ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Множество известных и широко распространенных многокритериальных эволюционных алгоритмов используют процедуру недоминирующей сортировки, или процедуру определения множества недоминирующих решений, которая может быть сведена к недоминирующей сортировке. Примерами таких алгоритмов могут послужить NSGA-II [1], PESA [2], PESA-II [3], SPEA2 [4], PAES [5], PDE [6] и многие другие алгоритмы. Вычислительная сложность одной итерации этих алгоритмов часто определяется сложностью процедуры недоминирующей сортировки, следовательно, снижение сложности последней делает такие многокритериальные эволюционные алгоритмы значительно быстрее, чем с использованием медленной недоминирующей сортировки.

Существуют разные алгоритмы недоминирующей сортировки, но эффективность их работы очень сильно отличается в зависимости от данных. Этим можно воспользоваться и совместить идеи разных алгоритмов в одном, чтобы получить новый алгоритм, сочетающий в себе преимущества существующих, избавившись при этом от их недостатков.

Цель данной работы – сделать гибридный алгоритм недоминирующей сортировки, который будет использовать наиболее подходящий алгоритм или переключаться между алгоритмами в ходе своей работы.

В Главе 1 данной работы представлен общий обзор работы. В разделе 1.1 подробно рассмотрены определение недоминирующей сортировки и необходимые для этого понятия, а также представлены примеры применения недоминирующей сортировки, подтверждающие актуальность данной работы. В разделе 1.2 произведен обзор имеющихся результатов и подробно описаны лучшие из них. В разделе 1.3 описаны недостатки существующих алгоритмов. В разделе 1.4 сформулирована постановка задачи.

В Главе 2 представлены теоретические исследования по гибридизации алгоритмов недоминирующей сортировки. В разделе 2.1 произведен анализ существующих алгоритмов и их сравнение. В разделе 2.2 описана предлагаемая схема гибридизации. В разделе 2.3 представлены рассуждения по поводу асимптотики гибридного алгоритма.

В Главе 3 представлены практические исследования и их результаты. В разделе 3.1 представлена адаптация алгоритмов для гибридизации. В разделе 3.2 описан гибридный алгоритм. В разделе 3.2 представлена многопоточная версия гибридного алгоритма. В разделе 3.4 описан подход, который мы использовали для настройки получившегося алгоритма. В разделе 3.5 представлены результаты сравнения получившегося алгоритма с существующими алгоритмами.

В заключении подведены итоги работы, а также сказано, какие могут быть дальнейшие пути развития гибридных алгоритмов недоминирующей сортировки.

ГЛАВА 1. ОБЗОР РАБОТЫ

В этой главе представлен общий обзор работы: уточнены цели и объяснены термины и понятия, присутствующие в решении задачи. Также произведен обзор имеющихся результатов и сформулирована постановка задачи.

1.1. Недоминирующая сортировка

В данном разделе представлено определение недоминирующей сортировки и необходимые для ее понимания понятия. Также рассмотрены недостатки рассмотренных алгоритмов, и обоснована необходимость их резрешения.

1.1.1. Определение

Недоминирующая сортировка — это процедура, которая ранжирует множество точек в многомерном пространстве R^n. Если описывать неформально, то ее задача определить, какие точки "лучше"других. При этом допускается, что две точки могут быть одинаково "хорошими". На рисунке 1 подставлен пример недоминирующей сортировки конкретных точек.

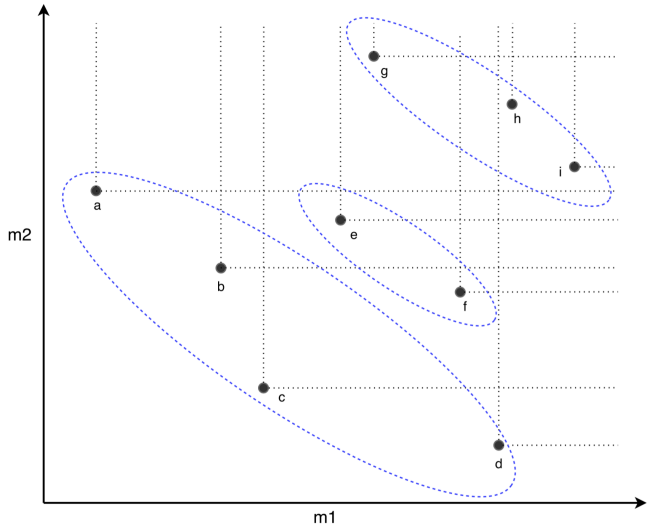


Рисунок 1 – На рисунке представлен пример недоминирующей сортировки для точек {a,b,c,d,e,f,g,h,i}. Точки {a,b,c,d} имеют ранг 0, {e,f} — 2, {g,h,i} — 3.

Для того, чтобы сформулировать определение недоминирующей сортировки, сначала надо определить, какие точки мы считаем "луч-

ше"других. Для этого введем определение доминирования одной точки другой.

Определение. В M-мерном пространстве, точка $A = (a_1, ..., a_M)$ доминирует точку $B = (b_1, ..., b_M)$ тогда и только тогда, когда для всех $1 \leqslant i \leqslant M$ выполняется неравенство $a_i \leqslant b_i$, и существует такое j, что $a_j < b_j$.

