Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 21.Б07-мм

Улучшение кода алгоритма по поиску численных ассоциативных правил в Desbordante

ВОЛГУШЕВ Иван Романович

Отчёт по производственной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2025

Оглавление

Введение

В контексте современной разработки программного обеспечения рефакторинг становится [1] неотъемлемой частью жизненного цикла программного продукта. Масштабы кодовых баз велики, как никогда, а вместе с этим растут и размеры команд разработчиков. В таких условиях поддержание высокого качества кода становится критически важным, поскольку трудоемкость общения между разработчиками и сложность взаимодействия между компонентами системы значительно увеличиваются. Однако написание кода, отвечающего индустриальным стандартам, не всегда возможно с самого начала. В силу сжатых сроков, отсутствия профессиональных навыков или других причин происходит накопление технического долга. Это проявляется в ухудшении структуризации и читаемости кода, неэффективном использовании вычислительных ресурсов и памяти, зависимостях от конкретных версий библиотек и платформ, отсутствии тестирования. Иногда это может привести к значительному замедлению разработки или даже к невозможности её продолжить, не прибегнув к рефакторингу.

Desbordante [2] — наукоёмкий профайлер данных с открытым исходным кодом. На момент написания этой работы основной репозиторий Desbordante насчитывает 51 участника, внесшего вклад в исходный код проекта. При этом, проект являтся достаточно большим: на данный момент он включает 681 файл на языке C++, составляющий ядро профайлера, и ещё 116 C++-файлов, связанных с Python интерфейсом и тестами.

Проект Desbordante ориентирован на длительный жизненный цикл, что делает его уязвимым к проблемам, описанным в первом абзаце. Долговечность проекта требует особого внимания к этим вопросам, так как их влияние может оказаться критическим для успешного функционирования. Открытый исходный код вносит свою специфику. Плохая структуризация и читаемость кода могут отпугнуть потенциальных участников, поскольку усилия на их интеграцию в процесс разработки станут слишком велики по сравнению с усилиями, затраченными на

полезный вклад в проект. Открытый исходный код предполагает возможность дистанционной разработки любыми заинтересованными лицами, и передача знаний о кодовой базе в таких условиях осложнена по сравнению с разработкой в финансируемой команде. Это связано с отсутствием непрерывного взаимодействия между разработчиками, чего нельзя сказать, к примеру, о командах, проводящих ежедневные онлайн-собрания или же имеющих общие офисные помещения. Помимо этого, лица, заинтересованные в разработке, могут покинуть проект в любой момент. Если знание об устройстве значительной части кода принадлежало только им, то их уход из проекта приведёт к появлению "чёрных ящиков" — разделов исходного кода, об устройстве которых не осведомлён ни один из активных разработчиков, восстановление знаний о которых требует значительных трудозатрат.

Оптимизация также играет значительную роль в Desbordante. Ядро Desbordante написано на C++ — языке, дающем точный контроль над использованием ресурсов платформы. Это является одним из ключевых преимуществ [3] Desbordante перед аналогами.

В течение прошлого семестра автор курсовой работы реализовывал[1] алгоритм для поиска численных ассоциативных правил под названием Differential Evolution Solver (DES) [4] в Desbordante. Была реализована и интегрирована в проект первая версия данного алгоритма, которая вошла в релиз 2.3.0.

В этом семестре я сфокусируюсь на рефакторинге и документировании существующего кода алгоритма DES. Документирование позволит сохранить знания, а обеспечение читаемости и структуризация — значительно облегчат их восстановление. Наконец, для DES также возможны множественные оптимизации использования памяти и процессорного времени.

По этим причинам, улучшение структуры и оптимизация кода DES стали основной задачей этой курсовой работы.

[1] https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/DES Integration - Ivan Volgushev - 2024 spring.pdf (дата доступа: 8 марта 2025 г.).

1. Постановка задачи

Целью работы стал рефакторинг алгоритма DES, созданного в течение предыдущей учебной практики. Для этого были поставлены следующие задачи:

- Провести анализ, выявить недостатки нынешней реализации DES и классифицировать их по типам;
- Исправить эти недостатки;
- Произвести экспериментальное сравнение двух реализаций.

2. Обзор изменений

Будем называть ту версию кода алгоритма DES, которая была создана в течение прошлой практической работы, изначальной версией кода или старым кодом. Модифицированную версию кода DES, добавленную в основную ветку репозитория Desbordante, будем называть новым кодом или модифицированным кодом.

