## ОГЛАВЛЕНИЕ

## ВВЕДЕНИЕ

Многие задачи оптимизации в реальном мире являются многокритериальными, то есть требуется максимизировать или минимизировать критерии объектов, которые часто противоречат друг другу. Выбрать единственное решение обычно невозможно, вместо этого получают множества несравнимых объектов. Поиск разнообразных несравнимых решений часто используется в многокритериальных эволюционных алгоритмах.

В области эволюционных алгоритмов рассматривают три основных подхода [1]: на основе Парето фронтов, на основе индикаторов и на основе декомпозиции. Хотя существуют хорошо известные алгоритмы на основе декомпозиции [2] и на основе индикаторов [3], большинство современных алгоритмов основано на Парето: PESA-II [4], NSGA-II [5], [6], SPEA2 [7] и многие другие. Алгоритмы, основанные на Парето, делятся на следующие группы в зависимости от того, как выбираются или ранжируются решения: алгоритмы, поддерживающие недопустимые решения [4, 8, 9], алгоритмы, основанные на недоминирующей сортировке [5, 6], алгоритмы, использующие показатель доминирования [10] или силу доминирования [7]. В данной дипломной работе сфокусировано внимание на недоминирующей сортировке, так как многие популярные эволюционные алгоритмы ее используют [5, 6].

Цель данной работы – разработать алгоритм недоминирующей сортировки, который будет подходить для крупномасштабной многокритериальной оптимизации. Итоговый алгоритм будет гибридом двух хорошо известных алгоритмов: на основе метода “разделяй и властвуй”, изначально предложенного Йенсеном [11], и на основе недоминирующего дерева, предложенного Густавссоном и Соберфильдтом [12].

Предложенный алгоритм во-первых в худшем случае имеет асимптотику алгоритма на основе метода “разделяй и властвуй”, во-вторых он эффективен на практике. Наше экспериментальное исследование показало ускорение обоих родительских алгоритмов на основных видах данных: на равномерно распределенных точках в гиперкубе и на точках в одной гиперплоскости, имеющих один фронт. Размеры множеств точек достигали $10^6$ и с размерностью до $15$.

В Главе 1 данной магистерской работы представлен общий обзор. В разделе 1.1 приведены основные понятие, определение недоминирующей сортировки, а также для подтверждение актуальности данной работы, представлены примеры ее применения. В разделе 1.2 представлено краткий список существующих алгоритмов недоминирующей сортировки и рассмотрены подробно кандидаты для создания гибрида. В разделе 1.3 описаны слабые места алгоритмов и условия при которых возможно создание нового гибридного алгоритма. В разделе 1.4 сформулирована итоговая задачи магистерской диссертации.

В Главе 2 представлены теоретические исследования существующих алгоритмов и приведено теоретическое обоснование разработанного алгоритма недоминирующей сортировки с асимптотической оценкой времени работы. В разделе 2.1 приведено сравнение существующих алгоритмов. В разделе 2.2 описана предлагаемая схема гибридного алгоритма В разделе 2.3 описана попытка адаптации существующих алгоритмов для создания гибрида. В разделе 2.4 представлен новый алгоритм недоминирующей сортировки с доказательством асимптотики времени работы, который впоследствии используется в гибридном алгоритме. В разделе 2.5 представлена итоговая формулировка гибридного алгоритма и приведено доказательство асимптотики времени работы гибридного алгоритма.

В Главе 3 представлена реализация нового и гибридного алгоритма и проведен эмпирический анализ времени работы новых алгоритмов в сравнении с существующими. В разделе 3.1 представлены детали реализации нового алгоритма ENS-NDT-ONE , основанного на алгоритме Густавссона и Соберфильдта. В разделе 3.2 описана реализация гибридного алгоритма на основе алгоритма Буздалова и Густавссона. В разделе 3.3 описан подход, который мы использовали для настройки получившегося алгоритма. В разделе 3.4 представлены результаты сравнения времени работы получившегося алгоритма с существующими на искусственных данных. В разделе 3.4 представлены результаты сравнения времени работы алгоритмов на худшем случае. В разделе 3.6 представлена многопоточная версия гибридного алгоритма.

В заключении представлены итоги работы и предложены некоторые идеи дальнейшего пути развития гибридных алгоритмов недоминирующей сортировки.

# ГЛАВА 1. ОСНОВНЫЕ ПОНЯТИЯ И ОБЗОР СУЩЕСТВУЮЩИХ АЛГОРИТМОВ

В этой главе введены термины и понятия, присутствующие в решении задачи, произведен обзор имеющихся результатов, затем сформулирована постановка задачи.

## 1.1. Недоминирующая сортировка

В данном разделе представлены основные понятия и определение недоминирующей сортировки.

### 1.1.1. Определение

Для определения недоминирующей сортировки необходимо ввести отношение доминирование по Парето. С текущего момента, не умаляя общности, будем считать, что наша задача минимизировать все критерии. Будем обозначать в данной работе $M$ — число критериев точек, $N$ — количество точек.

*Определение.* В $M$-мерном пространстве, точка $p = (p_1, ..., p_M)$ доминирует по Парето точку $q = (q_1, ..., q_M)$, обозначается как $p \prec q$, если для всех $1 \leqslant i \leqslant M$ выполняется неравенство $p_i \leqslant q_i$, и существует такое $j$, что $p_j < q_j$.

*Определение.* Недоминирующая сортировка — это процедура назначения ранов точкам множества $S$ в многомерном пространстве $R^m$. Точки, которые не доминируются ни одной другой точкой, имеют ранг $0$. Остальные ранги назначаются следующим образом: точка имеет ранг $i + 1$, если максимальный ранг среди доминирующих её точек равен $i$.

На рисунке 1 подставлен пример недоминирующей сортировки для девяти точек в двумерном простанстве. Множество точек с одинаковым рангом называют фронтом или слоем. На риссунке изображено три фронта.

### 1.1.2. Применение и актуальность

Первоначально использование недоминирующей сортировки в эволюционных алгоритмах было предложено в [13], она выполнялась за $O(N^3M)$. Позднее время работы было улучшено до $O(N^2M)$ в работе, в которой был представлен знаменитый алгоритм NSGA-II [5].

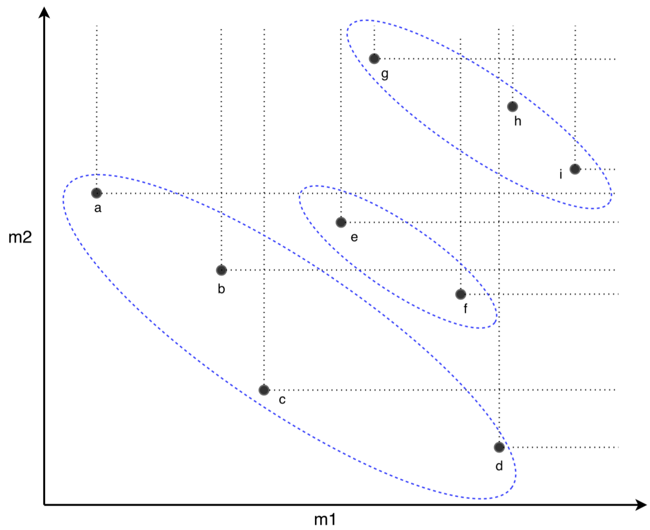



Рисунок 1 – На рисунке представлен пример недоминирующей сортировки для точек $\{a, b, c, d, e, f, g, h, i\}$. Точки $\{a, b, c, d\}$ имеют ранг $0$, $\{e, f\} - 1$, $\{g, h, i\} - 2$.

Эволюционные алгоритмы на каждой итерации генерируют множество потенциальных решений, надо отобрать набор лучших, для чего и требуется неоминирующая сортировка.

Иногда подсчет каждого критерия для каждого решения занимает много времени, в этом случае время работы недоминирующей сортировки не имеет большого значения, так как асимптотика каждой итерации алгоритма зависит от вычисления критериев. Но гораздо чаще встречаются задачи, в которых подсчет каждого критерия занимает значительно меньшее времени, чем необходимо для недоминирующей сортировки. Именно в таких случаях ускорение времени работы недомирирующей сортировки ускорит время выполнения каждой итерации алгоритма, а следовательно и время выполнения всего алгоритма.

Помимо фундаментального стремления к разработке эффективных алгоритмов, исследование, приведенное в данной работы, было мотивировано очень важной практической задачей: многокритериальной задачей оптимизации управления топливом, быстрое решение которой необходимо в ходе функционирования ядерного реактора [14]. Эта задача является сложной задачей оптимизации ряда противоречащих друг другу критериев, таких как мощность, получаемая от реактора, количество нейтронов, вылетающих из реактора и т. д. В 1995 году размер объектов для сортировки достигал $10^5$, сейчас он увеличился до $10^6$.

В настоящее время существуют много разных алгоритмов недоминирующей сортировки, но каждый из них имеет свои слабые стороны. Это означает, что есть возможность создать новый гибридный алгоритм, который будет совмещать в себе два других алгоритма, время работы которого будет лучше, как на практике, так и теоретически.

### 1.2. Анализ существующих алгоритмов

Рассмотрим в данном разделе историю развития алгоритмов недоминирующей сортировки. Особое внимание будет уделено самым эффективным алгоритмам, которые применяются для создания гибрида в данной работе.