"Лучшими"в данном контексте будут считаться точки, которые не доминируются ни одной другой точкой или, другими словами, лежащие на Парето-фронте. Однако часто бывают не только точки с Парето-фронта, но и другие "хорошие"точки. Таким образом мы приходим к определению процедуры недоминирующей сортировки.

Определение. Недоминирующая сортировка множества точек S в M-мерном пространстве — это процедура, которая назначает всем точкам из S ранг. Все точки, которые не доминируются ни одной точкой из S имеют ранг ноль. Точка имеет ранг $i + 1$, если максимальный ранг среди доминирующих её точек равен i.

1.1.2. Применение и актуальность

Самый яркий пример применения процедуры недоминирующей сортировки — алгоритмы многокритериальной оптимизации, особенно эволюционные алгоритмы. Последние на каждой итерации генерируют множество потенциальных решений и оценивают каждое решение по всем критериям. Если критериев M, то получается набор из N M-мерных векторов, где N — число потенциальных решений. И эволюционному алгоритму на каждой итерации надо отобрать лучшие решения, для чего и требуется недоминирующая сортировка.

Если каждый критерий для каждого потенциального решения считается достаточно долго, то время выполнения недоминирующей сортировки становится неважным, так как асимптотика каждой итерации алгоритма зависит в основном от асимптотики времени подсчета критериев. Однако гораздо чаще встречаются задачи, в которых подсчет каждого критерия занимает время значительно меньшее, чем время, необходимое для недоминирующей сортировки. Именно в таких случаях ускорение алгоритмов недоминирующей сортировки ускорит время

выполнения итерации алгоритма, а следовательно и время выполнения всего алгоритма.

В настоящее время существует много алгоритмов недоминирующей сортировки, но каждый из них имеет свои слабые стороны. Это означает, что существует возможность сделать алгоритм, который мог бы заранее предсказывать, время работы какого алгоритма на данном наборе точек будет меньше, и выбирать оптимальный. Более того, есть возможность совместить идеи разных алгоритмов в одном, который всегда будет работать не хуже существующих. Такие возможности делают проблему ускорения алгоритмов недоминирующей сортировки еще более актуальной.

1.2. Анализ существующих алгоритмов

В данном разделе будут рассмотрена история развития алгоритмов недоминирующей сортировки. Особое внимание будет уделено самым эффективным алгоритмам, которые применяются для гибридизации в данной работе. Они будут рассмотрены наиболее подробно.

1.2.1. Наивные алгоритмы

Опишем самый наивный алгоритм недоминирующей сортировки. Он перебирает все пары точек и сравнивает их по всем критериям. После этого он присваивает нулевой ранг тем из них, которые не доминируются ни одной другой точкой и отбрасывает их из множества. Данная процедура повторяется, пока в множестве остаются точки. Причем на каждом новом шаге присваивается новое значение ранга, на единицу больше, чем на предыдущем шаге. Рассмотрим время работы данного наивного алгоритма. Пусть N — это число точек, а M — размерность пространства. Тогда сравнение всех пар точек по M критериям займет $O(MN^2)$, а всего шагов алгоритма будет не больше, чем максимальное число рангов — N. Таким образом, время работы данного алгоритма не превышает $O(MN^3)$, причем эта оценка достигается в худшем случае при максимальном числе рангов в сортируемом множестве.

В работе Кунга и др. [7] предлагается алгоритм определения множества недоминируемых точек, при этом его вычислительная сложность составляет $O(Nlog^{M-1}N)$. Этот алгоритм можно использовать для выполнения недоминирующей сортировки аналогично вышеописанному

алгоритму. Сначала в множестве S алгоритм Кунга находит множество точек с рангом 0. Затем алгоритм Кунга запускается на оставшемся множестве точек, и получившемуся множеству точек присваивается ранг 1. Процесс выполняется до тех пор, пока имеются точки, которым не присвоен ранг. Описанная процедура в худшем случае выполняется за $O(N^2 log^{M-1} N)$, если максимальный ранг точки равен $O(N)$.

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

1.2.2. Алгоритм "Разделяй и Властвуй"

Йенсен [8] впервые предложил алгоритм недоминирующей сортировки с вычислительной сложностью $O(Nlog^{M-1} N)$. Однако, как корректность, так и оценка сложности алгоритма доказывалась в предположении, что никакие две точки не имеют совпадающие значения ни в какой размерности. Однако довольно часто алгоритмы оптимизации работают с дискретными критериями, поэтому совпадение разных решений по одному критерию может быть довольно частым событием. Устранить указанный недостаток оказалось достаточно трудной задачей — первой успешной попыткой сделать это, насколько известно исполнителю данной НИР, является работа Фортена и др. [9]. 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 [8], равна $O(N^2 M)$. Наконец, в работе Буздалова и др. [10] предложены модификации алгоритма из работы [8], которые позволили доказать в худшем случае также и оценку $O(Nlog^{M-1} N)$, не нарушая корректности работы алгоритма.