Изначальный код, несмотря на успешное прохождение обязательных тестов непрерывной интеграции, имеет множество недостатков. Ниже представлены наиболее значимые изменения, внесённые в изначальный код, разделённые на три категории. Списки изменений в категориях представлены в таблицах 1, 2 и 3. В случае, если аналогичные изменения из таблицы были внесены во множестве мест в изначальном коде, в скобках указывается их число. В общей сложности изменения затронули 18 из 22 файлов, 440 строк было удалено и 352 строки было вставлено согласно системе контроля версий git.

2.1. Читаемость

Таблица 1: Стиль кода / Читаемость

Улучшение	Количество
публичные поля следуют за приватными	(3)
перенос многострочной реализации в файл .cpp из .h	(2)
укорочение пространства имён при вызове функции	
`Type var_ = Type()` теперь `Type var_{}`	(2)
замена цикла `for` на аналог из `std::`	(4)
замена `sort()` на `ranges::sort()`	(3)
обращение к статическому методу через оператор `::`	
добавление комментария	
удаление излишнего выражения `break`	
удаление излишнего `else`	(2)
удаление лишнего пространства имён при вызове	
введение единого порядка `kString kDouble kInt`	
замена `.size() == 0` на `.empty()`	(2)
`CalcQualities` теперь имеет единообразие вывода	
`CalcQualities` имеет все вычисления в одном месте	
замена постинкремента преинкрементом	(2)
удаление лишнего явного преобразования типа	
замена имени на более явное	
добавление переноса строки в конце файла	(5)
`using` для укорочения имён	
изменение логики на более простую в `SetQualities()`	
упрощение `MapFitsValue()`	

Изменения, представленные в таблице 1 имеют отношение к стилю кода и его читаемости. В качестве примера того, как подобные изменения влияют на лёгкость восприятия кода, можно рассмотреть функцию `CalcQualities()`, которая имеет 5 параметров типа `size_t` и возвращает структуру `NARQualities`:

```
1 NARQualities CalcQualities(size_t num_rows_fit_ante, size_t
    num_rows_fit_ante_and_cons,
2                            size_t included_features, size_t feature_count,
    size_t num_rows) {
3     NARQualities result;
4     if (num_rows_fit_ante == 0) {
5         result.fitness = 0.0;
6         result.confidence = 0.0;
7         return result;
8     } else {
9         result.confidence = num_rows_fit_ante_and_cons / (double)
    num_rows_fit_ante;
10    }
11    result.support = num_rows_fit_ante_and_cons / (double)num_rows;
12    if (result.support == 0.0) {
13        result.fitness = 0.0;
14        result.support = 0.0;
15        return result;
16    }
17
18    double inclusion = included_features / (double)feature_count;
19    result.fitness = (result.confidence + result.support + inclusion) / 3.0;
20    return result;
21 }
```

Листинг 1: Изначальный вид функции CalcQualities

`NARQualities` предназначена для расчёта трёх характеристик, которые используются в алгоритме DES для оценки численных ассоциативных правил и определения того, насколько они полезны: поддержки (support), доверия (confidence) и приспособленности (fitness). На вход принимаются размеры множеств строк и столбцов, удовлетворяющим определённым частям численного ассоциативного правила. Изначально функция `NARQualities` занимала 21 строку.

```
1 NARQualities CalcQualities(size_t num_rows_fit_ante, size_t
    num_rows_fit_ante_and_cons,
2                            size_t included_features, size_t  feature_count,
    size_t num_rows) {
3     if (num_rows_fit_ante == 0) {
```

```
4         return {0.0, 0.0, 0.0};
5     }
6     double support = static_cast<double>(num_rows_fit_ante_and_cons) /
      num_rows;
7     if (support == 0.0) {
8         return {0.0, 0.0, 0.0};
9     }
10    double confidence = static_cast<double>(num_rows_fit_ante_and_cons) /
      num_rows_fit_ante;
11    double inclusion = static_cast<double>(included_features) / feature_count;
12    double fitness = (confidence + support + inclusion) / 3.0;
13    return {fitness, support, confidence};
14 }
```

Листинг 2: Модифицированная функция CalcQualities

Обновлённая версия функции, представленная выше, занимает 14 строк и выполняет то же самое вычисление. Структура `NARQualities`, созданием которой занимается эта функция, неявно создаётся в последней строке функции после оператора `return`. NARQualities является агрегатным типом, и такое сокрытие её конструктора заостряет на этом внимание, при этом значительно укорачивая названия переменных, которые в конечном счёте будут объединены для создания `NARQualties`. В краевых случаях, за которые отвечают два оператора `if` также используется неявное конструирование `NARQualties`, и во всех краевых случаях теперь явно возвращается одинаковое значение — структура со всеми полями, равными нулю. Кроме того, если первая ветка `if` содержит оператор `return`, то в операторе `else` нет необходимости.