#### 1.2.1. Наивные алгоритмы

Наивный алгоритм недоминирующей сортировки перебирает все пары точек и сравнивает их по всем критериям. Точки, которые не доминируются ни одной другой точкой, получают ранг $0$ и исключаются из рассматрения. Далее данная процедура повторяется, причем на каждом новом шаге присваивается новое значение ранга, на единицу больше, чем на предыдущем шаге. Время работы наивного алгоритма $O(MN^3)$, где $N$ — это число точек, а $M$ — размерность пространства, так как сравнение всех пар точек по $M$ критериям займет $O(MN^2)$, а всего шагов алгоритма будет не больше максимального числа рангов — $N$.

В работе Кунга и др. [15] предлагается алгоритм поиска множества недоминируюмых точек. Его вычислительная сложностью равна $O(Nlog^{M-1}N)$. Алгоритм недомирирующей сортировки будет следующим. Сначала в множестве $S$ алгоритм Кунга находит множество точек с рангом $0$. Затем алгоритм Кунга запускается на оставшемся множестве точек, и новому найденому множеству точек присваивается ранг $1$. Процесс выполняется до тех пор, пока имеются точки, которым не присвоен ранг. Описанная процедура в худшем случае выполняется за $O(N^2log^{M-1}N)$, если максимальный ранг точки равен $O(N)$.

Также существует много других алгоритмов, асимптотика которых равна $O(MN^2)$, например, алгоритм ENS Жанга и др. [16].

### 1.2.2. Алгоритм “Разделяй и Властвуй”

Йенсен [11] впервые предложил алгоритм недоминирующей сортировки с вычислительной сложностью $O(Nlog^{M-1}N)$. Однако корректность и оценка сложности алгоритма доказывалась в предположении, что никакие две точки не имеют совпадающие значения ни в какой размерности. Так как часто алгоритмы оптимизации работают с дискретными критериями, совпадение разных решений по одному критерию часто встречающееся событие. Устранить указанный недостаток оказалось достаточно трудной задачей — первой успешной попыткой сделать это, насколько известно исполнителю данной НИР, является работа Фортена и др. [17]. Исправленный (или, согласно работе, «обобщенный») алгоритм корректно работает во всех случаях, и во многих случаях его время работы составляет $O(Nlog^{M-1}N)$, но единственная оценка времени работы для худшего случая, доказанная в работе [11], равна $O(N^2M)$. Наконец, в работе Буздалова и др. [18] предложены модификации алгоритма из работы [11], которые позволили доказать в худшем случае оценку $O(Nlog^{M-1}N)$, не нарушая корректности работы алгоритма.

В данной работы базовым алгоритмом для создания гибрида будет алгоритм Буздалова и др., поэтому опишем его подробнее. Основная идея алгоритма — принцип “разделяй и властвуй”. Исходное множество по медианному значению по последнему критерию разделяется на три подмножества: меньших медианы по последнему критерию, больших и равных ей. Далее алгоритм рекурсивно запускается на каждом подмножестве, при этом по возможности уменьшая количество значимых для сравнения критериев. На рисунке 2 схематически изображена идея алгоритма Буздалова на основе метода “разделяй и властвуй”.

Рассмотрим некоторые процедуры, использующиеся в алгоритме Буздалова, необходимые для понимания итогового гибридного алгоритма. Основными из них являются процедуры $HelperA$ и $HelperB$. Первая процедура представляет из себя основной алгоритм недоминирующей, процедура $DivideConquerSorting$, сортировки на множестве $S$, которое в качестве аргумента приходит на вход. Данная процедура сравнивает точки только по первым $k$ критериям. При запуске недоминирующей сортировки все точки инициализируются рангом $0$, и запускается проце-

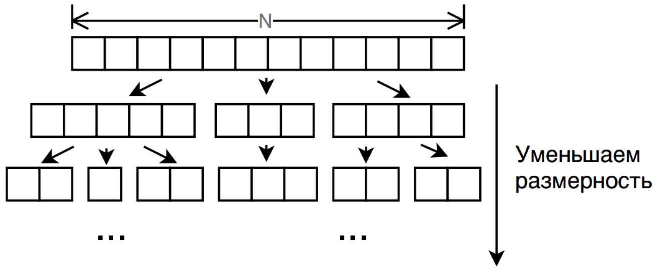



Рисунок 2 – Идея алгоритма Буздалова на основе метода “разделяй и властвуй”.

дура $HelperA$ с аргументами $S$ и $M$, где $M$ — это размерность пространства.

Процедура $HelperA$ на размерности два запускает алгоритм на основе сканирующей прямой с асимптотической сложностью $O(n \log n)$.

В остальных случаях процедура $HelperA$ находит медианное значение $q$ и, используя это значение разбивает, множество точек разбивается на три подмножества $P_L$, $P_M$, $P_R$ с помощью процедуры $Split$. Множество $P_L = \{p \in P | p_j < q\}$, $P_M = \{p \in P | p_j = q\}$ и $P_R = \{p \in P | p_j > q\}$. После такого разбиения ни одна точка из множества правее, не может доминировать ни одну точку из множества левее. Используя это свойство, мы можем найти ранги для множества $P_L$ независимо от других, затем обновить ранги множества $P_M$ на основе ранов множества $P_L$, далее независимо найти ранги множества $P_M$, используя уже поставленные на предыдущем шаге ранги. И продолжить аналогично для третьего множества $P_R$.

Псевдокод основной процедуры недоминирующей сортировки $DivideConquerSorting$ и $HelperA$ представлен на листинге 1.

Следующая процедура $HelperB$ запускается между рекурсивными запусками $HelperA$ на двух подмножествах. Задача этой процедуры — обновить ранги второго множества на основе первого, для дальнейшего запуска на втором множестве процедуры $HelperA$. Данная процедура также использует принцип “разделяй и властвуй” и при расстановке рангов разбивает множества на более мелкие и запускается на них рекурсивно.

На листинге 2 представлен псевдокоде процедуры $HelperB$.

Листинг 1 – Основная процедура $DivideConquerSorting$ и процедура $HelperA$, которая назначает ранги точкам из $S$ по первым $k$ критериям.

**procedure** DivideConquerSorting(S, M)
    Инициализация рангов 0
    $HeplperA(S, M)$
**end procedure**
**procedure** HelperA(S, k)
    **if** $|S| <= 1$ **then return**
    **else if** $|S| = 2$ **then**
        Сравнить точки по первым k критериям
    **else if** $k = 2$ **then**
        Алгоритм на основе сканирующей прямой
    **else**
        $q \leftarrow$ Median$(\{s_m | s \in S\})$
        $P_L, P_M, P_R \leftarrow$ Split$(S, m, q)$
        HelperA($P_L$, $k$)
        HelperB($P_L$, $P_M$, $k - 1$)
        HelperA($P_M$, $k - 1$)
        HelperB($P_L \cup P_M$, $P_R$, $k - 1$)
        HelperA($P_R$, $k$)
    **end if**
**end procedure**

Листинг 2 – Процедура $HelperB$, которая обновляет ранги точек из $R$ на основе множества точек $L$ по первым $k$ критериям.

**procedure** HelperB(L, R, k)
    **if** $|L| <= 1$ or $R <= 1$ **then return**
        Сравнить все пары точкек по первым k критериям
    **else if** $k = 2$ **then**
        Алгоритм на основе сканирующей прямой
    **else if** $\min\{l_k | l \in L\} \leqslant \max\{h_k | h \in H\}$ **then**
        $q \leftarrow$ Median$(\{p_m | p \in L \cup H\})$
        $L_L, L_M, L_R \leftarrow$ Split$(L, m, q)$
        $R_L, R_M, R_R \leftarrow$ Split$(R, m, q)$
        HelperB($L_L$, $R_L$, $k$)
        HelperB($L_R$, $R_R$, $k$)
        HelperB($L_L \cup L_M$, $R_M \cup R_R$, $k - 1$)
    **end if**
**end procedure**

Асимптотическое время выполнения алгоритма сортировки на основе сканирующей прямой $O(n \log n)$, где $n$ — размер множества точек. Используя этот факт и то, что $max(|P_L|, |P_R|) \leqslant 1/2 \cdot |P|$ and $max(|L_L| + |R_L|, |L_R| + |R_R|) \leqslant 1/2 \cdot (|L| + |R|)$, далее можно использовать мастер-теорему[19] для решения рекурентного соотношения и доказать, что асимптотическое время работы алгоритма равно $O(|P| \cdot (\log |P|)^{M-1})$ в худшем случае.

### 1.2.3. Алгоритм Роя и др.

Некоторый интерес для рассматрения в качестве кондидата для создания гибрида представляет алгоритм Роя $Best\ Order\ Sort$ $(BOS)$ [20], его вычислительная сложность $O(MNlogM + MN^2)$. В лучшем случае алгоритм работает за $O(MNlogM)$, что лучше алгоритма предложенного Буздаловым и др, но в худшем случае его асимптотика — $O(MN^2)$.

Опишем кратко идею этого алгоритма, так как в данной работы было рассмотрена воможность создания гибридного алгоритма с использованием алгоритма Роя. Хотя она и не увенчалась успехом, но это исследование представляет некоторый научный интерес. Более детальное описание можно найти в статье [20].