Опишем подробнее алгоритм Буздалова и др., так как он будет использоваться в гибридном алгоритме, разрабатываемом в данной работе. Основная идея алгоритма — принцип "разделяй и властвуй", основанный на разбиении исходного множества на несколько меньших множеств и решении задачи на этих множествах. Алгоритм на каждом шаге находит медиану множества по последнему критерию и делит его на три подмножества элементов, меньших медианы по последнему критерию, больших и равных ей. Далее алгоритм рекурсивно запускается на каж-

дом подмножестве с некоторыми промежуточными вычислениями. На рисунке 2 схематически изображена идея алгоритма на основе метода “разделяй и властвуй”.

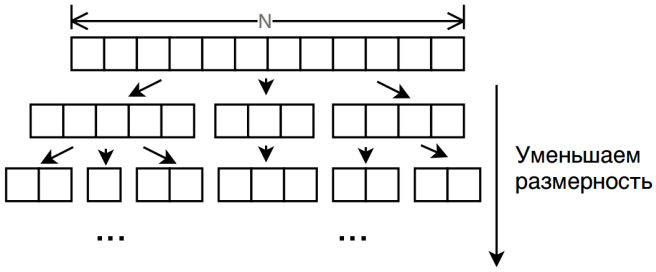


Рисунок 2 – Идея алгоритма на основе метода “разделяй и властвуй”.

Рассмотрим подробнее процедуры, использующиеся в алгоритме Буздалова. Основными из них являются процедуры $NDHelperA$, $NDHelperB$ и $SplitBy$. Первая как раз и представляет собой основной алгоритм на множестве S, которое дается ему на вход. Однако она может также присвоить ранги точкам так, чтобы они не стали меньше, чем ранги, которые эти точки имели до исполнения процедуры. Также данная процедура сравнивает точки только по первым k критериям. Эти дополнения необходимы для возможности корректного рекурсивного вызова данной процедуры. Алгоритм Буздалова, по сути, заключается в том, что расставляет всем точкам множества S ранг ноль, а затем запускает процедуру $NDHelperA$ с аргументами S и M.

Процедура $NDHelperA$ не работает с размерностями меньше 2, так как для них есть более эффективные алгоритмы, которые она и запускает при необходимости.

Псевдокод $NDHelperA$ представлен на листинге 1.

Следующая процедура $NDHelperB$ запускается между рекурсивными запусками $NDHelperA$ на трех подмножествах. Задача этой процедуры — расставить минимально возможные ранги точек для подмножества, на котором сейчас запустится $NDHelperA$. Простая имплементация данной процедуры могла бы перебрать все пары точек из множества точек с уже проставленными рангами и точек, на которых сейчас запустится $NDHelperA$, и проставить каждой точке из второго множества ранг, на единицу большие, чем максимальный ранг точки с уже

Листинг 1 – Процедура NDHelperA. Она присваивает ранги точкам из S по первым k рангам.

```
1: procedure NDHelperA(S, k)
2:     if |S| < 2 then return
3:     else if |S| = 2 then
4:         {s^(1), s^(2)} ← S
5:         if s^(1)_{1:k} ≺ s^(2)_{1:k} then
6:             rank(s^(2)) ← max{rank(s^(2)), rank(s^(1)) + 1}
7:         end if
8:     else if k = 2 then
9:         SweepA(S)
10:    else if |{s_k | s ∈ S}| = 1 then
11:        NDHelperA(S, k − 1)
12:    else
13:        L, M, H ← SplitBy(S, median{s_k | s ∈ S}, k)
14:        NDHelperA(L, k)
15:        NDHelperB(L, M, k − 1)
16:        NDHelperA(M, k − 1)
17:        NDHelperB(L ∪ M, H, k − 1)
18:        NDHelperA(H, k)
19:    end if
20: end procedure
```

проставленным рангом, которая ее доминирует. Однако это работало бы квадратичное время по размеру подмножеств, поэтому данная процедура также использует принцип “разделяй и властвуй” и при расстановке минимальных рангов также разбивает множества на более мелкие и запускается на них рекурсивно.

Более подробно процедура $NDHelperB$ описана в псевдокоде на листинге 2.

Листинг 2 – Процедура NDHelperB. Она подгоняет ранги точек из H используя первые k критериев, сравнивая их с точками из L.

```
1:  procedure NDHelperB(L, H, k)
2:      if L = {} or H = {} then return
3:      else if |L| = 1 or |H| = 1 then
4:          for all h ∈ H, l ∈ L do
5:              if l_{1:k} ⪯ h_{1:k} then
6:                  rank(h) ← max{rank(h), rank(l) + 1}
7:              end if
8:          end for
9:      else if k = 2 then
10:         SweepB(L, H)
11:     else if max{l_k | l ∈ L} ⩽ min{h_k | h ∈ H} then
12:         NDHelperB(L, H, k − 1)
13:     else if min{l_k | l ∈ L} ⩽ max{h_k | h ∈ H} then
14:         m ← median{s_k | s ∈ L ∪ H}
15:         L_1, M_1, H_1 ← SplitBy(L, m, k)
16:         L_2, M_2, H_2 ← SplitBy(H, m, k)
17:         NDHelperB(L_1, L_2, k)
18:         NDHelperB(L_1, M_2, k − 1)
19:         NDHelperB(M_1, M_2, k − 1)
20:         NDHelperB(L_1 ∪ M_1, H_2, k − 1)
21:         NDHelperB(H_1, H_2, k)
22:     end if
23: end procedure
```

Последняя процедура, которую стоит упомянуть для описания алгоритма Буздалова и др. — процедура разбиения множеств $SplitBy$. Она работает за линейное время по размеру множества, которое она разбивает и делит его на три множества: точки, большие m по критерию k, равные m и меньшие m. В каждом получившемся подмножестве сохраняется порядок точек, который был в оригинальном множестве.