Таким образом, подобные изменения облегчают восприятие кода и делают его поведение более очевидным.

2.2. Расширяемость

Таблица 2: Облегчение расширения

Улучшение	Количество
замена иерархии `ValueRange` на шаблон	
добавление теста	
вынесение повторного вычисления в переменную	(3)
вынесение локальной функции	
шаблонный оператор `[]`	
добавление обработки ошибки (пустой набор данных)	
добавление значения по умолчанию для члена класса	
добавление `nodiscard`	
замена синглтона RNG на передаваемое поле в классе	
двойное объявление функции	
добавление `const` к методам	(4)
добавление `assert`	
замена `uint` на `size_t`	(2)

Изменения из этой категории направлены на облегчение дальнейшей разработки и поддержки кодовой базы. Одним из самых больших подобных изменений стало объединение классов ValueRange под единым шаблоном, что позволяет легко добавить новые типы промежутков, например, промежутки дат, для поиска числовых ассоциативных правил на таблицах с датами.

```
1 std::shared_ptr<ValueRange> CreateValueRange(model::TypedColumnData const&
    column) {
2     switch (column.GetTypeId()) {
3         case TypeId::kString:
4             return std::make_shared<StringValueRange>(column);
5         case TypeId::kDouble:
6             return std::make_shared<NumericValueRange<Double>>(column);
7         case TypeId::kInt:
8             return std::make_shared<NumericValueRange<Int>>(column);
9         default:
```

```
10             throw std::invalid_argument(std::string("Column has invalid
    type_id in function: ") + __func__);
11     }
12 }
```

Листинг 3: Функция CreateValueRange

В новом коде функция `CreateValueRange()` используется для создания объектов `ValueRange` и является легко расширяемой. При внесении необходимых изменений в другие части кода, возможно будет добавить в эту функцию следующий `case`:

```
case TypeId::kDate:
    return std::make_shared<NumericValueRange<Date>>(column);
```

Это позволит использовать тип данных `Date` в промежутках значений, при наличии у `Date` всех необходимых интерфейсов и операторов. Однако, это расширение выходит за рамки этой курсовой работы.

2.3. Производительность

Таблица 3: Производительность

Улучшение	Количество
передача параметра через move	(6)
передача параметра по константной ссылке	(7)
ссылка у параметра	(2)
замена string на ostringstream в ToString()	(2)
добавление reserve() когда размер vector известен	(5)

Изменения из этой категории направлены на более эффективное использование ресурсов компьютера. Язык C++ позволяет использовать ссылки и move-семантику для предотвращения избыточного копирования данных. Функция std::move преобразует переменную в r-value выражение. Это позволяет передавать владение ресурсами от одной

переменной к другой без затрат на копирование. Такой подход особенно полезен при работе с крупными объектами, например, контейнерами стандартной библиотеки. Однако после перемещения исходная переменная остаётся в валидном, но неопределённом состоянии, и её повторное использование без присваивания новых данных может привести к ошибкам.

Для повышения производительности в DES внесённые изменения сыграли значительную роль. Конкретные эксперименты по сравнению производительности приведены в следующей главе.

3. Эксперимент

Читаемость и производительность кода зачастую требуют компромиссов. После рефакторинга, ориентированного на реструктуризацию кода в пользу читаемости, следует ожидать ухудшения производительности программы. Однако рефакторинг был также направлен и на улучшение производительности. Было решено произвести экспериментальное сравнение старой и новой реализации. Для этого были поставлены следующие исследовательские вопросы (ИВ):

- ИВ1: Как введённые изменения повлияли на время исполнения и потребляемую память алгоритма?
- ИВ2: Повлияли ли введённые изменения на выходные данные алгоритма?

