Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Якупов Илья Юрьевич

# Разработка структуры данных для выполнения инкрементальной недоминирующей сортировки

Научный руководитель: к.т.н. А. С. Станкевич

Санкт-Петербург
2015

# Содержание

# Введение

*Оптимизацией* в математике называется задача нахождения экстремума целевой (оптимизируемой) функции в некоторой области конечномерного векторного пространства.

Существует два вида оптимизационных задач: [1]

1. Структурная оптимизация - задача выбора оптимальной структуры объекта (функции);
2. Параметрическая оптимизация - задача выбора оптимальных числовых значений параметров функции при заданной структуре.

Стандартная математическая задача оптимизации звучит следующим образом. Среди элементов $x$, образующих множество $X$, найти такой элемент $x*$, при котором некоторая функция $f(X)$ достигает минимального (либо максимального) значения $f(x*)$.

*Многокритериальной оптимизацией* называется процесс одновременной оптимизации нескольких конфликтующих целевых функций в некоторой области определения. Задача многокритериальной оптимизации состоит в поиске вектора целевых переменных, удовлетворяющего наложенным ограничениям и оптимизирующего векторную функцию, элементы которой соответствуют целевым функциям. [2] Фактически, задачей многокритериальной оптимизации является одновременное достижение несколькими функциями значений, приемлемых для экспериментатора. Критерии оптимальности выбираются в зависимости от поставленной задачи.

В данной работе в качестве критерия оптимальности рассматривается *эффективность по Парето*. Эффективность по Парето — такое состояние системы, при котором значение каждого частного показателя, характеризующего систему (например, каждой из оптимизируемых функций), не может быть улучшено без ухудшения по крайней мере одного друго-

го показателя (функции). Множество состояний системы, оптимальных по Парето, называется «множеством Парето», или же множеством «неулучшаемых» альтернатив. [3]

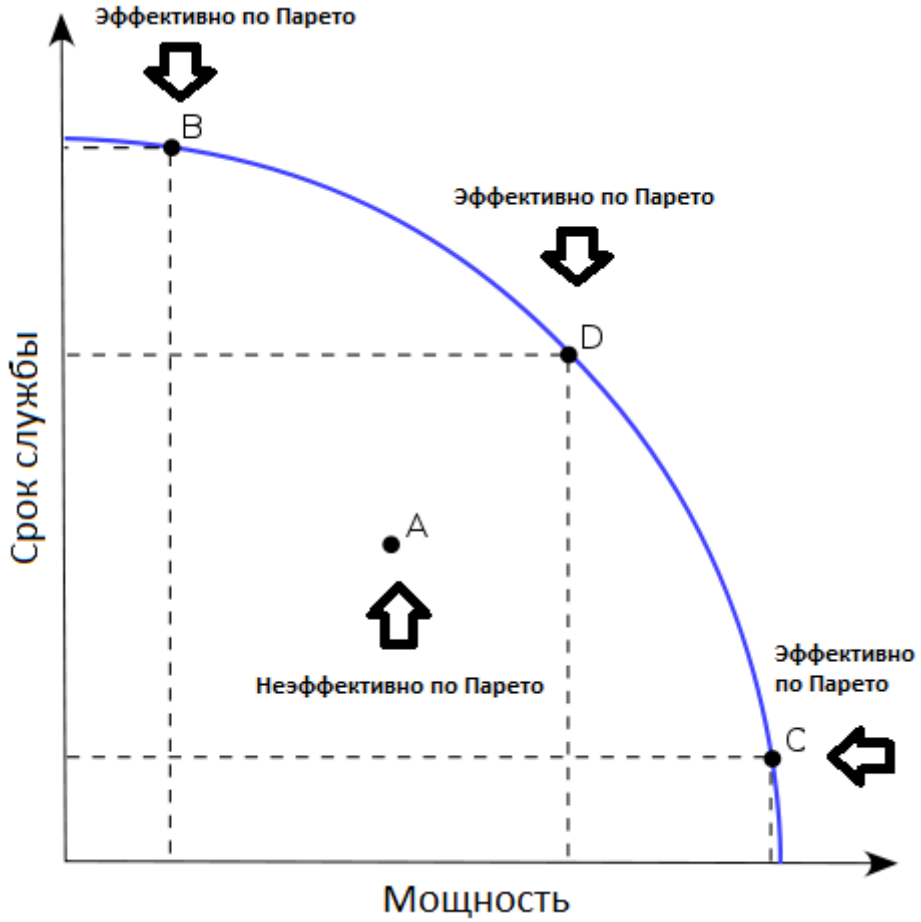


Рис. 0.1: Пример кривой Парето

При решении задачи многокритериальной оптимизации, как правило, используется процедура *недоминирующей сортировки.* [4]

Говорят, что в K-мерном пространстве точка $A = (a_1, ..., a_K)$ доминирует над точкой $B = (b_1, ..., b_K)$, когда для $1 \leq i \leq K$ выполняется неравенство $a_i \leq b_i$ и существует $j$, такое что $a_j < b_j$.

Недоминирующей сортировкой точек в K-мерном пространстве называется процедура, в результате которой:

- Точки, над которыми не доминирует ни одна точка, помечаются *рангом* 0;
- Точки, над которыми доминирует хотя бы одна точка ранга 0, по-

мечаются рангом 1;

- Точки, над которыми доминирует хотя бы одна точка ранга i–1, помечаются рангом i.

В результате выполнения недоминирующей сортировки исходное множество точек разбивается на *фронты Парето*. Фронтом Парето (или *уровнем недоминирования*) называется множество точек, являющихся Парето-оптимальными друг относительно друга. Каждый уровень недоминирования характеризуется рангом точек, которые в нем содержатся. Фронт Парето нулевого ранга (в случае, если недоминируемые точки помечаются рангом 1 - первого ранга) является множеством точек, оптимальных по Парето.

Целью данной работы является разработка структуры данных, поддерживающей корректную структуру уровней недоминирования при добавлении новой точки в двумерном пространстве. В рамках данной работы предложен алгоритм, позволяющий добавлять или удалять точку за линейное ($O(N)$, где $N$ - суммарное количество точек на всех уровнях недоминирования) время. Использование данной структуры данных позволяет, в частности, эффективно реализовать инкрементальный алгоритм многокритериальной оптимизации *NSGA-II*. [5].

# Глава 1. Обзор предметной области

## 1.1. Эволюционные алгоритмы

### 1.1.1. Основные принципы

*Эволюционные алгоритмы* применяются для решения задач оптимизации и используют механизмы, основанные на принципах естественной эволюции.[6]

Эволюционные алгоритмы работают с множеством *особей*-кандидатов на оптимальное решение. Как правило, особи представляются в виде точек в $K$-мерном пространстве, где $K$ - количество оптимизируемых критериев особи. Целью работы эволюционного алгоритма является нахождение наиболее приспособленной особи.

Функция, позволяющая оценить приспособленность особи, называется *функцией приспособленности*. В задачах многокритериальной оптимизации требуется одновременно рассматривать несколько функций приспособленности.

Каждая итерация *классического* эволюционного алгоритма (например, NSGA-II) работает с *поколением* особей. Особи следующего поколения генерируются путем применения к особям текущего поколения операторов скрещивания, мутации и отбора. В результате работы каждой итерации эволюционного алгоритма формируется множество особей следующего поколения, которое будет оптимизироваться следующей итерацией эволюционного алгоритма. Как правило, размер поколения фиксирован, поэтому в конце каждой итерации наименее приспособленные особи отбрасываются.

В *инкрементальных* (англ. *steady-state*) алгоритмах понятие поколения является менее актуальным по причине того, что новые особи генерируются независимо (возможно, в разных потоках). Классический эволюционный алгоритм можно сделать инкрементальным, задав размер популяции, равный единице.

Эволюционный алгоритм завершает работу, как только достигнуто хотя бы одно из условий останова. Как правило, в качестве критериев останова используются либо число итераций, либо значения функций приспособленности.

### 1.1.2. Популярные реализации

#### 1.1.2.1. NSGA

Алгоритм *NSGA (Nondominated Sorting Genetic Algorithm)* является одним из первых алгоритмов многокритериальной оптимизации. [4]

Данный алгоритм на каждой итерации выполняет недоминирующую сортировку за $O(KN^3)$. Помимо значений функции приспособленности, в качестве критерия перехода особи в следующую популяцию используется *фенотипическая дистанция*: если дистанция до какой-либо особи из следующего поколения меньше, чем задано параметром, то текущая особь в следующее поколение не добавляется. [7]

У данного алгоритма есть следующие недостатки: [4]

1. Высокая вычислительная сложность недоминирующей сортировки;
2. Необходимость подбора параметра, определяющего минимально допустимое фенотипическое расстояние между особями;
3. Отсутствие элитизма.