Процедура $SplitBy$ описана на листинге 3.

Листинг 3 – Процедура разбиения. В каждом подмножестве используется такой же порядок точек, как и до разбиения.

```
1: procedure SplitBy(S, m, k)
2:     L ← {s ∈ S|s_k < m}
3:     M ← {s ∈ S|s_k = m}
4:     H ← {s ∈ S|s_k > m}
5:     return L, M, H
6: end procedure
```

1.2.3. Алгоритм Роя и др.

Большой интерес представляет алгоритм Роя $Best\,Order\,Sort\,(BOS)$ [11], который в отличии вышеупомянутых не использует метод разделяй и властвуй. Его вычислительная сложность $O(MNlogM + MN^2)$. В лучшем случае алгоритм работает за $O(MNlogM)$, что лучше алгоритма предложенного Буздаловым и др. Однако в худшем случае его асимптотика совсем другая — $O(MN^2)$. Авторами алгоритма не было проведено более точных исследований по его времени работы.

1.2.4. Алгоритм Густавссона и др.

Большой интерес представляет алгоритм Густавссона и Сиберфильдта ENS-NDT [12], который в отличии вышеупомянутых не использует метод разделяй и властвуй. 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, модификации алгоритма и для понимания итогового гибридного алгоритма.

Алгоритм Густавссона и Сиберфильдта ENS-NDT относится к группе Efficient Non-dominated Sort. 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.

Недоминирующее дерево (NDTree) основано на Bucket k-d дереве. Bucket 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. Новые вершины ассоциированы 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 3 подставлена иллюстрация bucket 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, bucket size представленной на рисунке структуры равен трем. Также, чтобы избежать создания слишком большого количества уровней, NDTree имеет параметр максимальной глубины. Это означает, что если достигнута максимальная глубина, то параметры максимального количества точек в вершине ингорируется.

NDTree использует преподсчитанное split множество, где все возможные медианные значения преподсчитаны до того, как они потребуются. Это отличается от стандартной реализации bucket k-d дерева, в котором обычно значение считается в момент переполнения максимального допустимого количества точек в вершине. Это становится возможно сделать, потому что мы знаем множество точек для сортировки заранее. Основное преимущество использования split дерева заключается в том, что девево остается сбалансированным, что значительно улучшает производительность поиска. Split множество удобно хранить в виде дерева, похожем на финальное недоминирующее дерево.

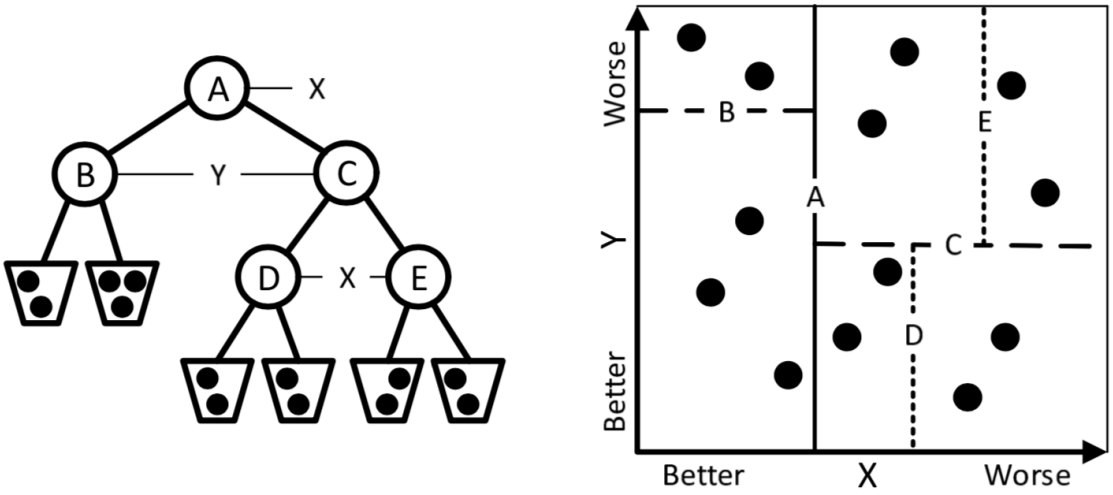
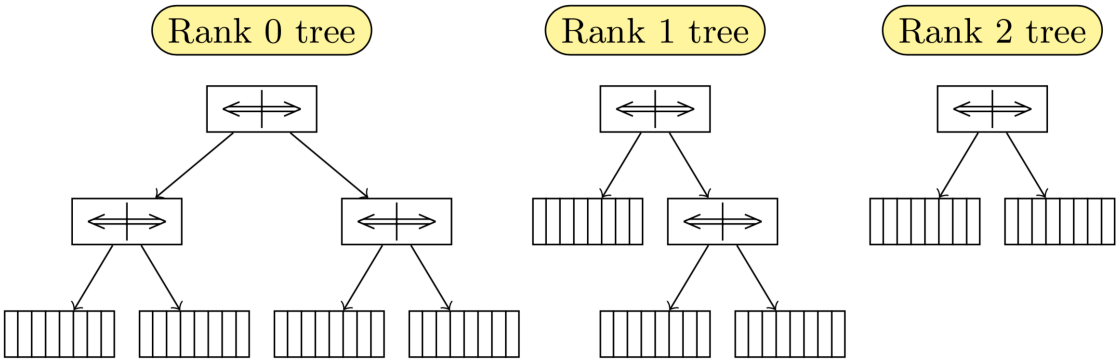


