 | 554636.1 | 649000.8 | 3489156 | 3510561 | 2909973 |

## C.2 Independent Data

*Table 23: Standard deviation, independent, 3 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 351.6455 | 545.5887 | 483.3572 | 450.0527 | 1475.854 | 1416.551 | 2527.808 |
| 512 | 631.5915 | 1185.484 | 1011.88 | 1202.55 | 3913.26 | 3733.01 | 6739.389 |
| 1024 | 1286.89 | 3279.843 | 2611.68 | 2630.654 | 12144.19 | 11194.72 | 20368.49 |
| 2048 | 2716.553 | 7624.354 | 5492.376 | 5454.612 | 36908.81 | 34159.06 | 58803.05 |
| 4096 | 6242.771 | 19899.36 | 14950.39 | 12961.23 | 114071.4 | 104137.9 | 190307.5 |
| 8192 | 10454.16 | 64259.07 | 36051.16 | 29900.99 | 364037.7 | 348818.2 | 579061.8 |

*Table 24: Standard deviation, independent, 5 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 880.076 | 1279.47 | 1143.497 | 1658.053 | 3815.954 | 3835.58 | 5211.271 |
| 512 | 1720.915 | 3673.452 | 2727.516 | 3920.11 | 11632.55 | 11532.68 | 16458.98 |
| 1024 | 3082.392 | 12107.61 | 8989.995 | 11837.61 | 37001.03 | 36397.57 | 48935.62 |
| 2048 | 6417.672 | 28517.37 | 19971.81 | 30213.15 | 95300.32 | 93185.9 | 142909.6 |
| 4096 | 10344.1 | 97770.99 | 71662.06 | 112443.7 | 328824.3 | 316013.1 | 449278.9 |
| 8192 | 20605.88 | 292394 | 218207.6 | 331087.2 | 970264.9 | 916733.5 | 1462187 |

*Table 25: Standard deviation, independent, 7 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 1285.741 | 2281.039 | 1904.676 | 2702.267 | 9072.434 | 9096.585 | 10954.55 |
| 512 | 2866.427 | 6547.897 | 5605.231 | 10765.51 | 25885.53 | 25952.69 | 32592.94 |
| 1024 | 6768.354 | 16331.53 | 12572.95 | 16875.56 | 65689.23 | 65772.9 | 87122.11 |
| 2048 | 13857.12 | 48604.89 | 35357.79 | 61362.06 | 197405.2 | 197011.6 | 269404.6 |
| 4096 | 26426.57 | 169596.5 | 121978.5 | 186513.4 | 606281 | 604013.4 | 759070.2 |
| 8192 | 54856.7 | 610665.1 | 433083.8 | 1425477 | 2116481 | 2098318 | 2606316 |

*Table 26: Standard deviation, independent, 9 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 1734.064 | 2321.772 | 2386.648 | 3091.693 | 16522.13 | 16611.66 | 14475.29 |
| 512 | 3788.913 | 7054.398 | 7170.973 | 10073.17 | 49665.47 | 49822.72 | 43845.52 |
| 1024 | 9217.546 | 22151.86 | 19967.52 | 30824.75 | 127520.2 | 128119.8 | 127358.6 |
| 2048 | 18532.49 | 78011.62 | 67311.81 | 111260.2 | 461198.6 | 462274.2 | 419227.5 |
| 4096 | 46682.08 | 259495.2 | 215045.9 | 688679.6 | 1301883 | 1301902 | 1279604 |
| 8192 | 94197.35 | 708930.6 | 560283.6 | 758886 | 3548287 | 3551203 | 4247243 |

## C.3 Positively Correlated Data

*Table 27: Standard deviation, positively correlated, 3 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 358.4789 | 308.5008 | 296.0121 | 274.4543 | 545.0924 | 523.1614 | 767.4425 |
| 512 | 667.6598 | 736.1475 | 741.4397 | 695.3863 | 1410.255 | 1396.135 | 2150.031 |
| 1024 | 1283.916 | 1321.363 | 1394.34 | 1356.49 | 3520.087 | 3438.049 | 5393.545 |
| 2048 | 2969.135 | 4008.121 | 3447.346 | 2999.592 | 9422.429 | 9324.87 | 17938.43 |
| 4096 | 6072.295 | 8563.753 | 7005.678 | 6508.38 | 29218.44 | 29534.44 | 56666.15 |
| 8192 | 11041.15 | 19395.56 | 16909.23 | 15040.87 | 88563.01 | 90106.14 | 177265.4 |