#### 1.1.2.2. Классический NSGA-II

Алгоритм NSGA-II призван исправить основные недостатки алгоритма NSGA: [4]

1. Сортировка выполняется за $O(KN^2)$;
2. При формировании нового поколения используются лучшие особи как из модифицированной (т. е. полученной в результате мутации и скрещивания особей текущего поколения) популяции, так и из исходной;

3. При формировании нового поколения из комбинированной популяции отбрасываются особи с наибольшим рангом. Если в популяции остается только часть особей с некоторым рангом, выбираются особи с наибольшей *crowding distance* - средней длиной ребра кубоида, образованного соседними особями (для крайних особей этот показатель равен $\infty$).

Несмотря на это, у данного алгоритма есть недостаток, свойственный всем классическим (т. е. оперирующим поколениями особей) алгоритмам - невозможность эффективного распараллеливания. Перед тем, как пересчитать ранг особи, алгоритм должен завершить расчет функций приспособленности всех особей. Решить данную проблему призваны *инкрементальные (steady-state)* алгоритмы. [5]

#### 1.1.2.3. SPEA-II

Алгоритм SPEA-II, хотя и позволяет достичь схожих с NSGA-II результатов, не использует процедуру недоминирующей сортировки (равно как и ранг точки) в том виде, в каком она используется в NSGA и NSGA-II и рассматривается в данной работе.

Для точки определены следующие величины: [8]

1. *Сила* - число точек, над которыми текущая точка доминирует;
2. *Исходная жизнеспособность* (англ. *raw fitness*) - сумма сил всех точек, которые доминируют над текущей;
3. *Плотность* - величина, определяющая расстояние до ближайших точек. Данная величина по модулю меньше единицы.

Особи оцениваются по сумме исходной жизнеспособности и плотности.

#### 1.1.2.4. Инкрементальные алгоритмы

Наиболее очевидным способом реализовать инкрементальные версии алгоритмов NSGA-II и SPEA-II является использование классических

(*population-based*) алгоритмов с модифицированной популяцией размера 1. [9]

Было отмечено, что инкрементальные версии алгоритмов позволяют получить сравнимый результат за меньшее число итераций, ценой в 10-20 раз большего времени работы.

Основной причиной низкой производительности является скорость выполнения полной сортировки множества особей. Лучшие из известных на сегодня алгоритмов позволяют выполнять сортировку за $O(Nlog^{K-1}N)$

Кроме того, такие алгоритмы на время выполнения сортировки блокируют все уровни недоминирования, что не позволяет эффективно выполнять асинхронное добавление точек. Таким образом, эффективное распараллеливание таких алгоритмов является невозможным.

## 1.2. Алгоритмы недоминирующей сортировки

### 1.2.1. Алгоритмы полной сортировки

#### 1.2.1.1. NSGA

Простейшая реализация процедуры недоминирующей сортировки была предложена в рамках алгоритма NSGA. [7]

**Листинг 1** Алгоритм недоминирующей сортировки, входящий в NSGA.

```
1:  procedure NDS_NSGA(P)
2:      i ← 0
3:      while P ≠ ∅ do
4:          for p ∈ P do
5:              fDominated ← false
6:              for p' ∈ P do
7:                  if p' ≺ p then
8:                      fDominated ← true
9:                  end if
10:             end for
11:             if !fDominated then
12:                 P[i] ← P[i] ∪ p
13:                 P ← P \ p
14:             end if
15:         end for
16:         i ← i + 1
17:     end while
18: end procedure
```

Данный алгоритм $O(N)$ раз выполняет поиск всех недоминируемых особей и перемещает их на текущий уровень недоминирования, удаляя их из исходной популяции. Каждый раз поиск недоминируемых особей выполняется за $O(KN^2)$. Итоговое время работы алгоритма составляет $O(KN^3)$ в худшем случае. [4]

#### 1.2.1.2. NSGA-II

В алгоритме NSGA-II был предложен более эффективный алгоритм сортировки, работающий за $O(KN^2)$ в худшем случае.[4]

**Листинг 2** Алгоритм недоминирующей сортировки, входящий в NSGA-II.

```
1:  procedure NDS_NSGA2(P)
2:      for p ∈ P do
3:          S_p ← ∅
4:          n_p ← 0
5:          for p' ∈ P do
6:              if p ≺ p' then
7:                  S_p ← S_p ∪ p'
8:              else if p' ≺ p then
9:                  n_p ← n_p + 1
10:             end if
11:         end for
12:         if n_p = 0 then
13:             p_rank ← 1
14:             F_1 ← F_1 ∪ p
15:         end if
16:     end for
17:     i ← 1
18:     while F_i ≠ ∅ do
19:         Q ← ∅
20:         for p ∈ F_i do
21:             for q ∈ S_p do
22:                 n_q ← n_q − 1
23:                 if n_q = 0 then
24:                     p_rank ← i + 1
25:                     Q ← Q ∪ q
26:                 end if
27:             end for
28:         end for
29:         i ← i + 1
30:         F_i ← Q
31:     end while
32: end procedure
```

Первая часть алгоритма (работающая за $O(KN^2)$) находит все недоминируемые точки способом, аналогичным используемому в алгоритме NSGA. Для каждой точки сохраняется множество точек, над которыми

она доминирует, а также число точек, доминирующих над ней.

На основании этих данных, вторая часть алгоритма распределяет точки (для которых число доминирующех точек, еще не распределенных по уровням недоминирования, равно нулю) по уровням недоминирования за $O(KN^2)$).

Итоговое время работы алгоритма - $O(KN^2)$) в худшем случае.

#### 1.2.1.3. Быстрые алгоритмы

Кунгом *с соавт.* (Kung *et al*) [10] был предложен алгоритм, позволяющий выполнять поиск недоминируемых особей за $O(N \log^{K-1} N)$, где K - размерность пространства, и N - размер популяции. Данный алгоритм можно использовать для выполнения недоминирующей сортировки [5], последовательно находя недоминируемые решения и удаляя их их популяции, но итоговое время работы алгоритма составит $O(N^2 \log^{K-1} N)$ в худшем случае.

Дженсеном *с соавт.* (Jensen *et al*) [11] был предложен алгоритм, позволяющий выполнять недоминирующую сортировку за $O(N \log^{K-1} N)$ в худшем случае, исходя из предположения, что для любых двух точек ни одна из координат не может оказаться одинаковой.

Наконец, Буздалов *с соавт.* модифицировали алгоритм Дженсена таким образом, чтобы время работы в худшем случае составило $O(N \log^{K-1} N)$ без каких-либо дополнительных ограничений на значения точек [12].

### 1.2.2. Алгоритм инкрементальной сортировки ENLU

Первый известный автору алгоритм инкрементальной недоминирующей сортировки был предложен Ке Ли *с соавт.* (Ke Li *et al*). [13]

**Листинг 3** Алгоритм недоминирующей сортировки ENLU, основная часть. Выполняется определение ранга добавляемой особи и добавление ее на соответствующий уровень недоминирования.

```
1: procedure ENLU_Main(P, x^c, l) – P - популяция, x^c - добавляемая особь, l - максимальный ранг особей в популяции
2:     fNonDominated ← false
3:     fDominated ← false
4:     fDominates ← false
5:     for i ← 1 to l do
6:         S ← ∅
7:         for j ∈ F_i – F_i – уровень недоминирования i-го ранга do
8:             if x^c ⊀ j and j ⊀ x^c then
9:                 fNonDominated ← true
10:            else if x^c ≺ j then
11:                fDominated ← true
12:                break
13:            else
14:                fDominates ← true
15:                S ← S ∪ j
16:            end if
17:        end for
18:        if fDominated then
19:            continue
20:        else if !fNonDominated and fDominates then
21:            Ранги уровня F_i и выше увеличить на 1.
22:            F_i ← x_c
23:            break
24:        else
25:            F_i ← F_i ∪ x_c
26:            ENLU_Update(S, i + 1) – добавить точки на i-й уровень недоминирования
27:            break
28:        end if
29:    end for
30: end procedure
```

**Листинг 4** Алгоритм недоминирующей сортировки ENLU, перемещение множества особей на следующий уровень недоминирования

```
1: procedure ENLU_Update(S, i, l) – S - множество особей, i - уровень, на который их требуется добавить.
2:     if i = l + 1 then
3:         F_i ← S
4:     else
5:         F_i ← F_i ∪ S
6:         T ← ∅
7:         for i ← 1 to |S| do
8:             for j ← 1 to |F_i| do
9:                 if S(i) ⪯ F_i(j) then
10:                    T ← T ∪ F_i(j)
11:                end if
12:            end for
13:        end for
14:        if T ≠ ∅ then
15:            Update(T, i + 1, l)
16:        end if
17:    end if
18: end procedure
```

Алгоритм состоит из двух частей. В первой части определяется ранг добавляемой особи и находится уровень недоминирования, куда ее следует добавить. Новая особь добавляется на найденный уровень недоминирования, вытеснив с него некоторое (возможно, пустое) множество точек. Затем данное множество переносится на следующий уровень недоминирования, также вытесняя с него некоторое (возможно, пустое) множество точек. Алгоритм завершает работу, когда множество вытесненных точек становится пусто.