Рисунок 3 – На рисунке представлена иллюстрация внутреннего устройства структуры Bucket k-d tree. Слева изображена структура, справа точки на плоскости, по которым получена структура.

Для выполнения недоминирующей сортировки следует поддерживать отдельное дерево для каждого ранга. На рисунке 4 представлено схематическое представление структуры использующейся в недоминирующей сортировке.



.

Рисунок 4 – 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) Перебираем точки в лексикографическом порядке.

1) Определяем ранг.
2) Добавляем в соответствующее рангу дерево.

Ниже представлен псевдокод на листинге 4 основного метода недоминирующей сортировки, который принимает в качестве аргументов множество точек P, M - размерность и B - порог, то есть максимальное количество точек в вершине. Для получения split структуры используется функция CreateSplits, которая на вход получает множество точек, порог и размерность M-1, размерность M-ая игнорируется, так как ранее множество точек было лексикографически отсортировано. Лексикографическая сортировка выбрана еще потому, что она позвоняет уменьшить число сравнений.

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 из-за лексикографической сортировки. Главный цикл на строке 8 определяет ранги точек и добавляет их в структуру.

Возможная реализация CreateSplits изображена на листинге 5. 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. [**Blum**].

Вкаратце опишем процедуру CreateSplit, TODO

Поледней интересной для нас функцией являтся функция определения ранга точки FrontIndexBinarySearch, представленная на листинге 6. Эта процедура бинарным поиском определяет минимальное древо в структуре $\mathcal{T}$, где ни одна точка не доминировала бы рассматриваемую.

Подробное описание алгоритма ENS-NDT необходимо для понимания дальнейшей модификации 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 хо-

Листинг 4 – Главная процедура алгоритма ENS-NDT.

```
1: procedure ENS-NDT(P, M, B)
2:     S ← CreateSplits(P, M − 1, B)
3:     P ← Sort(P, a^M ≺ b^M, ..., a^1 ≺ b^1)
4:     F ← {{P_1}}
5:     T ← {newNDTree(S, B)}
6:     InsertIntoNDTree(T_1, P_1)
7:     j ← 1
8:     for i = 2, ..., |P| do
9:         if P_{i−1} ≠ P_i then
10:            j ← FrontIndexBinarySearch(T, P_i)
11:            if j > |T| then
12:                F_j ← 0
13:                T_j ← newNDTree(S, B)
14:            end if
15:            InsertIntoNDTree(T_j, P_i, )
16:        end if
17:        F_j ← F_j ∪ P_i
18:    end for
19:    return F
20: end procedure
```

Листинг 5 – Пример реализации процедуры CreatSplit, которая вычисляет медианные значения для NDTree.

```
1: procedure CreateSplit(P, M, B, d ← 0)
2:     o ← 1 + (d mod M)
3:     P ← Sort(P, a^o ≺ b^o)
4:     m ← P_{1+⌊|P|/2⌋}
5:     S ← {newNDSplit(o, m)}
6:     if |P| > B then
7:         Better ← {P_i, i < 1 + ⌊|P|/2⌋}
8:         Worse ← {P_i, i ⩾ 1 + ⌊|P|/2⌋}
9:         S.BetterSplit ← CreateSplit(Better, M, B, d + 1)
10:        S.WorseSplit ← CreateSplit(Worse, M, B, d + 1)
11:    end if
12:    return S
13: end procedure
```

Листинг 6 – Процедура определения ранга точки s.

```
1:  procedure FrontIndexBinarySearch(𝒯, s)
2:      i ← 1
3:      j ← |𝒯| + 1
4:      while i ≠ j do
5:          k ← ⌊i + (j − i)/2⌋
6:          if FromntDominates(𝒯_k, s) then
7:              i ← k + 1
8:          else
9:              j ← k
10:         end if
11:     end while
12:     return i
13: end procedure
```

рошую 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. ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ

В данной главе будут представлены основные результаты работы. Сначала будут рассмотрены основные кандидаты для гибридизации. Далее опишем идею гибридизации и проведем теоретический анализ времени работы.