*Table 28: Standard deviation, positively correlated, 5 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 733.2873 | 545.238 | 524.6377 | 582.4195 | 1180.031 | 1193.817 | 1777.813 |
| 512 | 1818.418 | 1306.325 | 1311.796 | 1288.23 | 2694.686 | 2731.873 | 4982.74 |
| 1024 | 3393.584 | 3226.251 | 3094.674 | 3236.142 | 8039.497 | 8183.072 | 12411.1 |
| 2048 | 6577.033 | 7891.273 | 7026.954 | 7468.203 | 24011.65 | 24127.24 | 40493.85 |
| 4096 | 12816.32 | 20836.24 | 18046.68 | 19524.4 | 59588.95 | 59715.8 | 119559.7 |
| 8192 | 24819.33 | 62405.61 | 50463.04 | 48699.42 | 196891.7 | 196180.6 | 335805.3 |

*Table 29: Standard deviation, positively correlated, 7 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 2327.974 | 754.6573 | 684.0627 | 847.7003 | 1751.337 | 1776.13 | 2343.949 |
| 512 | 3246.837 | 1731.975 | 1960.995 | 2256.497 | 4991.095 | 5036.946 | 7401.988 |
| 1024 | 5517.548 | 5081.598 | 4248.892 | 5579.636 | 16750.58 | 16853.53 | 23676.27 |
| 2048 | 11322.89 | 12617.86 | 10972.94 | 15832.19 | 46757.67 | 46843.64 | 65286.94 |
| 4096 | 23281.92 | 36904.87 | 31131.73 | 39594.23 | 139485.1 | 140015.9 | 234946.2 |
| 8192 | 48215.66 | 99200.68 | 85348.84 | 113057.3 | 361027.9 | 360819.4 | 588417 |

*Table 30: Standard deviation, positively correlated, 9 objectives*

| Population | D&C | $LIS_{bin}$ | LIS | $LIS_{half}$ | $SaLSa_1$ | $SaLSa_2$ | $SaLSa_3$ |
|---|---|---|---|---|---|---|---|
| 256 | 2980.667 | 1021.234 | 982.6211 | 1231.465 | 2973.516 | 2996.687 | 3568.975 |
| 512 | 7877.694 | 2472.928 | 2634.772 | 3289.258 | 8394.96 | 8429.754 | 11837.33 |
| 1024 | 16193.1 | 6148.458 | 6449.706 | 8920.278 | 23918.75 | 23997.73 | 32226.9 |
| 2048 | 25489.52 | 19039.52 | 18551.06 | 24392.92 | 81610.14 | 81929.13 | 99056.08 |
| 4096 | 42939.96 | 50373.75 | 44228.21 | 70755.93 | 228812.1 | 229654.5 | 294115 |
| 8192 | 92821.2 | 159994 | 140672.9 | 218831.3 | 734706.4 | 735651.7 | 936938.5 |

# List of Abbreviations/Acronyms

| | |
|---|---|
| BNL | Block-Nested Loops (skyline algorithm) |
| D&C | Divide and Conquer (non-dominated sorting algorithm) |
| DND | Department of National Defence |
| DRDC | Defence Research & Development Canada |
| DRDKIM | Director Research and Development Knowledge and Information Management |
| LIS | Limiting Index Sort |
| MOO | Multi-objective Optimization |
| ndhA | ndhelper_A (helper function for D&C) |
| ndhB | ndhelper_B (helper function for D&C) |
| NDF | Non-dominated front |
| NDS | Non-dominated set |
| NSGA-II | Non-dominated Sorting Genetic Algorithm II |
| R&D | Research & Development |
| SaLSa | Sort-and-Limit Skyline algorithm (skyline algorithm) |
| SD&C | Skyline Divide & Conquer (skyline algorithm) |
| SFS | Sort-Filter Skyline (skyline algorithm) |

# Distribution list

Document No.: DRDC CORA TM 2010-097

| Print | email | LIST PART 1: Internal Distribution by Centre |
|---|---|---|
| | 1 | Air SH |
| | 1 | DASOR |
| | 1 | CFAWC OR |
| | 1 | ORAD (1 CAD OR Team) |
| 1 | 1 | DRDC CORA Library (CD) |
| 2 | 1 | Authors |
| 3 | 6 | TOTAL LIST PART 1 |
| | | **LIST PART 2: External Distribution by DRDKIM** |
| | 1 | DRDKIM |
| 0 | 1 | TOTAL LIST PART 2 |
| **3** | **7** | **TOTAL COPIES REQUIRED** |

## DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Canadian Forces Aerospace Warfare Centre (CFAWC)
OR Team
Centre for Operational Research and Analysis
Defence R&D Canada
3701 Carling Avenue
Ottawa, ON K1A 0K2
Canada

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Limiting Index Sort: A New Non-Dominated Sorting Algorithm and its Comparison to the State-of-the-Art

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Mazurek, M.; Wesolkowski, S.B.

| 5. DATE OF PUBLICATION (Month and year of publication of document.) | 6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.) | 6b. NO. OF REFS (Total cited in document.) |
|---|---|---|
| May 2010 | 55 | 11 |

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Canadian Forces Aerospace Warface Centre (CFAWC)
Trenton CFB
P.O. Box 1000 Station Forces
Astra, ON K0K 3W0

| 9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) | 9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.) |
|---|---|
| | |

| 10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) | 10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.) |
|---|---|
| DRDC CORA TM 2010-097 | |

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

An important problem in the realm of evolutionary multi-objective optimization (MOO) is that of finding all non-dominated fronts (NDFs). We specifically address the computational efficiency of the non-dominated sorting algorithm for finding the non-dominated fronts for the NSGA-II algorithm. We introduce the Limiting Index Sort (LIS) algorithm which improves on the existing state-of-the-art algorithm for datasets which are positively correlated or uncorrelated. LIS indexes individuals based on their scores in each objective, and intelligently iterates through the index to limit the number of comparisons required between individuals. LIS also makes use of a stopping condition, allowing it to exclude some individuals from the skyline without comparing them to any other individuals. LIS was compared to two other state-of-the-art non-dominated sorting algorithms, the Sort and Limit Skyline Algorithm (SaLSa), and the Divide-and-Conquer (D&C) approach. LIS outperformed SaLSa in all tests, and it outperformed D&C when sorting individuals with positively correlated or uncorrelated objective scores, except for sorting large, uncorrelated, datasets on many objectives. LIS outperformed D&C for sorting small, negatively correlated datasets on three or five objectives. By understanding the appropriate non-dominated sorting algorithm to use in different situations, NSGA-II can be utilized more efficiently for MOO. This will allow optimizations to be run with larger population sizes, more objectives, or for more generations, which should ultimately increase the quality of solutions returned by the algorithm.

Un problème important dans le domaine d'optimisation multi-objective (MOO) évolutive est de trouver tous les fronts non-dominés (NDFs). Nous abordons spécifiquement l'efficacité de l'algorithme de tri non-dominé pour trouver les fronts non-dominés pour l'algorithme NSGA-II. Nous introduisons l'algorithme Limiting Index Sort (LIS) et démontrons sa supériorité pour le tri des données positivement corrélés ou décorrélés. LIS indexe les solutions individuelles sur la base de leurs scores pour chaque objectif, et itère intelligemment à travers les indices pour limiter le nombre de comparaisons nécessaires entre les solutions individuelles. LIS également fait usage d'une condition d'arrêt, ce qui permet d'exclure certaines solutions de la ligne d'horizon, sans les comparer à d'autres solutions. LIS a été comparée à deux autres algorithmes de tri non-dominée de pointe, le Sort and Limit Skyline Algorithm (SaLSa) et l'algorithme Divide-and-Conquer (D&C). LIS a surclassé SaLSa dans tous les tests, et il surpassé D&C dans le tri des individus pour des objectifs à corrélation positive ou étant décorrélé, sauf pour le tri d'un grand nombre d'individus avec un grand nombre d'objectifs decorrélées. LIS a surclassé D&C pour le tri d'un petit nombre d'individus sur trois ou cinq objectifs anti-corrélées. En comprenant les circonstances pendant lesquelles chacun des algorithmes pour le tri de vecteurs non-dominé est meilleur, NSGA-II peut être utilisé plus efficacement pour MOO. Cela permettra de faire des optimisations sur de plus grandes populations, sur plus objectifs, ou pour plus de générations, qui devrait, augmenter la qualité des solutions retournées par l'algorithme.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Non-dominated Sorting; LIS; Limiting; Index; Sort; Multi-objective Optimization; NSGA-II

**Defence R&D Canada**

Canada's Leader in Defence
and National Security
Science and Technology

**R & D pour la défense Canada**

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale


**www.drdc-rddc.gc.ca**