Идея алгоритма в следующем: для каждой точки составим по каждому критерию множество, которое доминируется этой точкой, получаем $M$ множеств для каждой точки. Далее для того, чтобы найти ранг точки $s$ достаточно рассмотреть только одно из множеств соответствующих точке $s$, назовем его $T$. Предположим, максимальный ранг точек из $T$ равен $r$, тогда точка $s$ будет иметь ранг $(r + 1)$. Важным моментом является то, что любое множество из $M$ множеств может быть рассмотрено для поиска ранга $s$. Алгоритм $BOS$ находит для каждой точки минимальное возможное множества для этой цели. Чтобы это обеспечить будем поддерживать множества $Q_1, Q_2, ..., Q_M$ для каждого критерия, каждое $Q_i$ будет содержать отсортированный набор точек по криерию $i$. Далее будем определять ранги точек в следующем порядке: сначала для первой точки из множества $Q_1$, затем для первой точки из $Q_2$, так до множества по последнему критерию $Q_M$. Потом перейдем ко вторым точкам множеств и так далее. Если встретилась точка, ранг которой мы уже определили, пропускаем эту точку. На рисунке 3 изображены $Q_1$ и $Q_2$ для некоторого

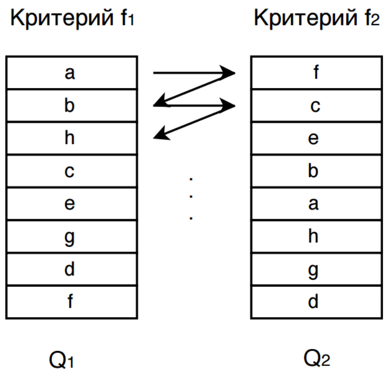



Рисунок 3 – На рисунке представлены отсортированные списки по критерию $f1$ и $f2$. Точка $b$ будет сравниваться только с точкой $a$ и впоследствии ее ранг не будет меняться.

множества двумерных точек. Стрелками показан порядок обхода точек. Процесс завершается как только все точки будут иметь ранг.

Последним важным моментом в описании этого алгоритма является то, что определение ранга в каждом множестве происходит последовательным поиском по подмножествам одного ранга от минимального до максимального, так как присутствует монотонность. То есть, если мы хотим определить ранг точки $s$, мы находим минимальное множество, по которому можно определить ранг, назовем $T$, в нем находим подмножество минимального ранга, если в этом подмножестве ни одна точка не доминирует точку $s$, это означает, что ей присваивается ранг этого подмножества, иначе переходим к подмножеству, ранга котрого на один больше. Монотонность гарантирует, что ни одна точка из других подмножеств в большим рангом, не будет доминимать нашу точку $s$, и мы назначили ранг корректрно. Благоряря монотонности определение ранга происходит быстро. А нарушение ее влечет значительное замедление. Впоследствии именно это станет основной причиной, почему создание гибридного алгоритма на основе алгоиртма Роя и алгоритма Буздалова невозможна.

### 1.2.4. Алгоритм Густавссона и др.

Большой интерес представляет алгоритм Густавссона и Соберфильдта ENS-NDT [12], который основан на идее $k$-$d$ деревьев ($k$-мерных деревьев). Его вычислительная сложность на случайно сгенерированных независимых точек равна $O(N^{1.43})$. Однако в худшем случае алгоритм работает за квадратичное время $O(MN^2)$.

Опишем основную идею этого алгоритма и приведем псевдокоды основных методов. Более детальное описание можно найти в статье [12]. Подробное описание необходимо в данной работе для понимания оптимизаций, модификаций алгоритма и для понимания итогового гибридного алгоритма.

Алгоритм Густавссона и Сиберфильдта ENS-NDT относится к группе алгоритмов Efficient Non-dominated Sort(ENS). Еще одним представилем этой группы является алгоритм ENS-BS (Efficient Non-dominated Sort Binary Strategy), скорость работы которого сильно ухудшается с ростом количества точек. Алгоритм ENS-NDT справляется и с большим количеством точек и с точками большой размерности в общем случае.

Алгоритм ENS-NDT использует недоминирующее дерево(NDTree). Недоминирующее дерево основано на корзиночных(bucket) $k$-$d$ деревьях. Корзиночное $k$-$d$ дерево - это вид бинарного дерева, где хранятся точки $k$-мерного пространства. Каждая вершина дерева ассоциирована с одной размерностью. Листья имеют так называемый bucket size - максимальное количество точек, которое может содержать лист, если появляется необходимость добавить больше точек, вершина делится на две. Медианы, которые используются при построении дерева преподсчитаны для всех вершин для соответствующих им размерностям. Далее, если точка по рассматриваемой в данной вешине координате меньше, чем медианное значение, то процесс добавления продолжается в левом ребенке-вершине, если больше — то в правом. Точки, чьи координаты равны медианному значению, добавляются в левого или правого ребенка, в зависимости от настроек $k$-$d$ дерева. Следующее поколеник вершин ассоциировано с новым критерием, отличным от родительской вершины, обычно используется следующий критерий, то есть критерий родительской вершины + 1 или первый критерий, если родительский параметр ассоциирован с максимальной размерностью. На рисунке 4 подставлена иллюстрация корзиночного $k$-$d$ дерева. Вершина $A$ ассоциирована с критерием $X$, вершины $B$ и $C$ ассоциирована с критерием $Y$, вершины $D$ и $E$ ассоциирована с критерием $X$ снова, размер корзины представленной на рисунке структуры равен трем. Также, чтобы избежать создания слишком большого количества уровней, недоминирующее дерево имеет параметр максимальной глубины. Это означает, что если достигнута максимальная глубина, то параметры максимального количества точек в вершине ингорируется.

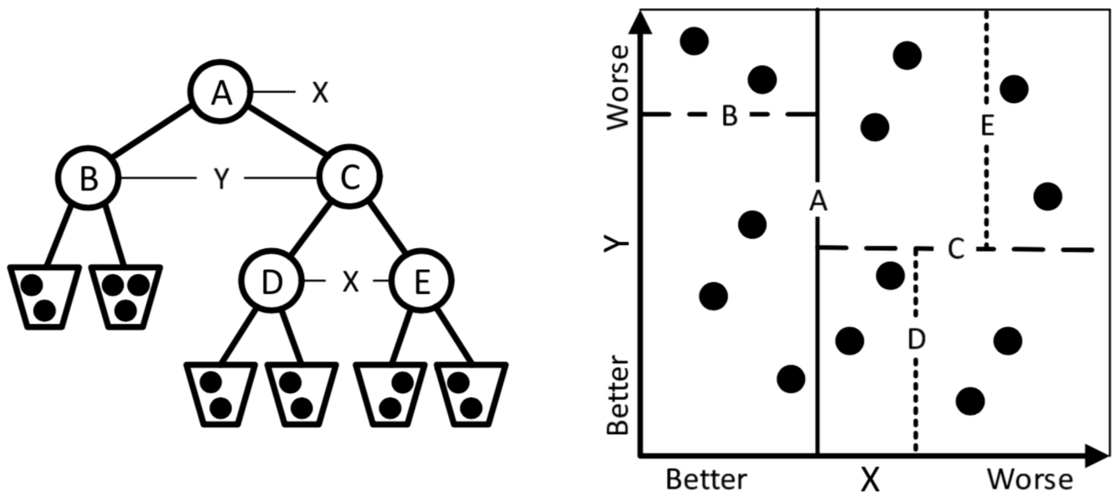



Рисунок 4 – На рисунке представлена иллюстрация внутреннего устройства структуры Bucket k-d tree. Слева изображена структура, справа точки на плоскости, по которым получена структура.

Недоминирующее дерево использует преподсчитанную $split$ структуру, где все возможные медианные значения преподсчитаны до того, как они потребуются. Это отличается от стандартной реализации корзиночного $k-d$ дерева, в котором обычно значение считается в момент переполнения размера корзины в вершине. Это становится возможно сделать, потому что мы знаем множество точек для сортировки заранее. $Split$ структуру удобно хранить в виде дерева, похожего на финальное недоминирующее дерево. Основное преимущество использования $split$ дерева заключается в том, что девево остается сбалансированным, что значительно улучшает производительность добавления вершин и определения ранга.

И так, для выполнения недоминирующей сортировки следует поддерживать отдельное дерево для каждого ранга. На рисунке 5 представлено схематическое представление структуры использующейся в недоминирующей сортировке.

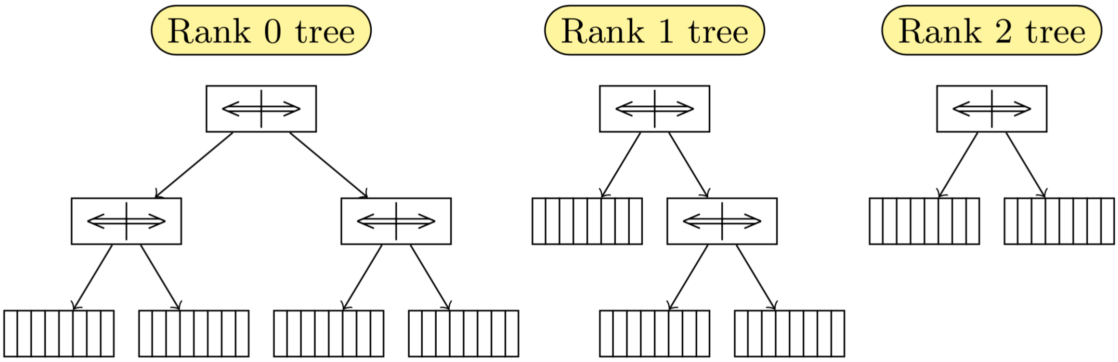



Рисунок 5 – Структура деревьев для алгоритма ENS-NDT. Каждое дерево ассоциировано с отдельным рангом.