Оптимизацию можно назвать успешной лишь в том случае, если новая версия алгоритма демонстрирует как минимум такое же качество результатов, что и старая, при этом обеспечивая снижение времени выполнения и/или потребления памяти.

3.1. Методология эксперимента

Эксперимент производился на трёх наборах данных, общая информация о которых представлена в таблице 4.

Таблица 4: Информация о наборах данных

	строки	**атрибуты**
abalone.csv	4177	9
cancer.csv	570	30
iris.csv	150	5

Запуски алгоритма, проводимые на одинаковых наборах данных, могут иметь различные параметры, которые изменяют поведение алгоритма DES и приводят к различным результатам. От запуска к запуску

варьировались два параметра DES: “популяция” и “итерации”, остальные имели значения по умолчанию. Параметр “популяция” отвечает за количество правил, которые будут “мутировать” и “скрещиваться” в течение времени исполнения алгоритма. Параметр “итерации” отвечает за количество “скрещиваний” и “мутаций”, которые будут произведены в течение работы алгоритма. Более подробно алгоритм и его параметры описаны в статье [4]. Сценарии, по которым производились запуски алгоритмов, представлены в таблице ниже:

Таблица 5: Информация о сценариях

сценарий	**строки**	**популяция**	**итерации**	**набор данных**
small abalone	4177	1000	1000	abalone.csv
big abalone	4177	4500	4500	abalone.csv
small cancer	570	1000	1000	cancer.csv
big cancer	570	4500	4500	cancer.csv
big iris	150	4500	4500	iris.csv
huge iris	150	7200	14400	iris.csv

Измерение времени выполнения проводилось при помощи функции `std::clock()` из стандартной библиотеки, измерение использованной памяти проводилось при помощи модуля massif из инструментального программного обеспечения valgrind.

3.2. Исследование результатов эксперимента

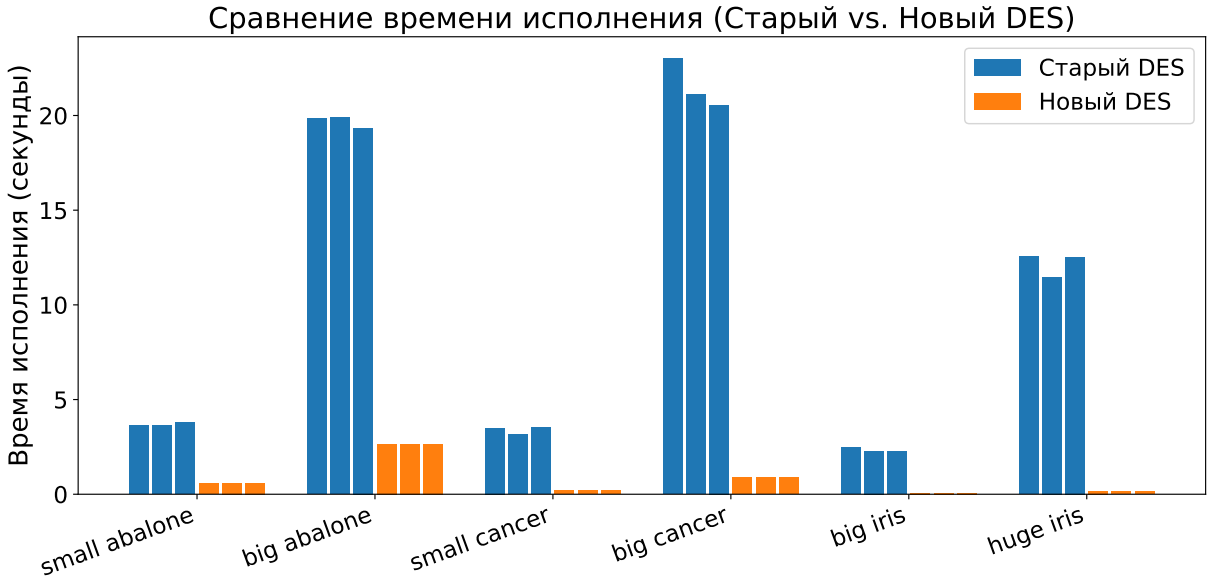

Рис. 1: Сравнение времени исполнения (Новый и Старый DES)

На рисунке 1 представлено время работы старой и новой версий DES в каждом из трёх запусков всех шести сценариев. В общей сложности обе версии алгоритма были запущены 18 раз. Как видно из результатов, новая версия DES значительно превосходит старую по скорости работы: во всех сценариях время выполнения уменьшилось минимум в пять раз.