Заявленное авторами время работы - $O(KN\sqrt{N})$ в худшем случае, но можно привести пример, когда данный алгоритм работает за $O(KN^2)$. Такое время работы можно получить, если новая особь добавляется на уровень недоминирования нулевого ранга, и на каждой итерации с текущего уровня вытесняются все точки кроме одной. Оценка $O(KN\sqrt{N})$ же верна только для среднего случая.

## 1.3. Выводы по главе 1

В данной главе было приведено определение эволюционных алгоритмов. Описаны основные принципы работы, рассмотрены некоторые наиболее популярные (и, с точки зрения автора, эффективные) реализации.

Рассмотрены два подхода к выполнению недоминирующей сортировки: инкрементальный и основанный на поколениях. Были рассмотрены наиболее популярные алгоритмы. Обнаружено, что лучшие основанные на поколениях алгоритмы могут выполнять полную сортировку за $O(N \log^{K-1} N)$ в худшем случае, тогда как единственный известный автору инкрементальный алгоритм ($ENLU$) работает за $O(KN^2)$ в худшем случае и за $O(KN\sqrt{N})$ - в среднем.

Использование основанных на поколениях алгоритмов для инкрементальной сортировки целесообразно только в отсутствие распараллеливания. Основная проблема в том, что в классических алгоритмах на время сортировки необходимо блокировать все особи, тогда как при инкремен-

тальной сортировке можно блокировать только точки на уровнях недоминирования ранга, большего или равного рангу добавляемой особи.

# Глава 2. Постановка задачи и описание метода ее решения

В данной главе описывается новый алгоритм инкрементальной недоминирующей сортировки. Предложенный метод является развитием идей, заложенных в основу алгоритма $ENLU$, рассмотренного в предыдущей главе.

## 2.1. Требования, предъявляемые к разрабатываемому алгоритму

Целью данной работы является разработка алгоритма инкрементальной недоминирующей сортировки, позволяющего выполнять добавление и удаление особи за линейное ($O(N)$, где $N$ - размер популяции) время.

Требования к данной исследовательской работе:

- Разработка алгоритма инкрементальной недоминирующей сортировки, корректно работающего за линейное время для точек из двумерного пространства;
- Доказательство корректности работы алгоритма;
- Оценка времени работы алгоритма (в т. ч. доказательство линейного времени работы времени работы в худшем случае);
- Сравнение с наилучшими известными алгоритмами недоминирующей сортировки (инкрементальными и базирующимися на поколениях) на различных видах входных данных.

## 2.2. Общее описание алгоритма

В данном разделе приведено описание предложенного алгоритма и используемых в нем структур данных. Структуры данных подробно рассмотрены в секции 2.2.1. Процедура поиска уровня недоминирования, на который требуется добавить точку, описана в секции 2.2.2. Процедура

вставки точки описана в секции 2.2.3. Наконец, процедура удаления худшей точки описана в секции 2.3.1.

При оценке времени работы алгоритма, мы обозначаем как $N$ общее число точек в структуре уровней недоминирования. Текущее число уровней недоминирования обозначается как $M$.

### 2.2.1. Структуры данных

Для получения линейного время работы при вставке и удалении нам потребовалась структура данных, позволяющая выполнять следующие операции за $O(\log N)$:

- Поиск элемента в контейнере;
- Разбиение контейнера на две части по ключу (часть, где все элементы больше ключа, и часть, где все элементы меньше или равны ключу);
- Объединение контейнеров $C_1$ и $C_2$, где каждый элемент из $C_1$ не больше каждого элемента из $C_2$).

Таким требованиям удовлетворяют Декартовы деревья [14] и Splay-деревья [15]. Для Декартова дерева время работы составляет $O(\log N)$ *с высокой вероятностью*, тогда как для Splay-деревьев время $O(\log N)$ *амортизированное*. На практике Декартовы деревья работают несколько быстрее и являются проще в реализации, поэтому они были выбраны для использования в разрабатываемом алгоритмею

Для решения данной задачи была разработана двухуровневая структура данных. Уровни недоминирования хранятся в двоичном дереве поиска, отсортированные по рангу. Каждый узел такого *дерева верхнего уровня* также представляет из себя двоичное дерево поиска *нижнего уровня*, в котором точки (особи) отсортированы по возрастанию первой координаты.

Поскольку для любых двух различных точек $a$ и $b$ либо $a_X > b_X$ и $a_Y \leq b_Y$, либо $a_X < b_X$ и $a_Y \geq b_Y$, все точки дерева нижнего уровня

автоматически оказываются отсортированы в порядке убывания по второй координате.

В дереве верхнего уровня, помимо самого уровня недоминирования, требуется хранить размер поддерева (что позволяет определять ранг уровня за $O(\log M)$) и ссылку на следующий уровень недоминирования (для обеспечения возможности переноса точек на следующий уровень за $O(\log N)$).

Псевдокод для данного "дерева деревьев" представлен в листинге 5. Пример дерева деревьем представлен на рис. 2.1.

**Листинг 5** Псевдокод структур данных, используемых в разработанном алгоритме

```
1: structure Solution
2:     – точка
3:     X – первая координата
4:     Y – вторая координата
5: end structure
6: structure LLTNode
7:     – узел дерева нижнего уровня
8:     L : LLTNode – левое поддерево
9:     R : LLTNode – правое поддерево
10:    V : Solution – ключ
11: end structure
12: structure HLTNode
13:    – узел дерева верхнего уровня
14:    L : HLTNode – левое поддерево
15:    R : HLTNode – правое поддерево
16:    N : HLTNode – ссылка на узел - следующий уровень недоминирования
17:    V : LLTNode – ключ
18:    S : Integer – размер поддерева
19: end structure
```

### 2.2.2. Определение ранга

Имея дерево нижнего уровня $T$ и точку $s$, за $O(\log |T|)$ можно определать, доминирует ли хотя бы одна точка из $T$ над $s$. Для этого нужно найти точку $u \in T$, такую что $u_X \leq s_X$ и $u_X$ наибольшая. Это можно сделать путем обхода дерева $T$ из корня. Если $u$ найдена и доминирует над $s$, значит, доминирующее решение из $T$ найдено. Иначе, ни одна точка из $T$ не доминирует над $s$.

Для доказательства корректности данной операции необходимо рассмотреть два случая. Если $u$ не найдена, то для любой точки $t \in T$ вы-

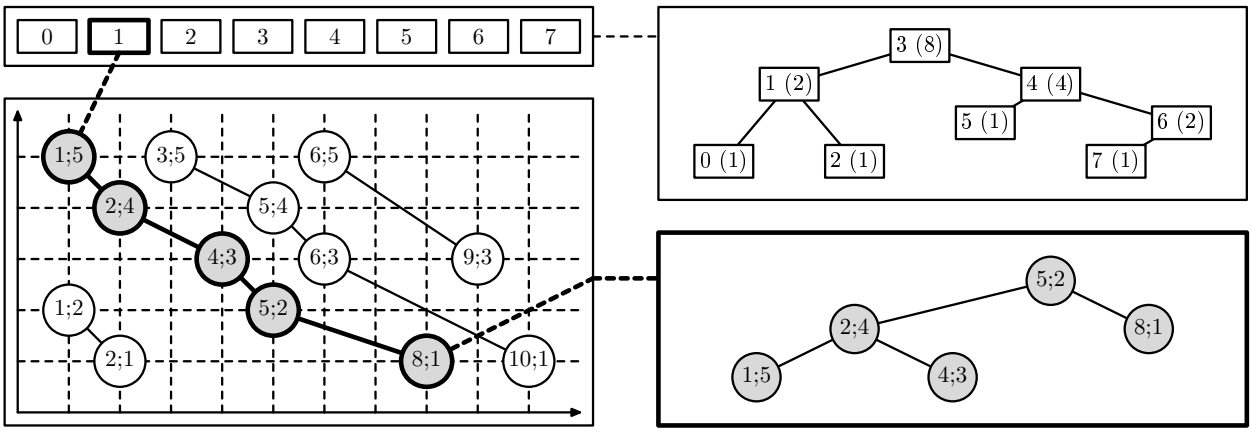


Рис. 2.1: Предложенная структура данных – “дерево деревьев”. Узлы “дерева верхнего уровня” соответствуют уровням недоминирования. Каждый уровень недоминирования представлен в виде “дерева нижнего уровня”, узлы которого упорядочены по первой координате. В каждом узле “дерева верхнего уровня” дополнительно хранится размер поддерева (на рисунке - числа в скобках).