Для выполнения недоминирующей сортировки следует выполнить следующие действия:

а) Создать split структуру для всех точек.
б) Осуществить лексикографическую сортировку.
в) Перебирать точки в лексикографическом порядке.
    1) Определить ранг.
    2) Добавить в соответствующее рангу дерево.

На листинге 3 представлен псевдокод основного метода недоминирующей сортировки, который принимает в качестве аргументов множество точек $P$, $M$ - размерность и $B$ - размер корзины, то есть максимальное количество точек в вершине. Для получения $split$ структуры используется функция $CreateSplits$, которая на вход получает множество точек, размер корзины и размерность $M-1$, размерность $M$-ая игнорируется, так как ранее множество точек было лексикографически отсортировано. Лексикографическая сортировка выбрана еще потому, что она позволяет уменьшить число сравнений впоследствии.

Следующим шагом будет создание множества $\mathcal{F}$ и $\mathcal{T}$ на строчке 4-6, $\mathcal{F}$ - это ранжированное множество точек, $\mathcal{T}$ - множество деревьев для каждого ранга, в каждом дереве ни одна точка не доминирует другую точку в том же дереве. Точка $P_1$ добавляется в оба множества с рангом один, так как ни одна другая точка не может доминировать первую в лексикографическои порядке точку. Главный цикл на строке 8 определяет ранги точек и добавляет их в структуру.

Листинг 3 – Главная процедура алгоритма ENS-NDT.

**procedure** ENS-NDT(P, M, B)
    $P \leftarrow Sort(P, a^M \prec b^M, ..., a^1 \prec b^1)$
    $S \leftarrow CreateSplits(P, M-1, B)$
    $\mathcal{F} \leftarrow \{\{P_1\}\}$
    $\mathcal{T} \leftarrow \{newNDTree(S, B)\}$
    $InsertIntoNDTree(\mathcal{T}_1, P_1)$
    $j \leftarrow 1$
    **for** $i = 2, ..., |P|$ **do**
        **if** $P_{i-1} \neq P_i$ **then**
            $j \leftarrow FrontIndexBinarySearch(\mathcal{T}, P_i)$
            **if** $j > |\mathcal{T}|$ **then**
                $F_j \leftarrow 0$
                $\mathcal{T}_j \leftarrow newNDTree(S, B)$
            **end if**
            $InsertIntoNDTree(\mathcal{T}_j, \mathcal{P}_i, )$
        **end if**
        $\mathcal{F}_j \leftarrow \mathcal{F}_j \cup P_i$
    **end for**
    **return** $\mathcal{F}$
**end procedure**

Возможная реализация CreateSplits изображена на листинге 4. Структура NDSplit похожа на NDTree, но вместо точек она хранит медианные значения. Общий подход построения сбалансированных k-d деревьев с помощью метода “разделяй и властвую” описан Бламом и др. [21].

Вкратце опишем процедуру CreateSplit. Процедура принимает четыре аргумента: множество точек $P$, количество критериев $M$, размер корзины $B$ и текущая глубина дерева $d$. При первом вызове процедуры $CreateSplit$ текущая глубина равна $0$. Далее значение текущей глубины дерева используется для выбора координаты в строчек 2, для которой происходит поиск медианного значения. Затем множество точек $P$ сортеруется в строчке 3 по выбранному критерию, а переменной $m$ в строчек 4 присваевается значение медианы, как средненго значения в отсортированном множестве. Далее создается новая вершина дерева, и если множество не помещается в одну вершину по ограничению на размер корзины, то создается два наследника $Better$ и $Worse$. После чего происходит рекурсивный вызов процедуры $CreateSplits$ для обоих подмножеств с увеличеным значением глубины. Таким образом создается все сбалансированное $split$ дерево и возвращается в строчке 12.

Листинг 4 – Пример реализации процедуры $CreatSplit$, которая вычисляет медианные значения для недоминирующего дерева.

1: **procedure** CreateSplit($P, M, B, d \leftarrow 0$)<br>
2:   $o \leftarrow 1 + (d \mod M)$<br>
3:   $P \leftarrow Sort(P, a^o \prec b^o)$<br>
4:   $m \leftarrow P_{1+\lfloor |P|/2 \rfloor}$<br>
5:   $S \leftarrow \{newNDSplit(o, m)\}$<br>
6:   **if** $|P| > B$ **then**<br>
7:    $Better \leftarrow \{P_i, i < 1 + \lfloor |P|/2 \rfloor\}$<br>
8:    $Worse \leftarrow \{P_i, i \geqslant 1 + \lfloor |P|/2 \rfloor\}$<br>
9:    $S.BetterSplit \leftarrow CreateSplit(Better, M, B, d + 1)$<br>
10:    $S.WorseSplit \leftarrow CreateSplit(Worse, M, B, d + 1)$<br>
11:   **end if**<br>
12:   **return** S<br>
13: **end procedure**

Поледней интересной для нас функцией являтся функция определения ранга точки $FrontIndexBinarySearch$, представленная на листинге 5. Эта процедура бинарным поиском определяет минимальное дерево в структуре $\mathcal{T}$, где ни одна точка не доминировала бы рассматриваемую.

Листинг 5 – Процедура определения ранга точки $s$.

```
procedure FrontIndexBinarySearch(T, s)
    i ← 1
    j ← |T| + 1
    while i ≠ j do
        k ← ⌊i + (j − i)/2⌋
        if FromntDominates(T_k, s) then
            i ← k + 1
        else
            j ← k
        end if
    end while
    return i
end procedure
```

Подробное описание алгоритма ENS-NDT необходимо для понимания дальнейших модификаций и самого гибридного алгоритма.

## 1.3. Недостатки существующих алгоритмов

Все описанные выше алгоритмы имеют разные разные преимущества и недостатки. Алгоритм Буздалова “разделяй и властвуй” имеет хорошую асимптотику, даже на самых плохих входных данных. Однако алгоритм сильно замедляется с ростом размерности задачи $M$.

Алгоритм Роя, имеет интересную идею и показал хорошие результаты на практике. Однако теоретическое время его работы пока не исследовано.

Алгоритм Густавссона имеет хорошую асимптотику на случайно распеределенных точках в гиперкубе, но имеет квадратичную асимптотику, в описанном авторами плохом случае.

## 1.4. Постановка задачи

Задача данной работы состоит в разработке нового гибридного алгоритма недоминирующей сортировки и разбивается на подзадачи:

а) Выбрать наиболее подходящие для гибридизации алгоритмы.

б) Адаптировать алгоритмы для гибридизации.
в) Реализовать гибридный алгоритм.
г) Настроить гибридный алгоритм для максимально эффективной работы.

# ГЛАВА 2. МЕТОДЫ РАЗРАБОТКИ ГИБРИДНЫХ АЛГОРИТМОВ НЕДОМИНИРУЮЩЕЙ СОРИТИРОВКИ

В данной главе будет описана схема гибридного алгоритма, приведены требования к кандидатам на роль базовых алгоритмов, на основе которых разрабатывается гибридный алгоритм, приведена адаптация существующих алгоритмов и описан гибридный алгоритм на основе существующих.

## 2.1. Отбор алгоритмов, подходящих для создания гибридного алгоритма

Основным кандидатом для создания гибридного алгоритма недоминирующей сортировки оказался алгоритм Буздалова и др. на основе метода “разделяй и властвуй”. Есть две основные причины, чтобы использовать этот алгоритм:

а) Данный алгоритм основан на методе “разделяй и властвуй”, что позволяет использовать рекурсивные вызовы к качестве точек подключения с одного алгоритма на другой.
б) Данный алгоритм является одним из лучших алгоритма на сегодняшний день, и имеет доказанную, хорошую асимптотику времени работы.

Вторым кандидатом стал алгоритм Роя, который уже показал неплохие результаты в гибриде с алгоритмом Буздалова [22], но алгоритм Роя был только наполовину приспособлен для гибридизации. В этой работе представлена попытка приспособить его полностью.

Третьим кандидатом стал алгоритм Густавссона ENS-NDT, использование которого для создания гибридного алгоритма сильно снизило эффективность гибридного алгоритма недоминирующей сортировки. Вдохновившись идеями алгоритма Густавссона, мы получили новый алгоритм и назвали его ENS-NDT-ONE. Сам по себе он уже имеет некоторый научный интерес, так как имеет хорошую асимптотику и эффективен на практике наравне с лучшими алгоритмами недоминирующей сортировки. Затем мы реализовали гибридный алгоритм на основе алгоритма “Разделяй и властвуй” и алгоритма ENS-NDT-ONE.

## 2.2. Предлагаемая схема гибридного алгоритма

В этом разделе будет описана предлагаемая схема гибридного алгоритма.

### 2.2.1. Выбор момент переключения

Алгоритм "разделяй и властвуй" очень хорошо подходит для создания гибридного алгоритма недоминирующей сортировки, так как в алгоритме рекурсивно вызываются подзадачи меньшего размера и меньшей размерности. В первой главе данной работы описан алгоритм подробно, в оригинальной статье функции, где мы предполагаем переключаться на другой алгоритм, названы $HeplerA$ и $HeplerB$. Мы предлагаем следующую идею создания гибридного алгоритма:

а) Запускаем алгоритм Divide&Conquer, согласно некоторым правилам переключаемся на другой алгоритм.
б) Моменты смены алгоритма:
   1) HelperA
      i. Входные данные: множество точек $S$ с предварительными рангами.
      ii. Результат выполнения: множество точек $S$ с обновленными рангами.
   2) HelperB
      i. Входные данные: множество точек $L$ с окончательными рангами и $R$ с предварительными рангами.
      ii. Результат выполнения: множество точек $R$ с обновленными рангами по множеству $L$.