2.1. Анализ существующих алгоритмов

В качестве основного кандидата для гибридизации был выбран алгоритм “разделяй и властвуй” Буздалова и др. Причин для этого две:

а) Данный алгоритм работает лучше большинства алгоритмов. Благодаря этому гибридный алгоритм тоже будет работать эффективно.

б) Данный алгоритм рекурсивно запускает себя в процессе своей работы. Это порождает точки возможного подключения на другие алгоритмы.

Вторым кандидатом стал алгоритм Роя, который уже показал неплохие результаты в гибриде с алгортмом Буздалова [13], но алгоритм Роя был только наполовину приспособлен для гибридизации. В этой работе представлена попытка приспособить его полностью.

Третьим кандидатом стал алгоритм Густавссона, на этот раз гибридизация оказалась удачной. Для начала мы адаптировали алгоритм END-NDT. Получившийся алгоритм мы назвали ENS-NDT-ONE, поскольку вместо множество деревьев было заменено единственным деревом для всех точек. Затем реализовали гибридный алгоритм на основе алгоритма “Разделяй и властвуй” и алгоритма ENS-NDT-ONE.

2.2. Предлагаемая схема гибридизации

В этом разделе будет описана предлагаемая схема гибридизации.

2.2.1. Выбор момент переключения

Алгорим “Разделяй и властвую” очень хорошо подходит для гибридизации, так как в алгоритме рекурсивно вызываются подзадачи меньшего размера и меньшей размерности. В первой главе данной работы описан алгоритм подробно. Мы предлагаем следующую идею гибридизации:

а) Запускаем алгоритм Divide and 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.

В оригинальной статье функции, где мы предполагаем переключаться на другой алгоритм, названы $HeplerA$ и $HeplerB$.

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.

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 $HeplperA$ ни что иное, как 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 адоптации для гибридизации:

а) Обходим точки в порядке предложенном в оригинальном алгоритме.
   1) Если точка принадлежит множеству с окончательными рангами L, добавляем точеку в структуру алгоритма с текущим рангом.
   2) Если точка принадлежит множеству с предварительными рангами, 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 $HelperA$ [13], 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, k ранга, 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 работа 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 канддата для гибридизации, потому что он является самым эффективным алгоритмом на сегодняшний день в общем случае.

В оригинальном алгоритме определение ранга происходило похожим на алгоритм Роя образом, то есть бинарным поиском по деревьям, где каждое дерево соответстует своему рангу. Осуществлялся поиск дерева соответствующего минимальному рангу, где ни одна точка не доминирует рассматриваемую, тогда рассматриваемой точке присуждался ранг этого дерева. Но аналогично проблеме описанной выше, в алгорит-

ме Роя, инвариант позволяющий осуществить бинарный поиск перестает работать.

Для решения этой проблемы мы адаптировали алгоритм ENS-NDT следующим образом: вместо множества деревьев теперь будем хранить одно дерево для всех точек. Параметрами дерева, как в оригинальной версии алгоритма, будет порог, ограничивающий максимальное количество точек, которое может содержаться в одной вершине, и вторым параметром будет глубина, начиная с которой первый порог игнорируется, и точки перестают делиться на две при превышении первого порога. Так же как в оригинальном алгоритме дерево будет сбалансированным, это обеспечивается предварительно посчитанной структурой split.

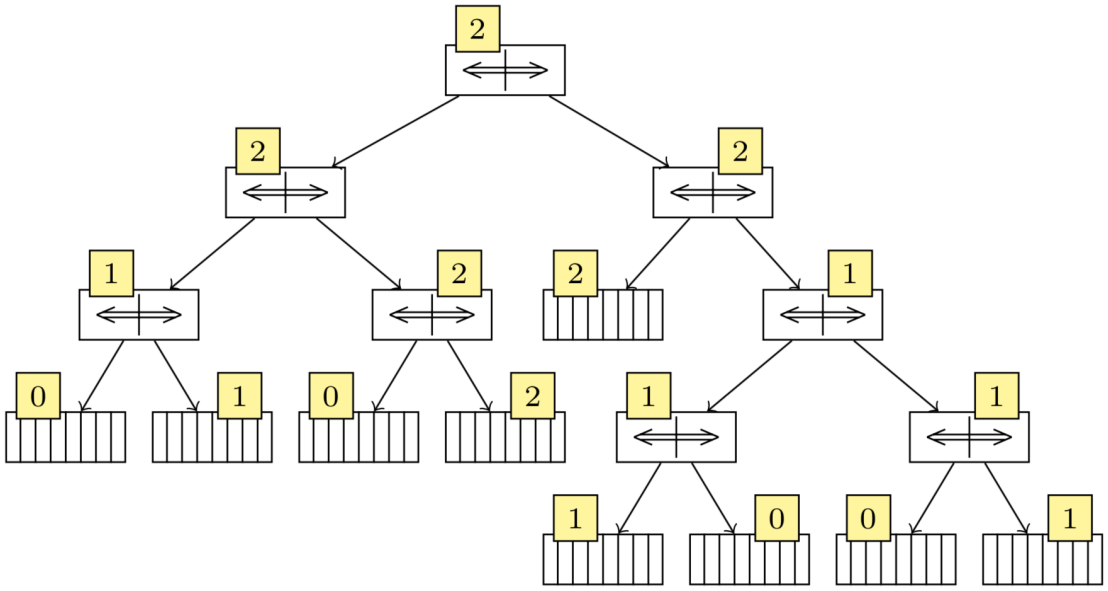
Одним из преимуществ производительности алгоритма ENS-NDT является то, что, выполняя определение ранга для точки p, как только найденная точка в дереве доминирующая точку p, можно сразу же закрыть это дерево, так как больше точек из этого дерева не может влиять на ранг p. Это не так для алгоритма ENS-NDT-ONE, так как в дереве могут встретиться точки с большим или равным раном, чем обновленный 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. Для $HeplerB$ была предложена следующая идея адоптации для гибридизации:

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

Адаптированный алгоритм сам по себе представляет некоторый интерес, поэтому мы ему присвоили название ENS-NDT-ONE. На рисунке 5 представлено схематическое представление алгоритма ENS-NDT-ONE.



.

Рисунок 5 – Алгоритм ENS-NDT-ONE

В худшем случае алгоритм работает за $O(MN^2)$, аналогично оригинальному алгоритму ENS-NDT. Однако зачастую время работы алгоритма сильно лучше. Например, на случайно сгенерированных точках в гиперкубе $[0;1]^M$, $O(N)$ точек с вероятностью более $1/2$ необходимо заходить в обоих детей в каждой не листовой вершине дерева. Таким образом получаем, верхняя граница асимптотики времени работы равна $O(MN^{1+log_2(3/2)}) \approx O(MN^{1.585})$.

2.4. Анализ времени работы гибридного алгоритма

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

We can now formulate the hybrid algorithm. We take the divide-and-conquer algorithm as a basis, however, before we enter the main parts of HelperA or HelperB, we check whether the subproblem is small enough. If it is, we use the ENS-NDT-ONE algorithm to solve this subproblem. Since ENS-NDT-ONE is immune to the features of these subproblems, such as the loss

of monotonicity, the resulting algorithm will always produce correct results. More formally, we define, for every number of objectives, a threshold which signifies that every subproblem with this number of objectives and the size below the threshold should be delegated to ENS-NDT-ONE. For HelperA, the size of the problem is the size of the set P, while for HelperB this is the sum of sizes of the sets L and R. We shall note that, since we define thresholds to be constants, the asymp- totic estimation of the running time of this algorithm is still $O(N(\log N)^{M-1})$. However, we note that more careful choices for thresholds, that possibly depend on the number of objectives or on other properties of the subproblems, may possibly result in smaller runtime bounds. Due to the complexity of this issue, including heavy input dependency, we leave this for possible future work.

ГЛАВА 3. РЕАЛИЗАЦИЯ И ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

В данной главе рассмотрены подробности реализации гибридного алгоритма, а также результаты экспериментов по сравнению реализованного алгоритма с уже существующими.

3.1. Реализация гибридного алгоритма

Листинг 7 – Главная процедура алгоритма ENS-NDT-ONE.

```
1:  procedure ENS-NDT-ONE(P, M, B)
2:      S ← CreateSplits(P, M − 1, B)
3:      P ← Sort(P, a^M ≺ b^M, ..., a^1 ≺ b^1)
4:      F ← {{P_1}}
5:      T ← newNDTreeOne(S, B)
6:      InsertIntoNDTreeOne(T, P_1)
7:      j ← 1
8:      for i = 2, ..., |P| do
9:          if P_{i−1} ≠ P_i then
10:             j ← FindRankInNDTreeOne(T, P_i)
11:             InsertIntoNDTreeOne(T, P_i)
12:         end if
13:         F_j ← F_j ∪ P_i
14:     end for
15:     return F
16: end procedure
```

В данном разделе будут описаны подробности реализации гибридного алгоритма.

3.1.1. Адаптация алгоритма Густавссона

Основным изменением алгоритма ENS-NDT стало то, что вместо множества деревьев для каждого ранга, теперь в структуре одно дерево. Новый алгоритм назван ENS-NDT-ONE, на листинге 7 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.

Для адаптации в функцию $FindRankInNDTreeOne$ будет добавлен дополнительный аргумент, ранг точки P_i, тогда функция

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

```
1: procedure FindRankInNDTreeOne(𝒯, p, r)
2:     if maxRank < r then
3:         return r
4:     end if
5:     if p[𝒯.splitCoordinate] >= 𝒯.splitValue then
6:         r ← FindRankInNDTreeOne(𝒯.worseNode, P_i, r)
7:     end if
8:     r ← FindRankInNDTreeOne(𝒯.betterNode, P_i, r)
9:     return r
10: end procedure
```

Листинг 9 – Процедура поиска ранга точки с предварительным рангом в терминальной вершине.

```
1: procedure FindRankInNDTreeOne(𝒯, p, r)
2:     if maxRank < r then
3:         return r
4:     end if
5:     for i = |𝒯.points|, ..., 1 do
6:         if 𝒯.points[i] ≺ p then
7:             return 𝒯.ranks[i] + 1
8:         end if
9:     end for
10:    return r
11: end procedure
```

$FindRankInNDTreeOne$ в терминальной вершине будет иметь реализацию представленную на листинге 9. 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 рассматриваемой, не найдется ни одной точки, которая бы доминировала нашу, следовательно заходить в такое поддерво нет смысла. На листинге 8 представлен псевдокод определения ранга в нетерминальной вершине.