полняется условие $t_X > s_X$, то есть $t$ не доминирует над $s$. Если $u$ найдена, все точки из $T$, отличные от $u$, могут быть разделены на две группы: $V = \{v \,|\, v_X < u_X\}$ и $W = \{w \,|\, w_X > u_X\}$. Для каждой точки $v \in V$ также верно $v_Y > u_Y$. Если $u$ не доминирует над $s$, выполняется $u_Y > s_Y$, потому что $u_X \leq s_X$. Из этого следует, что для каждой точки $v \in V$ верно $v_Y > u_Y > s_Y$, то есть ни одна точка из $V$ не может доминировать над $s$. В то же время, для любой точки $w \in W$ верно $w_X > s_X$ по построению ($u_X$ максимальна, при условии что $u_X \leq s_X$). Таким образом, ни одна точка из $W$ не может доминитовать над $s$.

Алгоритм, приведенный в листинге 6, позволяет выполнять обход дерева верхнего уровня и найти уровень недоминирования минимального ранга, который не доминирует над новой точкой $s$.

**Листинг 6** Псевдокод процедуры определения уровня недоминирования, на который следует добавить точку.

```
1: function LowLevelDominates(T, s)
2:     – доминирует ли хоть одна точка из T над s?
3:     T : LLTNode – корень дерева нижнего уровня
4:     s : Solution – добавляемая точка
5:     B ← null
6:     while T ≠ null do
7:         if T.V.X ≤ s.X then
8:             B ← T
9:             T ← T.R
10:        else
11:            T ← T.L
12:        end if
13:    end while
14:    if B = null then
15:        return false
16:    end if
17:    return B.Y < s.Y or B.Y = s.Y and B.X < s.X
18: end function
19: function SmallestNonDominatingLayer(H, s)
20:    – возвращает уровень доминирования наименьнего ранга, ни одна точка
21:    – из которого не доминирует над s H, а также ранг найденного уровня.
22:    H : HLTNode – tкорень дерева верхнего уровня
23:    s : Solution – добавляемая точка
24:    I ← 0 – счетчик доминирующих уровней
25:    B ← null
26:    while H ≠ null do
27:        if LowLevelDominates(H.V, s) then
28:            I ← I + H.S
29:            H ← H.R
30:            if H ≠ null then
31:                I ← I − H.S
32:            end if
33:        else
34:            B ← H
35:            H ← H.L
36:        end if
37:    end while
38:    return (B, I)
39: end function
```

Грубая оценка времени работы данного алгоритма составляет $O(\log M \log N)$, где $O(\log M)$ - оценка времени обхода дерева верхнего уровня, а $O(\log N)$ - оценка времени обхода каждого дерева нижнего уровня.

Более точная оценка основывается на следующей идее. Есть $k = O(\log M)$ уровней, которые участвовали в процедуре поиска (и по которым был совершен обход). Пусть их размеры составляют $L_1 \dots L_k$, и $L_1 + \dots + L_k \leq N$. Время обхода уровня размера $L_i$ составляет $O(1 + \log L_i)$

(необходимо добавить 1 на случай $\log L_i = o(1)$). Общее время работы процедуры поиска уровня составит:

$$O\left(k + \sum_{i=1}^{k} \log L_i\right).$$

Согласно неравенству Коши, $\sum_{i=1}^{k} \log L_i \leq k \log(N/k)$, что дает следующую оценку времени работы процедуры поиска уровня недоминирования:

$$O\left(\log M \left(1 + \log \frac{N}{\log M}\right)\right),$$

Поскольку $M \leq N$ и $\log(N/\log M)$ является $\omega(1)$, данную оценку можно упростить до:

$$O\left(\log M \log \frac{N}{\log M}\right).$$

При фиксированном $N$ и переменном $M$ данное выражение достигает максимума при $M = \Theta(N)$. Таким образом, время работы составит $O((\log N)^2)$ в худшем случае.

### 2.2.3. Вставка

Имея дерево верхнего уровня $H$ и точку $s$, процедура вставки обновляет $H$, добавляя $s$ в одно из деревьев нижнего уровня.

Ключевая идея, позволяющая быстро выполнять вставку, состоит в том, что множества точек, переходящих на следующий уровень, образуют непрерывные участки на исходном уровне, и остаются непрерывными на новом уровне. Пример такой ситуации представлен на рис. 2.2.

Псевдокод процедуры вставки приведен на листинге 7. В качестве одной из перемещенных используется дерево нижнего уровня, содержащее точки, которые требуется переместить на следующий уровень. Изначально данное дерево содержит одну - добавляемую - точку.

Уровень, на который необходимо добавить точку, находится с помощью процедуры определения ранга, описанной в предыдущей секции.

На каждой итерации предложенного алгоритма выполняются следующие действия:

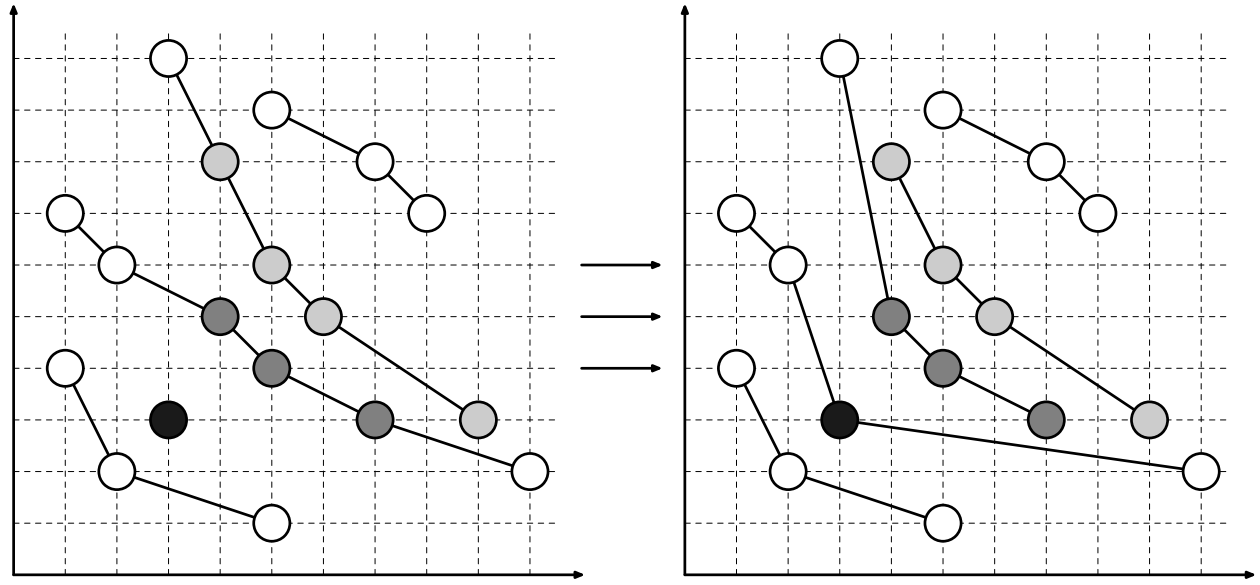

Рис. 2.2: Пример вставки. Точки, которые не переходят на другой уровень изображены белыми. Добавляемая точка - черная. Множества точек, перемещенные на другой уровень, изображены серыми. Необходимо заметить, что некоторые точки (точки наивысшего ранга, изображены белыми) не явно были перемещены на другой уровень недоминирования, поскольку вместо этого был увеличен ранг всего уровня.

- Дерево нижнего уровня текущего ранга делится на три части с помощью текущего множества перемещаемых точек $C$ следующим образом:
  - "левая часть" $T_L$ состоит из всех точек текущего ранга, чья координата $X$ меньше или равна наименьшей координате $X$ точек из $C$;
  - "средняя часть" $T_M$ ссостоит из всех точек текущего ранга, над которыми доминирует хотя бы одна точка из $C$;
  - "правая часть" $T_R$ состоит из всех точек текущего ранга, чья координата $Y$ меньше наименьшего значения координаты $Y$ точек из $C$.

  Корректность такого разделения будет доказана далее в лемме 2.1.
- Текущий уровень собирается из $T_L$, $C$ и $T_R$.
- Если $T_L$ и $T_R$ пусты, значит, весь текущий уровень доминируется точками из $C$. Следовательно, необходимо добавить новый уровень, состоящий из $T_M$, а ранги текущего уровня и всех последующих -

увеличить на единицу. Такая ситуация является условием остановки алгоритма вставки.

- Если $T_M$ пусто, все последующие уровни остаются неизменными. Такая ситуация также является условием остановки алгоритма вставки.
- В противном случае, $C \leftarrow T_M$, и алгоритм переходит на следующую итерацию.

Если после последней итерации множество $C$ остается непусто, оно формирует новый уровень недоминирования ранга, на единицу большего максимальному рангу точек перед вставкой. Данный уровень уставляется в дерево верхнего уровня с максимальным рангом.