### 2.2.2. Настройка параметров гибридизации

Настройка гибридного алгоритма будет представлять некоторый диапазон размеров множестве точек для каждой размерности, при котором происходит переключение. Например, при размерности точек три, договоримся, что смена алгоритма происходит если точек не больше 100, а при размерности 4 и более смена происходит при размере множеств точек не более 1000. Параметры гибридизации получаются экспериментальным путем для конкретного вида гибридного алгоритма. Параметры, полученные в данной работе, описаны в разделе 3.3.

## 2.3. Адаптация алгоритмов

В этом разделе опишем адаптацию алгоритмов для гибридизации. Для гибридизации было выбрано два алгоритма: алгоритм Роя и алгоритм Густавссона. Идеи гибридизации и адаптации схожи, поэтому опишем их сначала в общем случае, потом перейдем к деталям каждого алгоритма.

Функция $HeplperA$ — это сама недоминирующая сортировка, поэтому приспосабливать алгоритмы с этой точки зрения не надо.

Напомним, что функция $HeplerB$ в качестве входных параметров принимала два множества $L$ с окончательными рангами и $R$ с предварительными рангами, по результату работы назначаются ранги множеству точек $R$ по множеству точек $L$.

Для $HeplerB$ была предложена следующая идея адаптации для гибридизации:

а) Обходим точки в порядке предложенном в оригинальном алгоритме.
   1) Если точка принадлежит множеству с окончательными рангами $L$, добавляем точку в структуру алгоритма с текущим рангом.
   2) Если точка принадлежит множеству с предварительными рангами $R$, то мы определяем ранг рассматриваемой точки на основе текущего состояния структуры и не добавляем ее в структуру, так как ранжирование происходить только на основе точек из множества $L$.

Основные отличия от оригинального алгоритма:

а) В оригинальной статье на момент начала работы алгоритма все точки имели ранг 0, в нашем случае начальные ранги могут быть любыми.
б) Определение ранга точек надо изменить, чтобы алгоритм учитывал предпоставленные ранги.

Рассмотрим отдельно для каждого алгоритма адаптацию для последующей реализации гибридного алгоритма.

### 2.3.1. Алгоритм Роя и др.

Алгоритм предложенный Роем и др. описан подробно в главе Обзор работы. Ранее был предложен гибридный алгоритм, который использует только момент переключения $HelperA$ [22], второй момент переключения $HelperB$ не был рассмотрен в данной работе.

Определение ранга происходило бинарным или последовательным поиском с нулевого ранга по ранжированному множеству точек. Эффективность этих двух подходов практически совпадала. Найденное множество точек с минимальным рангом, где ни одна точка не доминирует рассматриваемую, означало, что точке можно присвоить ранг этого множества. Был справедлив следующий инвариант: для рассматриваемой точки до некоторого ранга $k$ все множества соответствующие меньшим $k$ рангам имеют хотя бы одну точку, которая доминирует рассматриваемую, а начиная с множества соответствующего рангу $k$ и больше во всех множествах нет ни одной точки, которая бы доминировала рассматриваемую точку. В таком случае точка получает ранг $k$.

В новой версии алгоритма в множестве $L$, точки которого добавляются в структуру позволяющую определять ранг, могут иметь совершенно любые ранги. И точка может доминироваться точкой, например, ранга $k$, но не доминироваться точкой $k-1$ ранга, это означает, что больше нельзя использовать бинарный поиск для определения ранга. Единственным выходом является перебор множеств начиная с наибольшего в структуре, пока не найдется множество, где есть хотя бы одна точка, которая доминирует рассматриваемую. После этого можно сделать вывод, что точка имеет ранг найденного множества + 1.

После такого значительного изменения алгоритма мы провели замеры времени работы и оказалось, что новый алгоритм работает на порядок хуже оригинального алгоритма, это означает, что создать гибридный алгоритм на его основе нельзя, по крайней мере используя такой подход гибридизации. Теоретическое исследование времени алгоритма является достаточно трудоемкой задачей и из-за настолько плохого практического результата, мы не стали им заниматься.

### 2.3.2. Алгоритм Густавссона и др.

Следующим кандидатом для гибридизации был алгоритм Густавссона и др.

Время работы этого алгоритма на случайно сгенерированных, независимых точках в гиперкубе составляет $O(N^{1.43})$, но авторами работы описан худший случай, на котором асимптотика становится квадратичной и составляет $O(MN^2)$, на больших N время работы становится неприемлемо большим. Алгоритм выбран в качестве основного кандидата для гибридизации, потому что он является самым эффективным алгоритмом на сегодняшний день в общем случае.

## 2.4. Выводы к главе 2

В оригинальном алгоритме точки, которые необходимо отсортировать, инициировались рангом $0$, и ранг назначался постепенно, в лексикографическом порядке. Теперь точки в качестве аргументов приходят в алгоритм Густавссона из базового алгоритма Буздалова. Точки могут иметь совершенно разные ранги, в том числе для конкретной рассматриваемой точки, может оказаться, что ни одна точки с рангом $0$, ее не доминирует, что раньше бы привело, к присвоениею этой точки ранга $0$, а точка с рангом больше $0$ доминирует. То есть определение ранга происходило похожим на алгоритм Роя образом, присутствовала монотонность, то есть первое найденное множество с минимальным рангом, где ни одна точка не доминирует рассматриваемую, означает, что точке можно присвоить ранг этого множества. В структуре недоминирующих деревьев, где каждое дерево соответствует своему рангу, мы бинарным поиском определяли минимально по рангу дерево, где, аналогично, ни одна точка не доминирует рассматриваемую, тогда рассматриваемой точке присваивался ранг этого дерева. Но аналогично проблеме описанной выше, в алгоритме Роя, инвариант позволяющий осуществить бинарный поиск перестает работать. Теперь надо искать точку с максимальным рангом, которая доминирует рассматриваемую. Тогда точке, для которой производится определение ранга, назначается ранг найденной точки $+1$.

Таким образом, алгоритм был адаптирован следующим образом: вместо бинарного поиска использовался последовательный перебор с дерева, соответствующего максимальному рангу, переходя к деревьям меньшего ранга. После этого, мы провели эмпирический анализ времени работы, который показал, что потеря монотонности, критически влияет на время работы. Теоретический анализ адаптированного алгоритма не был произведен, из-за плохого практического результата.

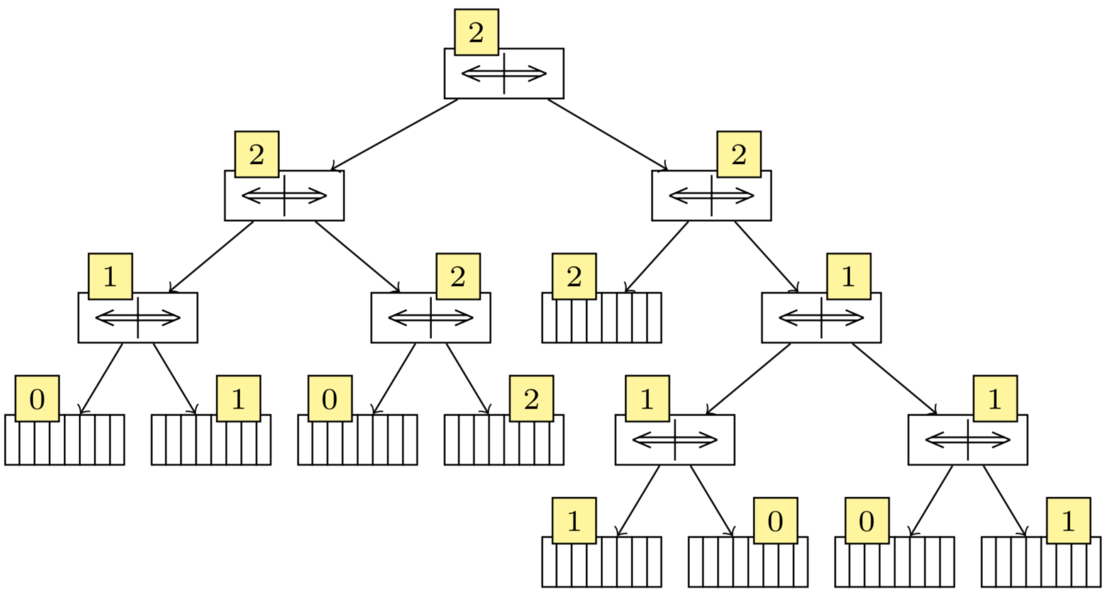
Описанные выше изменения вдохновили на создание нового алгоритма на основе недоминирующего дерева, на которое отсутствие монотонности не производило бы такого сильного замедления. Новый алгоритм был назван ENS-NDT-ONE.

# ГЛАВА 3. ОПИСАНИЕ И РЕАЛИЗАЦИЯ НОВОГО АЛГОРИТМА НЕДОМИНИРУЮЩЕЙ СОРТИРОВКИ ENS-NDT-ONE

В данной главе опишем новый алгоритм ENS-NDT-ONE, который будет лучше подходить на роль одно из алгоритмов, на основе которых создается гибридный алгоритм. Так же в данной главе будет представлена и доказана асимптотическую оценку времени работы.