3.1.1.1. 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.

3.1.1.2. HelperB

Для функции $HeplerB$ немного сложнее, представим псевдокод основного метода алгоритма ENS-NDT-ONE на листинге 10. Множество точек L уже окончательно отранжированы в базовом алгоритма “Разделяй и властвуй”, множество точек R имеют некоторые предварительно проставленные 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.

Мы реализовали гибридный алгоритм на основе алгоритма Буздалова и алгоритма Густавссона.

Листинг 10 – Главная процедура алгоритма ENS-NDT-ONE, адаптированная для переключения в момент $HeplerB$.

```
1:  procedure ENS-NDT-ONE-HelperB(L, R, M, B, Ranks)
2:      S ← CreateSplits(L, M − 1, B)
3:      T ← newNDTreeOne(S, B)
4:      InsertIntoNDTreeOne(T, P_1)
5:      j ← 1
6:      for p ∈ L ∪ R do
7:          r ← p.rank
8:          if p ∈ L then
9:              InsertIntoNDTreeOne(T, p, r)
10:         end if
11:         if p ∈ R then
12:             r ← FindRankInNDTreeOne(T, p, r)
13:         end if
14:     end for
15:     return R⊣\‖∫
16: end procedure
```

3.2. 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 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 привышает 100, а при размерностях больше трех переключение необходимо осуществлять на множествах точек размером не более 20000.

3.3. 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 недомиринующей сортировки на се-

годняшний день, и с адаптацией алгоритма END-NDT-ONE работающей 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. Серым обозначены лучшие в каждой группе алгоритмы.

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 временя 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, 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 гибридный алгоритм выигрывает во всех случаях, кроме $M = 5$ и $M = 7$ на гиперплоскоти. 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.

3.4. Адаптация для многопоточного выполнения

Алгоритм адаптирован для многопоточного выполнения, ускорение составляет до 1.8 на двух потоках и до трех раз на восьми потоках.

3.5. Сравнение с существующими алгоритмами на практических задачах

В данном разделе будут приведены сравнение работы гибридного алгоритма на реальных практических задачах.

ЗАКЛЮЧЕНИЕ

В результате данной работы был предложен новый гибридный алгоритм недоминирующей сортировки, сочетающий в себе достоинства двух лучших известных алгоритмов недоминирующей сортировки. 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 сортировки множеств точек размером 10^6 большой размерности с приемлемым временем работы.

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 этой работы была подготовлена для публикации статья на конференцию на PPSN 2018, и мы ждем рецензий.

СПИСОК ЛИТЕРАТУРЫ

1 *Deb K.* A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II // Transactions on Evolutionary Computation. — 2000. — Т. 6. — С. 182–197.

2 *Corne D.* The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization // Parallel Problem Solving from Nature Parallel Problem Solving from Nature VI. — 2000. — С. 839–848.

3 *Corne D.* PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization // Proceedings of Genetic and Evolutionary Com- putation Conference. — 2001. — С. 283–290.

4 *Zitzler E.* SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization // Proceedings of the EUROGEN'2001 Conference. — 2001. — С. 91–100.

5 *Knowles J.* Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy // Evolutionary Computation. — 2000. — Т. 8 no. 2. — С. 149–172.

6 *Abbass H.* PDE: A Pareto Frontier Differential Evolution Approach for Multiobjective Optimization Problems // Proceed- ings of the Congress on Evolutionary Computation. — 2001. — С. 971–978.

7 *Kung H.* On Finding the Maxima of a Set of Vectors // Journal of ACM. — 1975. — Т. 22 no. 4. — С. 469–476.

8 *Jensen M.* Reducing the Run-time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms // Transactions on Evolutionary Computation. — 2003. — Т. 7 no. 5. — С. 503–515.

9 *Fortin F.* Generalizing the Improved Run-time Complexity Algorithm for Non-dominated Sorting // Proceeding of Genetic and Evolutionary Computation Conference. — 2013. — С. 615–622.

10 *Buzdalov M.* A Provably Asymptotically Fast Version of the Gen- eralized Jensen Algorithm for Non-Dominated Sorting // International Conference on Parallel Problem Solving from Nature. — 2014. — С. 528–537.

11 *Roy P. M. Islam K. D.* Best Order Sort: A New Algorithm to Non-dominated Sorting for Evolutionary Multi-objective Optimization. — 2016.

12 *Gustavsson P. S. A.* A New Algorithm Using the Non-dominated Tree to improve Non-dominated Sorting. — 2017.

13 *M. M.*, *M B.* Hybridizing Non-dominated Sorting Algorithms: Divide-and-Conquer Meets Best Order Sort // CoRR. — 2017. — T. abs/1704.04205.