**Листинг 7** Псевдокод процедуры вставки в дерево верхнего уровня.

```
1: function SplitX(T, s)
2:     – делит дерево T на две части L, R
3:     – таким образом, что для любой l ∈ L верно l.X ≤ s.X
4:     – и для любой r ∈ R верно r.X > s.X
5:     T : LLTNode
6:     s : Solution
7: end function
8: function SplitY(T, s)
9:     – делит дерево T на две части L, R
10:    – таким образом, что для любой l ∈ L верно l.Y > s.Y
11:    – и для любой r ∈ R верно r.Y ≤ s.Y
12:    T : LLTNode
13:    s : Solution
14: end function
15: function Merge(L, R)
16:    – склеивает деревья L и R при условии что
17:    – для любой l ∈ L и r ∈ R верно l.X < r.X
18:    L : LLTNode
19:    R : LLTNode
20: end function
21: function Insert(H, s)
22:    – вставка точки s в дерево верхнего уровня H
23:    H : HLTNode
24:    s : Solution
25:    C ← new LLTNode(s)
26:    (G, i) ← SmallestNonDominatingLayer(H, s)
27:    while G ≠ null do
28:        C_min ← точка с минимальной x из C
29:        C_max ← точка с минимальной y из C
30:        (T_L, T_i) ← SplitX(G.V, C_min)
31:        (T_M, T_R) ← SplitY(T_i, C_max)
32:        G.V ← Merge(T_L, Merge(C, T_R))
33:        if T_M = null then
34:            return – больше никакие точки перемещать не требуется
35:        end if
36:        if T_L = null and T_R = null then
37:            – весь текущий уровень доминируется
38:            – необходимо вставить все точки на новый уровень
39:            Insert new HLTNode(T_M) after G
40:            return
41:        end if
42:        C ← T_M
43:        G ← G.N
44:    end while
45:    Вставить new HLTNode(C) после последнего узла в H
46: end function
```

## 2.3. Корректность и время работы

Для доказательства корректности, в первую очередь, необходимо доказать следующую лемму.

**Лемма 2.1.** *Пусть есть два множества точек в двумерном пространстве $A$ и $B$, такие что:*

- *никакие две точки из $A$ не доминируют друг над другом;*
- *никакие две точки из $B$ не доминируют друг над другом;*
- *каждая точка из $B$ доминируема хотя бы одной точкой из $A$.*

*Пусть $A' \subseteq A$: $\{a : a \in A, a.X \geq X_A, a.Y \geq Y_A\}$ для некоторых $X_A$ и $Y_A$. Пусть $B' \subseteq B$: $\{b : b \in B, \exists a \in A' : a$ доминирует над $b\}$.*

*Тогда существуют $X_B$ и $Y_B$, такие что $B' = \{b : b \in B, b.X \geq X_B, b.Y \geq Y_B\}$.*

**Доказательство.** Пусть $a_{\min}$ - точка из $A'$ с минимальной координатой $X$. Пусть $a_{\max}$ - точка из $A'$ с минимальной координатой $Y$. Пусть $X'_A = a_{\min}.X$ и $Y'_A = a_{\max}.Y$.

Поскольку никакие две точки из $A'$ не доминируют друг над другом, величина $X_M = a_{\max}.X$ является наибольшей $X$ среди всех точек из $A'$, а величина $Y_M = a_{\min}.Y$ является наибольшей $Y$ для всех точек из $A'$.

Очевидно, для любой $b \in B'$ верно $b.X \geq X'_A$, и $b.Y \geq Y'_A$, поскольку если хотя бы одно из этих условий не выполняется, $b$ не может быть доминируема какой-либо точкой из $A'$. Таким образом, $B' \subseteq \{b : b \in B, b.X \geq X'_A, b.Y \geq Y'_A\}$.

Нам необходимо доказать, что каждая точка $b \in B$, для которой верно $b.X \geq X'_A$ и $b.Y \geq Y'_A$, также принадлежит $B'$. Доказательство проведем от противного.

Пусть существует $b \in B$, такая что $b.X \geq X'_A$ и $b.Y \geq Y'_A$, но $b \notin B'$. По определению, существует $a \in A$, такая что $a$ доминирует над $b$. По определению $B'$, $a \notin A'$. Таким образом, либо $a.X < X'_A$, либо $a.Y < Y'_A$. Рассмотрим оба случая:

- Если $a.X < X'_A$, то $a.Y > Y_M$, иначе $a$ доминирует над $a_{\min}$. Следовательно, $b.Y > Y_M$. Тем не менее, $b.X \geq X'_A$, так что точка $a_{\min} \in A'$ действительно доминирует над $b$. Противоречие.

- Если $a.Y < Y'_A$, то $a.X > X_M$, иначе $a$ доминирует над $a_{\max}$. Следовательно, $b.X > X_M$. Тем не менее, $b.Y \geq Y'_A$, так что точка $a_{\max} \in A'$ действительно доминирует над $b$. Противоречие.

Поскольку оба случая противоречивы, лемма доказана.

Иными словами, доказано, что непрерывный фрагмент $i$-го уровня всегда доминирует непрерывный фрагмент $i+1$-го уровня. Это объясняет корректность разделения уровня на три части и перемещения среднего фрагмента на следубющий уровень.

**Теорема 2.2** (Корректность итерации)**.** *Для каждой итерации $t$ ($t \geq 1$) определим следующие переменные:*

- $L_i^t$ *- уровень ранга $i$;*
- $S^t$ *- ранг уровня, на который добавляются точки;*
- $C^t$ *- множество добавляемых в $L_{s^t}^t$ точек;*

*Пусть $M$ - число уровней перед началом работы алгоритма.*

*Если изначальная структура уровней недоминирования корректна (т.е. никакая точка из $L_1^1$ не доминируется какой-либо другой точкой, и для любыч $k \geq 2$ и $a \in L_k^1$ существует $b \in L_{k-1}^1$, которая доминирует над $a$, ни никакие две точки из $L_k^1$ не доминируют друг над другом) и $S^1$ выбран функцией* SmallestNonDominatingLayer*, должны выполняться следующие условия:*

1. *Число уровней в начале итерации $t$ равно $M$.*
2. *Уровни $L_1^t \ldots L_M^t$ образуют корректную структуру уровней недоминирования.*
3. *Каждая точка из $C^t$ доминируется хотя бы одной точкой из $L_{S^t-1}^t$, if $S^t > 1$.*
4. *Если $t > 1$, то $S^t = S^{t-1} + 1$ и существуют $X^t$ и $Y^t$, такие что $C^t = \{c : c \in L_{S^t-1}^{t-1}, c.X \geq X^t, c.Y \geq Y^t\}$.*
5. *Если $t > 1$, то для любого $1 \leq i \leq M$, $i \neq S^{t-1}$ верно $L_i^{t-1} = L_i^t$.*

**Доказательство.** Первое утверждение верно, потому как любое добавление нового уровня влечет за собой остановку алгоритма вставки.

Пятое утверждение верно, так как на $t$-й итерации меняется только $S^t$-й уровень.

Остальные утверждения доказываются по индукции. База индукции - $t = 1$, когда:

- Второе и третье утверждения верны по определению;
- При $t = 1$, четвертое утверждение не проверяется.

Пусть данные утверждения верны для $t$. Следующая итерация будет выполнена, если уровень $L^t_{S^t}$ разделен по минимальным $X$ и $Y$ из $C^t$ таким образом, что средняя часть $T_M$ непуста, и хотя бы одно из множеств $T_L$ и $T_R$ также непусто.

Поскольку $C^{t+1} = T_M$, по лемме 2.1 значения $X^{t+1}$ и $Y^{t+1}$ существуют. Условие $S^{t+1} = S^t + 1$ выполяется согласно строке 43 в листинге 7. Таким образом, четвертое утверждение верно для $t + 1$.

Поскольку $C^{t+1} = T_M$, для любой точки $a \in C^{t+1}$ существует $b \in C^t$, такая что $b$ доминирует над $a$. Тем не менее, $C^t \subset L^{t+1}_{S^t} = L^{t+1}_{S^{t+1}-1}$. Таким образом, третье утверждение также верно для $t + 1$.

Чтобы доказать пятое утверждение, необходимо доказать следующие утверждения:

- Никакие две точки из $L^{t+1}_{S^t}$ не доминируют друг над другом. Поскольку $L^{t+1}_{S^t} = T_L \cup C^t \cup T_R$ и $T_L \subset L^t_{S^t}$, $T_R \subset L^t_{S^t}$, необходимо показать что:
  - Для любых $a \in C^t$ и $b \in T_L$, $a$ и $b$ не доминируют друг над другом. Если $t = 1$, это выполняется по определению $S^t$. Иначе, $C^t \subset L^{t-1}_{S^t-1}$ и $T_R \subset L^{t-1}_{S^t}$ согласно индукционному предположению, и $b$ не может доминировать над $a$. Тем не менее, $b.X < a.X$ по построению, так что $a$ не может доминировать над $b$.
  - Для любых $a \in C^t$ и $b \in T_R$, $a$ и $b$ не доминируют друг над другом. Данное доказательство симметрично предыдущему.