## 3.1. Описание алгоритма

Алгоритм ENS-NDT был изменен следующим образом: вместо множества деревьев теперь будем хранить одно дерево для всех точек, именно по этой причине, алгоритм назван ENS-NDT-ONE. Параметрами дерева, как в оригинальной версии алгоритма, будет размер корзины, ограничивающий максимальное количество точек, которое может содержаться в одной вершине, и вторым параметром будет глубина, начиная с которой размер корзины игнорируется, и точки перестают делиться на две при превышении первого порога. Так же как в оригинальном алгоритме дерево будет сбалансированным, это обеспечивается предварительно подсчитанной структурой *split*. На рисунке 6 представлено схематическое представление алгоритма ENS-NDT-ONE.




.

Рисунок 6 – Алгоритм ENS-NDT-ONE

Одним из преимуществ производительности алгоритма ENS-NDT было то, что, выполняя определение ранга для точки $p$ в некотором дереве, как только найденная точка доминирующая точку $p$, можно сразу же закрыть это дерево, так как больше точек из этого дерева не может влиять на ранг $p$. Это не так для алгоритма ENS-NDT-ONE, так как в дереве могут встретиться точки с большим или равным раном, чем ранг рассматриваемой точки $p$.

Чтобы предотвратить потери производительности, мы предлагаем хранить максимальный ранг всех точек на поддереве. Это позволит при определении точки с изначальным рангом $k$, не спускаться в поддерево с рангом $\leqslant k$. На фазе добавления точки в дерево необходимо не забыть обновить значения максимального ранга по пути добавления.

Адаптация для функции $HeplerA$ не требуется, так как это обычная недоминирующая сортировка. При обновлении ранга надо только учитывать предварительный ранг точки, и обновлять ранг только в том случае, если он был меньше, чем новый ранг. Напомним, что процедура $HeplerB$ принимает в качестве аргументов два множества, одно с окончательными рангами $L$, другое предварительными рангами $R$. В процедуре $HeplerB$ происходит обновление рангов множества $R$, по рангам множества $L$. Опишем работу алгоритма ENS-NDT-ONE с двумя множествами $L$ и $R$:

а) Обходим точки в лексикографическом порядке, как в оригинальном алгоритме.
   1) Если точка принадлежит множеству с окончательными рангами $L$, добавляем точку в дерево с текущим рангом.
   2) Если точка принадлежит множеству с предварительными рангами $R$, то мы обновляем ранг рассматриваемой точки по дереву и не добавляем ее в структуру, так как ранжирование происходить только на основе точек из множества $L$.

### 3.2. Асимптотика времени работы

Вероятность пропуска поддеревьев сильно влияет на производительность алгоритма ENS-NDT-ONE. В частности, для многих видов входных точек можно показать постоянную верхнюю границу $\alpha$ на вероятность входа в дочерний узел, который соответствует более высокому значению по критерию, рассматриваемому на данном слое. Из этого получаем верхнюю границу $O(MN^{\log_2(1+\alpha)})$ на один запрос и $O(MN^{1+\log_2(1+\alpha)})$ для всей сортировки, что строго быстрее, чем $\Theta(N^2M)$, когда $\alpha < 1$. В худшем случае алгоритм ENS-NDT-ONE работает за $O(MN^2)$, однако зачастую время работы алгоритма сильно лучше. Например, на случайно сгенерированных точках в гиперкубе $[0;1]^M$, $O(N)$ точек с вероятностью не более $1/2$ необходимо заходить в обоих детей в каждой не листовой вершине дерева. Таким образом получаем, что верхняя граница асимптотики времени работы равна $O(MN^{1+\log_2(1+1/2)}) \approx O(MN^{1.585})$.

### 3.3. Реализация алгоритма ENS-NDT-ONE

В данном разделе будет описана реализация алгоритма ENS-NDT-ONE.

Вдохновившись идеями алгоритма ENS-NDT, был создан алгоритм ENS-NDT-ONE. Основным изменением алгоритма ENS-NDT стало то, что вместо множества деревьев для каждого ранга, теперь в структуре одно дерево. На листинге 6 приведен псевдокод основного метода этого алгоритма, который принимает в качестве аргументов множество точек $P$, $M$ — размерность и $B$ — порог, максимальное количество точек в вершине. Для получения $split$ структуры используется функция $CreateSplits$, о которой можно почитать в первой главе данной работы.

Помимо этого, для ускорения процесса определения ранга, будем хранить максимальный ранг на поддереве, что позволит добавить следующую эвристику: если у поддерева максимальный ранг меньше ранга рассматриваемой точки, то это поддерево никак не может повлиять на ранг данной точки.

В функцию $FindRankInNDTreeOne$ будет добавлен дополнительный аргумент, ранг точки $P_i$, тогда функция $FindRankInNDTreeOne$ в терминальной вершине будет иметь реализацию представленную на листинге 8. А в нетерминальной вершине реализация представляет из себя два рекурсивных вызова на вершинах-потомках с одним только отсечением, если координата рассматриваемой вершины больше либо равна медианному значению, то в левого ребенка можно не заходить, то есть в поддерево, где по текущей координате все точки больше рассматри-

Листинг 6 – Главная процедура алгоритма ENS-NDT-ONE.

**procedure** ENS-NDT-ONE(P, M, B)
    $P \leftarrow Sort(P, a^M \prec b^M, ..., a^1 \prec b^1)$
    $S \leftarrow CreateSplits(P, M-1, B)$
    $\mathcal{F} \leftarrow \{\{P_1\}\}$
    $\mathcal{T} \leftarrow newNDTreeOne(S, B)$
    $InsertIntoNDTreeOne(\mathcal{T}, P_1)$
    $j \leftarrow 1$
    **for** $i = 2, ..., |P|$ **do**
        **if** $P_{i-1} \neq P_i$ **then**
            $j \leftarrow FindRankInNDTreeOne(\mathcal{T}, P_i)$
            $InsertIntoNDTreeOne(\mathcal{T}, \mathcal{P}_i)$
        **end if**
        $\mathcal{F}_j \leftarrow \mathcal{F}_j \cup P_i$
    **end for**
    **return** $\mathcal{F}$
**end procedure**

ваемой, не найдется ни одной точки, которая бы доминировала нашу, следовательно заходить в такое поддерево нет смысла. На листинге 7 представлен псевдокод определения ранга в нетерминальной вершине.

Листинг 7 – Процедура поиска ранга точки с предварительным рангом в нетерминальной вершине.

1: **procedure** FindRankInNDTreeOne($\mathcal{T}, p, r$)
2:     **if** $maxRank < r$ **then**
3:         **return** r
4:     **end if**
5:     **if** $p[\mathcal{T}.splitCoordinate] >= \mathcal{T}.splitValue$ **then**
6:         $r \leftarrow FindRankInNDTreeOne(\mathcal{T}.worseNode, P_i, r)$
7:     **end if**
8:     $r \leftarrow FindRankInNDTreeOne(\mathcal{T}.betterNode, P_i, r)$
9:     **return** $r$
10: **end procedure**

Важно отметить, что если ранг на поддереве $< r$, то есть ранга рассматриваемой точки, то текущее поддерево не может повлиять на ранг точки, для которой происходит обновления ранга. В таком случае в строке 3 листингов 8 и 7 происходит выход из процедур.

Листинг 8 – Процедура поиска ранга точки с предварительным рангом в терминальной вершине.

**procedure** FindRankInNDTreeOne($\mathcal{T}, p, r$)<br>
    **if** $maxRank < r$ **then**<br>
        **return** r<br>
    **end if**<br>
    **for** $i = |\mathcal{T}.points|, ..., 1$ **do**<br>
        **if** $\mathcal{T}.points[i] \prec p$ **then**<br>
            **return** $\mathcal{T}.ranks[i] + 1$<br>
        **end if**<br>
    **end for**<br>
    **return** $r$<br>
**end procedure**

# ГЛАВА 4. ОПИСАНИЕ ГИБРИДНОГО АЛГОРИТМА И ЭМПИРИЧЕСКИЙ АНАЛИЗ ВРЕМЕНИ ЕГО РАБОТЫ

В данной главе описан гибридный алгоритм, приведен асимптотический анализ времени его работы, описана реализация гибридного алгоиртма, а также приведен эмпирический анализ времени работы реализованного алгоритма в сравнении с уже существующими.

## 4.1. Гибридный алгоритм

### 4.1.1. Формулировка гибридного алгоритма

Теперь мы можем сформулировать гибридный алгоритм. Мы используем алгоритм “разделяй и властвуй” как базовый алгоритм, каждый раз прежде чем вызывать процедуры $HelperA$ и $HelperB$, мы проверяем размер точек, на которых необходимо выполнить сортировку. Если точки с таким размером эффективно сортируются в алгоритме ENS-NDT-ONE, переключается с базового алгоритма на алгоритм ENS-NDT-ONE для решения этой подзадачи. Так как алгоритм приспособлен к гибридизации, в том числе невосприимчив к отсутствию монотонности, и учитывает предпоставленные ранги в базовом алгоритме, полученный алгоритм будет работать корректно.

Если говорить более формально, мы определяем для каждого значения размерности пороговое значение, которое означает, что каждая подзадача с таким и меньшим размером точек должна быть делегирована алгоритму ENS-NDT-ONE. Для процедуры $HelperA$ размер множества точек — это размер множества $S$, для процедуры $HelperB$ — это сумма размеров множеств $L$ и $R$.

### 4.1.2. Анализ времени работы гибридного алгоритма

В этом разделе дадим некоторую оценку асимптотики времени работы гибридного алгоритма.

Напомним, что время работы алгоритма Буздалова асимптотически равно $O(N(\log N)^{M-1})$. Так как мы определяем пороговые значения как константы, асимптотическая оценка времени работы итогового гибридного алгоритма по-прежнему равна $O(N(\log N)^{M-1})$. Заметим, что тщательный выбор пороговых значений, например, добавить зависимость от вида входных данных, может улучшить асимптотическую оценку. Так как эта задача является сложной, мы оставляем это для возможной будущей работы.

## 4.2. Реализация гибридного алгоритма

В данном разделе будут описаны подробности реализации гибридного алгоритма.

Так как при создании алгоритма ENS-NDT-ONE, мы учитывали, что точки могут иметь предпоставленные ранги, адаптировать алгоритм для гибридизации не составляет труда

### 4.2.1. HelperA

Адаптация для функции $HeplerA$ полностью совпадает с самим алгоритмом ENS-NDT-ONE, учитывая только то, что точки имеют на изначальный ранг. То есть помимо множества точек в функцию приходят ранги точек. Обновление ранга рассматриваемой точки происходит, только если изначальный ранг был меньше нового.

### 4.2.2. HelperB

Для функции $HeplerB$ немного сложнее, представим псевдокод основного метода алгоритма ENS-NDT-ONE на листинге 9. Множество точек $L$ уже окончательно проранжированы в базовом алгоритма "Разделяй и властвуй", множество точек $R$ имеют некоторые предварительно предпоставленные ранги. Задача метода обновить ранги точек множества $R$ на основе рангов точек множества $L$.

Первым интересным моментом является то, что $split$ структуру мы будем строить только для множества точек $L$, то есть точки из множества $R$ никак не влияют друг на друга и обновляются только на основе рангов точек $L$. Так же добавлять в структуру мы будет только точки из множества $L$, а точки из $R$ мы будем только ранжировать. Так как точки приходят из базового алгоритма в отсортированном порядке дополнительно делать лексикографическую сортировку нет необходимости. Таким образом мы перебираем объединение точек $L$ и $R$ в лексикографическом порядке. Далее в зависимости от вида точки либо обновляем ей ранг, либо добавляем точку в структуру для последующего ранжирования других точек.

Листинг 9 – Главная процедура алгоритма ENS-NDT-ONE, адаптированная для переключения в момент $HeplerB$.

**procedure** ENS-NDT-ONE-HelperB($L, R, M, B, Ranks$)
  $S \leftarrow CreateSplits(L, M-1, B)$
  $\mathcal{T} \leftarrow newNDTreeOne(S, B)$
  $InsertIntoNDTreeOne(\mathcal{T}, P_1)$
  $j \leftarrow 1$
  **for** $p \in L \cup R$ **do**
    $r \leftarrow p.rank$
    **if** $p \in L$ **then**
      $InsertIntoNDTreeOne(\mathcal{T}, p, r)$
    **end if**
    **if** $p \in R$ **then**
      $r \leftarrow FindRankInNDTreeOne(\mathcal{T}, p, r)$
    **end if**
  **end for**
  **return** $\mathcal{R}\dashv\backslash\|\int$
**end procedure**

Таким образом, мы реализовали гибридный алгоритм на основе алгоритма Буздалова и алгоритма Густавссона.

### 4.3. Настройка параметров гибридного алгоритма

Настройка гибридного алгоритма будет представлять некоторый диапазон размеров множеств точек для каждой размерности, при котором происходит переключение. Параметры основаны на экспериментальных данных и не зависят от размера множеств точек на которых изначально запускается гибридный алгоритм недоминирующей сортировки. Другими словами, эти параметры можно считать константами.

Наше экспериментальное исследование показало, что для размерности три оптимальным переключением будет, когда размер множества точек не превышает $100$, а при размерностях больше трех переключение необходимо осуществлять на множествах точек размером не более $20000$.

### 4.4. Сравнение с существующими алгоритмами на искусственно сгенерированных тестовых данных

В данном разделе приводится сравнение эффективности работы нового гибридного алгоритма с существующими. Сравнение производилось с двумя родительскими алгоритмами, которые в свою очередь являются лучшими алгоритмами недоминирующей сортировки на сегодняшний день, и с алгоритмом END-NDT-ONE работающим самостоятельно.

Замеры времени работы производились на множестве точек размером до $10^6$ с размерностями 3, 5, 7, 10, 15, на случайно сгенерированных независимых точках в гиперкубе $[0;1]^M$ и на точках расположенных на одной гиперплоскости и имеющих один ранг. Результаты приведены в таблице 1, серым обозначены лучшие в каждой группе алгоритм.

Таблица 1 – Среднее время работы алгоритмов в секундах. Лучшее время в каждой категории обозначено серым цветом.

| $N$ | $M$ | Divide&Conquer | | ENS-NDT | | ENS-NDT-ONE | | Hybrid | |
|---|---|---|---|---|---|---|---|---|---|
| | | hypercube | hyperplane | hypercube | hyperplane | hypercube | hyperplane | hypercube | hyperplane |
| $5 \cdot 10^5$ | 3 | 1.52 | 0.85 | 1.95 | 0.73 | 1.66 | 0.76 | 1.17 | 0.67 |
| $10^6$ | 3 | 2.82 | 1.60 | 5.25 | 1.61 | 4.25 | 1.65 | 2.63 | 1.50 |
| $5 \cdot 10^5$ | 5 | 22.7 | 16.6 | 8.31 | 2.01 | 6.25 | 2.22 | 6.43 | 4.68 |
| $10^6$ | 5 | 45.2 | 33.0 | 26.3 | 5.22 | 18.2 | 5.82 | 17.2 | 12.8 |
| $5 \cdot 10^5$ | 7 | 89.6 | 55.1 | 17.1 | 6.96 | 15.5 | 6.78 | 9.29 | 7.02 |
| $10^6$ | 7 | 191.5 | 120.2 | 55.4 | 19.4 | 46.1 | 18.9 | 26.8 | 20.1 |
| $5 \cdot 10^5$ | 10 | 197.7 | 99.9 | 27.6 | 15.9 | 36.7 | 17.7 | 14.5 | 11.5 |
| $10^6$ | 10 | 478.8 | 228.6 | 84.8 | 48.1 | 104.8 | 55.0 | 41.0 | 33.0 |
| $5 \cdot 10^5$ | 15 | 190.0 | 116.1 | 40.8 | 23.0 | 62.1 | 25.9 | 22.6 | 15.7 |
| $10^6$ | 15 | 587.9 | 337.5 | 135.4 | 76.3 | 206.8 | 85.4 | 64.5 | 46.0 |

Для каждой конфигурации ввода было создано 10 экземпляров с разными случайно сгенерированными точками. Мы измерили общее время во всех этих случаях и разделили их на 10, чтобы получить среднее время выполнения. Измерения времени выполнялись с использованием пакета Java Microbenchmark Harness с одной итерацией прогрева не менее 6 секунд, чего было достаточно для стабилизации работы программы. Был использован высокопроизводительный сервер с процессорами AMD OpteronTM 6380 и 512 GB ОЗУ, а код был запущен с виртуальной машиной OpenJDK 1.8.0 141. Репозиторий с кодом представлен на GitHub [1], также там можно найти графики времени работы. В таблице 1 показаны только средние результаты для $N = 5 \cdot 10^5$ и $10^6$. Видно, что ги-

[1]https://github.com/mbuzdalov/non-dominated-sorting/releases/tag/v0.1

бридный алгоритм выигрывает во всех случаях, кроме $M = 5$ и $M = 7$ на гиперплоскости. Еще одно интересное наблюдение, что ENS-NDT-ONE работает быстрее, чем ENS-NDT, на экземплярах гиперкуба с $M \leqslant 7$, что означает, что предложенная эвристика действительно эффективна. Однако константа реализации ENS-NDT-ONE немного больше.

### 4.5. Сравнение времени работы алгоритмов на худшем случае

Так как время работы алгоритма Густавссона ENS-NDT и нового алгоритма ENS-NDT-ONE имеют квадратичную асимптотику $O(MN^2)$ в некотором худшем случаем, мы провели сравнение времени работы этих алгоритмов на таком виде данных с получившимся гибридным алгоритмом.

Мы провели исследование на множестве точек размером до $10^5$ и разных размерностях: 3, 5, 7, 10, 15. Результаты сравнения времени работы на худшем случае для алгоритмов ENS-NDT и ENS-NDT-ONE приведены в таблице 2.