- Для любой $a \in L^{t+1}_{S^t}$ существует $b \in L^{t+1}_{S^t-1}$, такая что $b$ доминирует над $a$. Это выполняется, поскольку $L^{t+1}_{S^t} = T_L \cup C^t \cup T_R$, $T_L \subset L^t_{S^t}$, $T_R \subset L^t_{S^t}$, $L^t_{S^t}$ и $C^t$ доминируются $L^t_{S^t-1}$ и $L^{t+1}_{S^t-1} = L^t_{S^t-1}$.
- Для любой $a \in L^{t+1}_{S^t+1}$ существует $b \in L^{t+1}_{S^t}$, такая что $b$ доминирует над $a$. Поскольку $L^{t+1}_{S^t+1} = L^t_{S^t+1}$, для любой $a$ существует $b' \in L^t_{S^t}$, такая что $b'$ доминирует над $a$. Поскольку $L^{t+1}_{S^t} = T_L \cup C^t \cup T_R$, $T_L \subset L^t_{S^t}$, $T_R \subset L^t_{S^t}$, утверждение верно в случае если $b' \in T_L \cup T_R$. Единственная альтернатива - $b' \in T_M$. Тем не менее, для каждой $t \in T_M$ существует $t' \in C^t$, такая что $t'$ доминирует над $t$, так что $t'$ также доминирует над $a$.

  По итогам анализа данных утверждений, теорема доказана.

**Теорема 2.3** (Корректность алгоритма)**.** *Если перед выполнением вставки уровни недоминирования образовывали корректную структуру, то после завершения работы алгоритма:*

1. *Уровни будут формировать корректную структуру;*
2. *Каждая точка, присутствовавшая в структуре до запуска алгоритма, останется в структуре;*
3. *Добавляемая точка будет присутствовать в структуре.*

**Доказательство.** Поскольку при вставке никакие точки не удаляются, второе утверждение верно.

Поскольку добавляемая точка изначально присутствует в множестве перемещаемых точек, она будет добавлена в структуру, поскольку алгоритм завершается только в случае, если множество перемещаемых точек становится пустым. Таким образом, третье утверждение также верно.

Для доказательства первого утверждения необходимо рассмотреть условия завершения работы алгоритма:

- Алгоритм завершает работу на строке 34 листинга 7. В этот момент отсутствуют решения, которые необходимо добавить на какой-либо

уровень. По теореме 2.2, уровни недоминирования будут формировать корректную структуру.

- Алгоритм завершает работу на строке 40 листинга 7. В этом случае уровень, образованный $C$, целиком доминирует над уровнем, образованным $T_M = G.V$, который, в свою очередь, доминирует над всеми последующими уровнями (по второму условию теоремы 2.2). Таким образом, новый уровень, образованный $T_M$, может быть вставлен после $C$ без нарушения первого утвердления данной теоремы.
- Алгоритм завершает работу на строке 46 листинга 7. Перед добавлением нового уровня, все точки из $C$ не доминируют друг над другом, поскольку $C$ либо состоит из одной точки, либо представляет из себя фрагмент уровня. К тому же, все точки из $C$ доминируются первым уровнем (третье утверждение теоремы 2.2). Таким образом, после вставки такого уровня структура уровней недоминирования останется корректной.

Теорема доказана, поскольку все условия завершения алгоритма корректны.

Время работы алгоритма вставки состоит из времени работы алгоритма поиска уровня ($O(\log M \log(N/\log M))$), а также суммарного времени выполнения всех итераций.

Пусть было выполнено $P \leq M$ итераций. Предположим, что уровни $L_1 \dots L_P$ были разделены во время этих итераций. Обозначим размеры этих уровней после разделения как $L_1^L, L_1^M, L_1^R, \dots, L_P^L, L_P^M, L_P^R$. Значение $L_0^M = 1$ соответствует изначальному состоянию множества $C$ с единственным элементом - добавляемой точкой.

На $i$-й итерации были выполнены следующие операции сложности $\omega(1)$:

- Нахождение минимума и максимума $C$ за $O(1 + \log L_{i-1}^M)$;
- SplitX за $O(1 + \log(L_i^L + L_i^M + L_i^R))$;
- SplitY за $O(1 + \log(L_i^M + L_i^R))$;

- Внутренний MERGE за $O(1 + \log(L_{i-1}^M + L_i^R))$;
- Внешний MERGE за $O(1 + \log(L_i^L + L_{i-1}^M + L_i^R))$.

Сумма всех значений не превышает $4\sum_{i=1}^{P} L_i$, что дает итоговую сложность $O(N)$. По неравенству Коши, сумма времен работы всех итераций составляет $O(P(1 + \log(N/P)))$. При фиксированном $N$, эта функция достигает максимума при $P = \Theta(N)$, когда $O(P(1 + \log(N/P))) = O(M(1 + \log(N/M)))$ и достигается худшее время работы $O(N)$. Процедуры вставки новых уровней могут выполняться только в конце алгоритма и требуют лишь $O(\log M)$, а потому не влияют на асимптотическую сложность

Суммарное время работы алгоритма вставки выглядит следующим образом:

$$O\left(M\left(1 + \log\frac{N}{M}\right) + \log M \log\frac{N}{\log M}\right).$$

### 2.3.1. Удаление худшей точки

Большинство алгоритмов многокритериальной оптимизации не требует возможности удаления случайной точки. Удалять необходимо "худшие"точки, которые хранятся на последнем уровне предложенной структуры данных. Их можно удалять за $O(\log N + \log M)$.

# Глава 3. Результаты работы алгоритма

В данной главе проводится сравнение предложенного алгоритма с аналогами и оценка времени его работы на различных множествах тестовых данных.

## 3.1. Конфигурация эксперимента

Было проведено сравнение с ENLU и алгоритмом недоминирующей сортировки, входящим в NSGA-II. Все алгоритмы тестировались на одном и том же, случайно сгенерированном множестве точек. Для популяционной сортировки измерялось время сортировки всей популяции. Инкрементальные алгоритмы тестировались путем последовательного добавления всех точек в структуру данных. Замерялось суммарное время работы всех операций вставки.

Тестирование проводилось на множествах точек размерностей 250, 500, 1000, 2000 и 4000. Для каждой конфигурации выполнялось по 100 запусков, для каждого запуска множество точек генерировалось заново. Время работы алгоритмов вычислялось путем вычитания времени запуска алгоритма из времени его завершения. Время старта и время завершения определялись с помощью процедуры *java.lang.System.nanoTime()*.

### 3.1.1. Тестовые данные

Для оценки производительности использовались следующие виды входных данных:

- “квадрат”: генерируются $N$ случайных точек, равномерно распределенные в квадрате $N \times N$;
- “параллель”: генерируются $N$ случайных точек, $N/2$ из которых лежат на прямой $y = N - x$, а остальные - на $y = N - x + 1$;

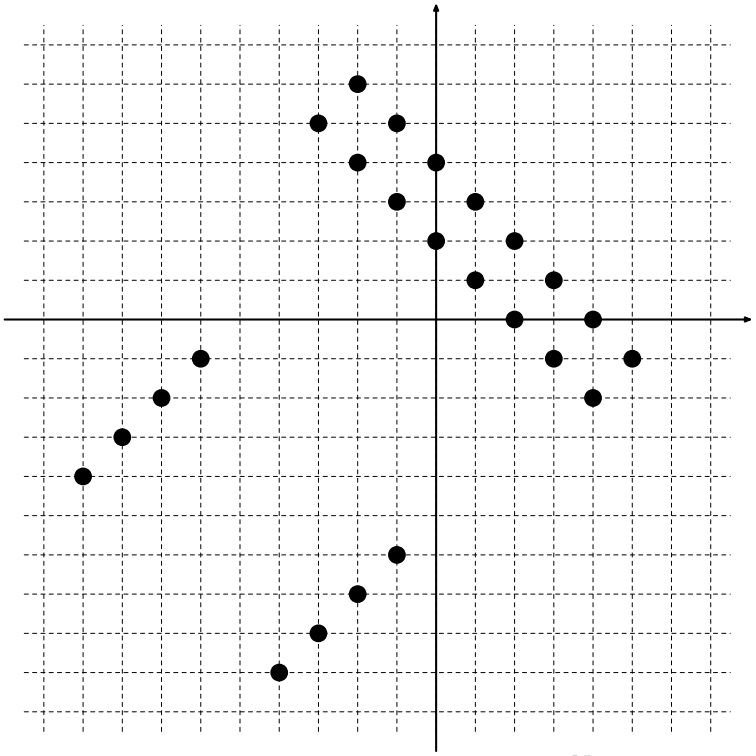

Рис. 3.1: Пример структуры “П” при $N = 24$

- “diag1”: генерируется последовательность из $N$ точек $(x, x)$, начиная с максимального $x$;
- “diag2”: генерируется последовательность из $N$ точек, поочередно находящихся на прямых $(x, x+5)$ и $(x+5, x)$, начиная с максимального $x$;
- “П”: генерируется “параллельно-перпендикулярная” структура (напоминающая повернутую на 45 градусов кириллическую букву П) следующего вида: $N/6$ точек на прямой $y = x + 5$, $N/6$ точек на прямой $y = x - 5$, $N/3$ точек на прямой $y = N/3 - x - 4$ и $N/3$ точек на прямой $y = N/3 - x - 6$. Пример такой структуры приведен на Рис. 3.1.

## 3.2. Результаты тестирования

При оценке производительности измерялось время работы алгоритма, а также суммарное число сравнений точек для проверки на доминирование. На большинстве тестов предложенный алгоритм показал значи-

тельно лучшие результаты чем сравниваемые решения.

Разработанный алгоритм показал худшие результаты были на данных “diag1” и “diag2” - время работы оказалось сравнимо с ENLU и алгоритмом сортировки из NSGA-II соответственно.

Результаты теста “diag1” (Рис. refplot-diag1-time) объявняются тем, что в данном тесте каждая вставка происходит на уровень недоминирования минимального ранга. Таким образом, при каждой вставке все точки, уже содержащиеся в популяции, увеличивают свой ранг на единицу (то есть, новая точка доминирует над всей популяцией). Алгоритм ENLU в таком случае определяет ранг за $O(1)$, и выполняет вставку нового уровня недоминирования за $O(\log M)$ (при условии хранения уровней недоминирования в двоичном дереве поиска - например, в красно-черном дереве). Предложенный нами алгоритм в данном случае имеет такую же асимптотическую эффективность.

Результаты теста “diag2” (Рис. 5.9) объясняются тем, что алгоритм сортировки из NSGA-II в данном случае работает за $O(N^2)$ (это время определяется временем определения ранга, которое у данного алгоритма постоянно, то есть не зависит от значений точек). Предложенный нами алгоритм на таких входных данных достигает худшего случая, так как при каждой второй вставке образуется множество точек (размера 1), "выталкивыемых"на следующий уровень недоминирования. Таким образом, добавление $N$ точек происходит за $O(N)$, что дает итоговое время работы $O(N^2)$ - такое же как для алгоритма сортировки из NSGA-II.

Лучшее время работы (дающее более чем стократное превосходство над алгоритмом ENLU) было достигнуто на П-образных входных данных при больших размерах популяции. ENLU в таком случае выполняет каждую вставку за квадратичное время, в то время как предложенный алгоритм даже в худшем случае работает за линейное время.

Многократное превосходство над NSGA-II в тесте “diag1” объясняется тем, что предложенный алгоритм (равно как и ENLU) выполняет каж-

дую вставку за $O(\log M)$, что дает суммарное время работы $O(N \log M)$, что значительно меньше времени работы NSGA-II, составляющего $O(N^2)$.

# Глава 4. Заключение

В данной работе предложен новый алгоритм инкрементальной недоминирующей сортировки, позволяющий выполнять добавление новой точки в двумерном пространстве за линейное время. Удаление худшей точки можно выполнить за $O(\log M)$, где $M$ - максимальный ранг точки в структуре данных. Была доказана корректность работы алгоритма и проведена оценка времени его работы.

Для эффективной реализации процедур вставки и удаления точек была разработана структура данных на основе Декартовых деревьев. Ранее такой подход для выполнения недоминирующей сортировки не применялся.

Результаты тестов показали, что предложенный алгоритм работает за время, сравнимое с временем работы наилучших известных автору алгоритмов классичеческой и инкрементальной сортировки в худшем случае, и за значительно меньшее (иногда - в десятки раз) время в большинстве случаев.

Использование предложенного алгоритма позволяет эффективно реализовать инкрементальные версии эволюционных алгоритмов - в частности, NSGA-II [5].

# Глава 5. Приложение 1. Результаты тестов

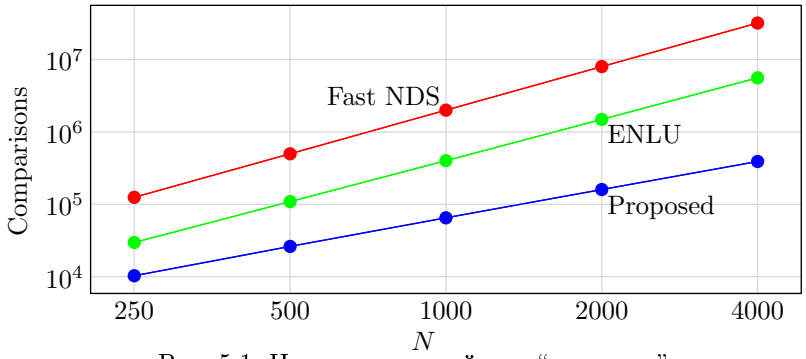

Рис. 5.1: Число сравнений для “квадрата”

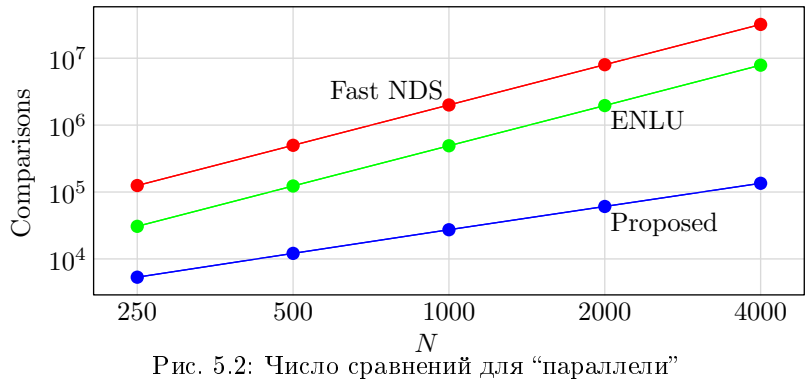

Рис. 5.2: Число сравнений для “параллели”

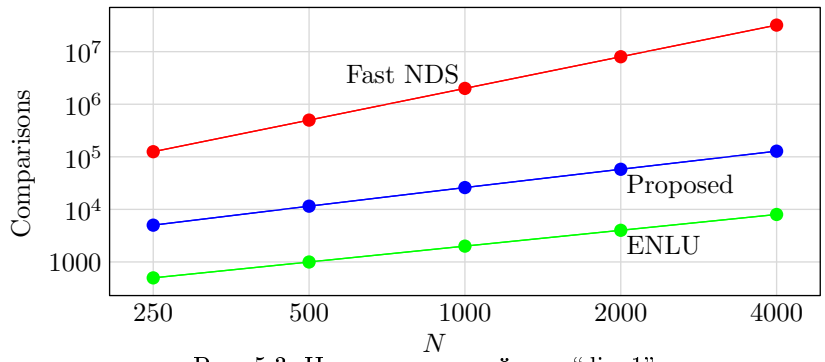

Рис. 5.3: Число сравнений для “diag1”

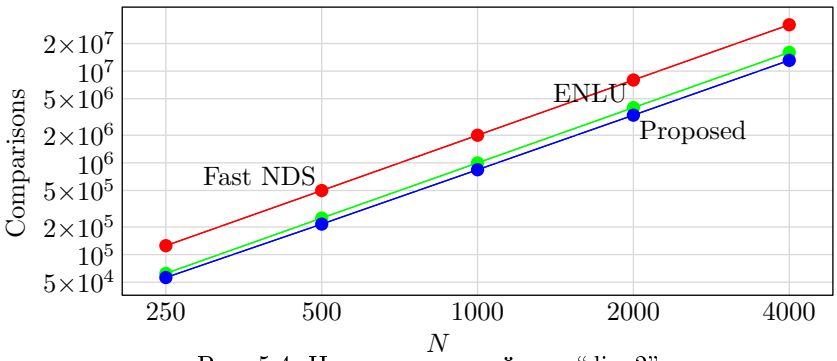

Рис. 5.4: Число сравнений для “diag2”