Таблица 2 – Среднее время работы алгоритмов на худшем виде данных для алгоритмов ENS-NDT и ENS-NDT-ONE в секундах.

| $N$ | $M$ | Divide&Conquer worst case | ENS-NDT worst case | ENS-NDT-ONE worst case | Hybrid worst case |
|---|---|---|---|---|---|
| $10^5$ | 3 | 0.04 | 0.32 | 0.38 | 0.04 |
| $10^5$ | 5 | 0.05 | 4.6 | 7.5 | 0.7 |
| $10^5$ | 7 | 0.05 | 45.5 | 38.3 | 1.58 |
| $10^5$ | 10 | 0.06 | 69.1 | 81.8 | 3.37 |
| $10^5$ | 15 | 0.08 | 175.3 | 185.3 | 4.45 |

Эмпирический анализ времени работы показал, что данные, которые только за квадратичное время сортируются алгоритмами ENS-NDT и ENS-NDT-ONE, нашим гибридным алгоритмом сортируются несравнимо быстро. Не смотря на то, что гибридный алгоритм проигрывает по времени алгоритму на основе метода “разделяй и властвуй”, это происходит только на специфическом, худшем виде данных, на остальных же наборах точек, как было видно в таблице 2, гибридный алгоритм работает сильно быстрее алгоритма Divide&Conquer. Это означает, что на любых видах данных наш алгоритм работает быстро.

# ГЛАВА 5. МНОГОПОТОЧНЫЙ АЛГОРИТМ НЕДОМИНИРУЮЩЕЙ СОРТИРОВКИ

В данной главе приведено описание многопоточной верисии алгоритма недоминирующей сортировки, описанного в Главе 4.

## 5.1. Описание многопоточного алгоритма

Наше исследование было мотивировано очень важной практической задачей: многокритериальной задачей оптимизации управления топливом, быстрое решение которой необходимо в ходе функционирования ядерного реактора [14]. Необходим быстрый алгоритм для большого количества точек до $10^6$. После успешной реализации алгоритма недоминирующей сортировки появилась идея написать параллельный алгоритм.

Для реализации многопоточного алгоритма использовалась специфика алгоритма Буздалова, который основан на методе “разделяй и властвуй”. В момент рекурсивного запуска на множествах точек, ранги которых только впоследствии повлияют друг на друга, можно выполнить сортировку этих двух множеств независимо, а значит в разных потоках. Важно отметить, что в этот же момент происходит проверка переключения на новый алгоритм, то есть если по параметрам настройки гибридизации необходимо переключить алгоритм, то алгоритм Баздалова переключается на алгоритм Густавссона, иначе алгоритм Буздалова продолжает работать, но в новом потоке.

Мы выявили две стратегиями разделения данных между потоками:

а) Создавать каждый раз новые объекты всех необходимых классов.
б) Переиспользовать ранее созданные классы, учитывая, что их может одновременно использовать несколько потоков.

Во втором случае необходимо заранее договориться, какая часть данные, будет использоваться в какой момент, и обеспечить, чтобы пересекающиеся данные не изменялись в разных потоках в одно время. При выборе второго способа пришлось бы в самом начала выделить память на максимальный возможный размер точек. Однако, на практике нам такого количества памяти не нужно, так как в гибридном алгоритме обычно происходит переключение не на всех точках, а на частях, которые часто сильно меньше общего количества точек.

После проведения теоретического и практического сравнение затрат по памяти этих двух стратегий. Оказалось, что суммарное количество памяти, которое необходимо в первом случае примерно равно затратам по памяти во втором случае.

Также при реализации, используя вторую стратегию, мы столкнулись с проблемой: при больших массивах размером с максимальное количество точек, алгоритм выполняется медленнее, чем аналогичный алгоритм, использующий маленькие массивы, созданные только для текущей итерации. Сравнение происходило на одном потоке. Мы пришли к выводу, что это особенности исполнения, таким образом, остановились на первой стратегии параллелизации.

### 5.2. Анализ времени работы многопоточного алгоритма

После написания корректно работающего алгоритма мы произвели замеры времени работы алгоритма на одном, на двух и на восьми потоках. В результате был реализован гибридный параллельный алгоритм недоминирующей сортировки, который оказался быстрее однопоточной версии до $1.8$ раз на двух потоках и до $3$ раз на восьми потоках. Так как базовый алгоритм невозможно полностью приспособить к многопоточности, то есть большее время сортировка все-таки выполняется в один поток, результат считается успешным.

## ЗАКЛЮЧЕНИЕ

В результате данной работы был предложен новый гибридный алгоритм недоминирующей сортировки, основанный на алгоритме Буздалова и др и алгоритма ENS-NDT-ONE, также разработанного в рамках данной работы. Скорость работы полученного алгоритма превосходит оба родительских алгоритма, которые являются лучшими алгоритмами недоминирующей сортировки на сегодняшний день. Основным его преимуществом оказалась хорошая производительность на очень больших множествах точек. Нам неизвестны публикации результатов с приемлемым временем работы недоминирующей сортировки множеств точек размером $10^6$ большой размерности.

Алгоритм адаптирован для многопоточного выполнения, используя свойство алгоритма “разделяй и властвуй”, ускорение составляет до $1.8$ на двух потоках и до трех раз на восьми потоках.

По результатом данной магистерской работы была принята статья на конференцию PPSN 2018 с названием “Towards Large-Scale Multiobjective Optimization: Extremely Fast Hybrid Non-Dominated Sorting”.

## СПИСОК ЛИТЕРАТУРЫ


1 *Brockhoff D.*, *Wagner T.* GECCO 2016 tutorial on evolutionary multiobjective optimization. // Proceedings of Genetic and Evolutionary Computation Conference Companion. — 2016. — С. 201–227.

2 *Zhang Q.*, *Li H.* MOEA/D: A multiobjective evolutionary algorithm based on decomposition. // IEEE Transactions on Evolutionary Computation. — 2007. — Т. 11, № 6. — С. 712–731.

3 *Zitzler E.*, *Kunzli S.* Indicator-based selection in multiobjective search. // Parallel Problem Solving from Nature – PPSN VIII. — 2004. — № 3242. — С. 832–842.

4 *Corne D.*, *Jerram N.*, *Knowles J.* PESA-II: Region-based selection in evolutionary multiobjective optimization. // Proceedings of Genetic and Evolutionary Computation Conference. — 2001. — С. 283–290.

5 *Deb K.*, *Pratap A.*, *Agarwal S.* A fast and elitist multi-objective genetic algorithm: NSGA-II. // IEEE Transactions on Evolutionary Computation. — 2002. — Т. 6, № 2. — С. 182–197.

6 *Deb K.*, *Jain H.* An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. // IEEE Transactions on Evolutionary Computation. — 2013. — Т. 18, № 4. — С. 577–601.

7 *Zitzler E.*, *Laumanns M.*, *Thiele L.* SPEA2: Improving the strength pareto evo- lutionary algorithm for multiobjective optimization. // Proceedings of the EURO-GEN’2001 Conference. — 2001. — С. 91–100.

8 *Coello Coello C.*, *Toscano Pulido G.* A micro-genetic algorithm for multiob- jective optimization. // Proceedings of International Conference on Evolutionary Multi-Criterion Optimization. — 2001. — № 1993. — С. 126–140.

9 *Knowles J.*, *Corne D.* Approximating the nondominated front using the Pareto archived evolution strategy. // Evolutionary Computation. — 2000. — Т. 8, № 2. — С. 149–172.

10 *Fonseca C.*, *Fleming P.* Nonlinear system identification with multiobjective genetic algorithm. // Proceedings of the World Congress of the International Fed- eration of Automatic Control. — 1996. — C. 187–192.

11 *Jensen M.* Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. // IEEE Transactions on Evolutionary Computation. — 2003. — T. 7, № 5. — C. 503–515.

12 *Gustavsson P.*, *Syberfeldt A.* A new algorithm using the non-dominated tree to improve non-dominated sorting. // Evolutionary Computation. — 2018. — T. 26, № 1. — C. 89–116.

13 *Srinivas N.*, *Deb K.* Multiobjective optimization using nondominated sorting in genetic algorithms. // Evolutionary Computation. — 1994. — T. 2, № 3. — C. 221–248.

14 *Schlunz E.* Multiobjective in-core fuel management optimisation for nuclear research reactors. // Ph.D. thesis, Stellenbosch University. — 2016.

15 *Kung H.*, *Luccio F.*, *Preparata F.* On finding the maxima of a set of vectors. // Journal of ACM. — 1975. — T. 22, № 4. — C. 469–476.

16 *Zhang X.*, *Tian Y.*, *Cheng R.* An Efficient Approach to Nondominated Sorting for Evolutionary Multiobjective Optimization. // IEEE Transactions on Evolutionary Computation. — 2015. — C. 201–213.

17 *Fortin F.*, *Grenier S.*, *Parizeau M.* Generalizing the improved run-time com- plexity algorithm for non-dominated sorting. // Proceedings of Genetic and Evo- lutionary Computation Conference. — 2013. — C. 615–622.

18 *Buzdalov M.* A provably asymptotically fast version of the generalized Jensen algorithm for non-dominated sorting. // Parallel Problem Solving from Nature – PPSN XIII. — 2014. — № 8672. — C. 528–537.

19 *Cormen T.*, *Leiserson C.*, *Rivest R.* Introduction to Algorithms, 2nd Ed. // MIT Press, Cambridge, Massachusetts. — 2001.

20 *Roy P., Islam M., Deb K.* Best Order Sort: A new algorithm to non- dominated sorting for evolutionary multi-objective optimization. — 2016.

21 *Blum M., Floyd R. W., Pratt V.* Time bounds for selection. // Journal of Computer and System Sciences. — 1973. — T. 7, № 4. — C. 448–461.

22 *Markina M., Buzdalov M.* Hybridizing non-dominated sorting algorithms: Divide-and-conquer meets Best Order Sort. // Proceedings of Genetic and Evolutionary Computation Conference Companion. — 2017. — C. 153–154.