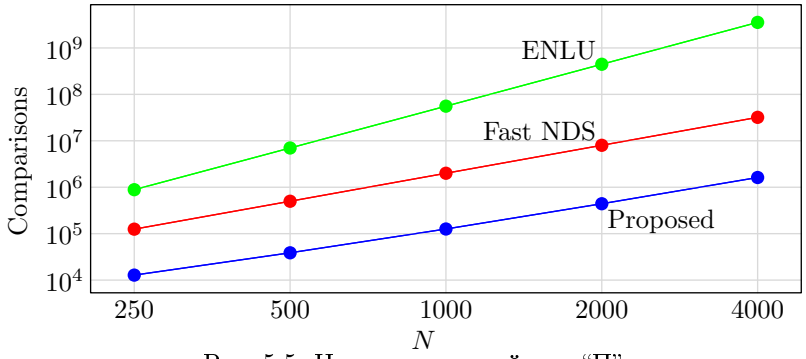

Рис. 5.5: Число сравнений для “П”

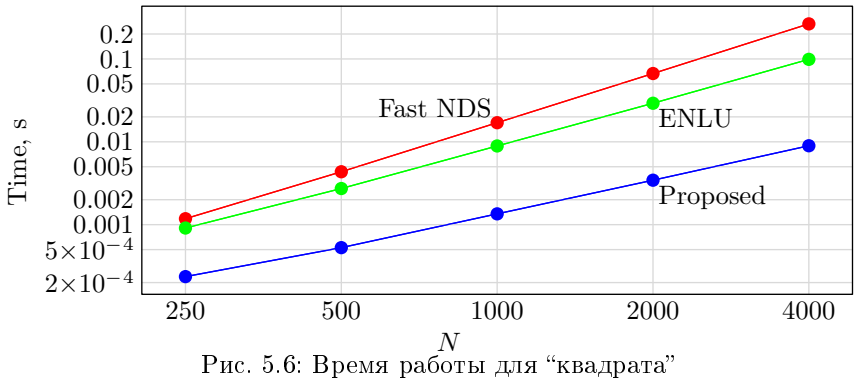

Рис. 5.6: Время работы для “квадрата”

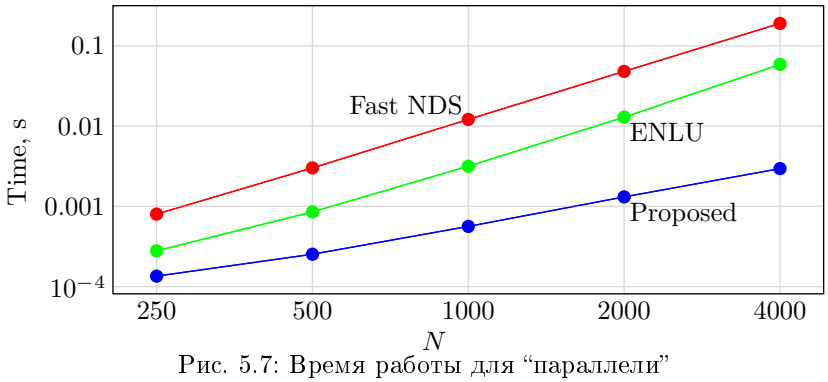

Рис. 5.7: Время работы для “параллели”

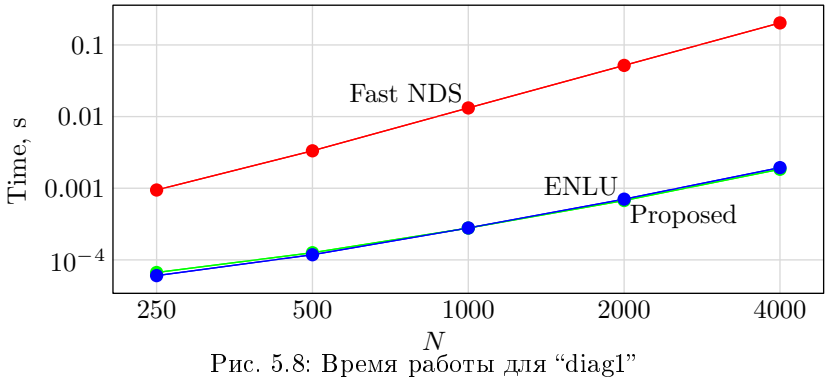

Рис. 5.8: Время работы для “diag1”

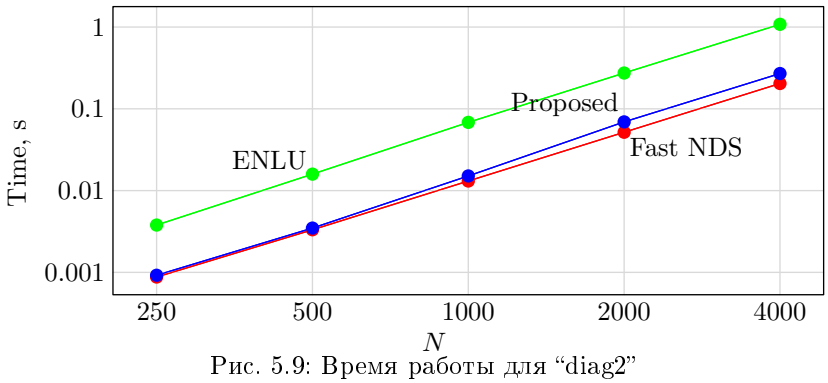

Рис. 5.9: Время работы для “diag2”

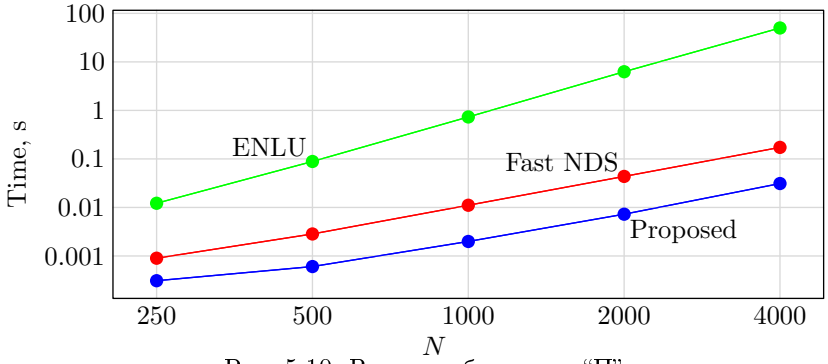

Рис. 5.10: Время работы для “П”

# Список литературы


[1]. *Wikipedia. Оптимизация (математика).* URL: https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0).

[2]. *Wikipedia. Многокритериальная оптимизация.* URL: https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BA%D1%80%D0%B8%D1%82%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F.

[3]. *Wikipedia. Эффективность по Парето.* URL: https://ru.wikipedia.org/wiki/%D0%AD%D1%84%D1%84%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%BF%D0%BE_%D0%9F%D0%B0%D1%80%D0%B5%D1%82%D0%BE.

[4]. Deb K. Pratap A. Agarwal S. Meyarivan T. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". In: *Transactions on Evolutionary Computation*. 2. 2002, pp. 182–197.

[5]. Buzdalov M. Yakupov I. Stankevich A. "Fast Implementation of the Steady-State NSGA-II Algorithm for Two Dimensions Based on Incremental Non-Dominated Sorting". In: *Proceedings of Genetic and Evolutionary Computation Conference*. 2015.

[6]. Петрова И. А. *Повышение эффективности алгоритмов многокритериальной оптимизации с помощью машинного обучения для решения задач составления расписаний.* 2013. URL: http://is.ifmo.ru/diploma-theses/2013/bachelor/petrova/petrova.pdf.

[7]. N. Srinivas and K. Deb. "Multiobjective function optimization using nondominated sorting genetic algorithms". In: *Evolutionary Computation*. 2. 1995, pp. 221–248.

[8]. M. Laumanns E. Zitzler and L. Thiele. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization." In: *Proceedings of the EUROGEN'2001 Conference*. 2001, pp. 95–100.

[9]. Juan J. Durillo Antonio J. Nebro Francisco Luna and Enrique Alba. "On the Effect of the Steady-State Selection Scheme in Multi-Objective Genetic Algorithms". In:

[10]. F. Luccio H. T. Kung and F. P. Preparata. "On finding the maxima of a set of vectors". In: *Journal of ACM*. 22(4). 1975, pp. 469–476.

[11]. M. T. Jensen. "Reducing the Run-time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms." In: *Transactions on Evolutionary Computation*. 7(5). 2003, pp. 503–515.

[12]. M. Buzdalov and A. Shalyto. "A provably asymptotically fast version of the generalized Jensen algorithm for non-dominated sorting." In: *International Conference on Parallel Problem Solving from Nature*. 8672. 2014, pp. 528–537.

[13]. K. Li K. Deb Q. Zhang and S. Kwong. "Efficient non-domination level update approach for steady-state evolutionary multiobjective optimization. Technical report". In: 2014.

[14]. Jean Vuillemin. "A Unifying Look at Data Structures". In: *Communications of ACM* 23.4 (1980), pp. 229–239.

[15]. Daniel D. Sleator and Robert E. Tarjan. "Self-Adjusting Binary Search Trees". In: *Journal of ACM* 32.3 (1985), pp. 652–686.