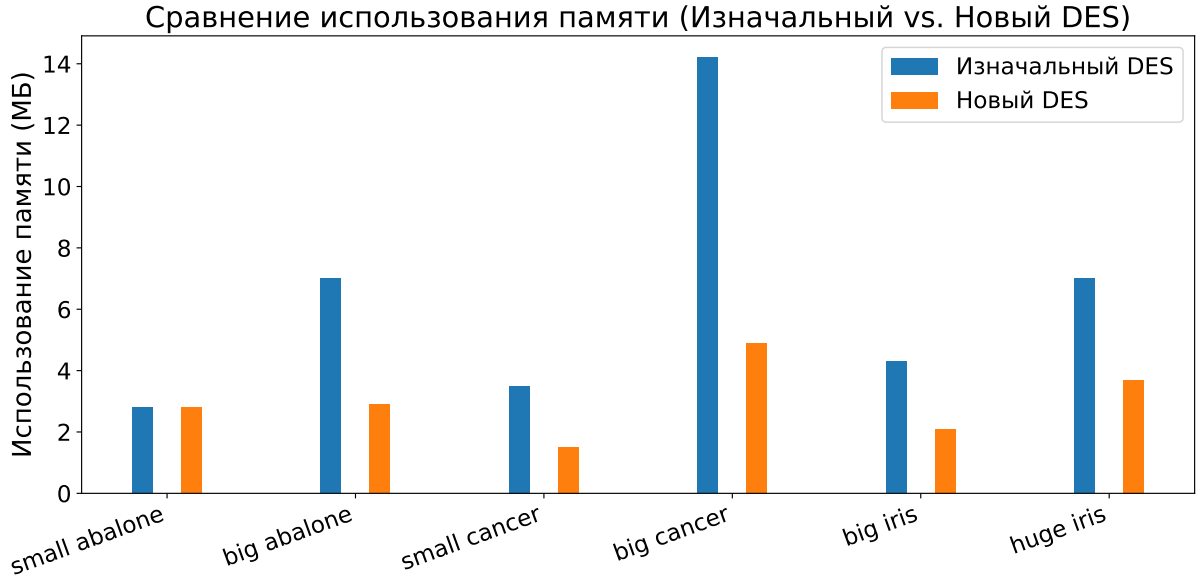

Рис. 2: Сравнение использованной памяти (Новый и Старый DES)

На рисунке 2 показано количество памяти, использованное двумя реализациями в каждом сценарии. Поскольку изменения потребления памяти при повторных запусках были незначительными, на графике для каждой версии алгоритма приведено лишь одно значение на сценарий. Как видно из результатов, во всех сценариях новая версия использовала столько же или меньше памяти по сравнению со старой.

Таким образом, был получен ответ на исследовательский вопрос один (ИВ1). Внесённые изменения значительно ускорили выполнение алгоритма и снизили затраты на память.

Таблица 6: Информация о результатах сценариев

	новый DES		**старый DES**	
сценарий	правила	“приспособленность”	правила	“приспособленность”
small abalone	179	0.74	197	0.68
big abalone	771	0.79	897	0.82
small cancer	1	0.13	2	0.18
big cancer	3	0.52	2	0.48
big iris	1038	0.78	1028	0.84
huge iris	3302	0.79	3243	0.80

В таблице 6 представлена информация о результатах запусков сценариев. Значения в столбце “приспособленность” являются максимальными значениями “приспособленности” среди численных ассоциативных правил, найденных в результате выполнения сценария, количество которых указано в столбце “правила”. “Приспособленность” правила — это значение целевой функции от этого правила, которая является средним арифметическим поддержки, доверия и включения данного правила и принимает значения от нуля до одного. Алгоритм DES является алгоритмом оптимизации и направлен на увеличение “приспособленности”. Количество найденных правил и их максимальная приспособленность являются простыми индикаторами успеха или провала этого алгоритма, поэтому сравнение будет производиться именно по этим метрикам.

Как видно по таблице 6, результаты двух реализаций отличаются.

Однако, однозначных тенденций по увеличению или уменьшению количества найденных правил и их максимальной “приспособленности” проследить невозможно. Такое поведение связано со стохастической природой алгоритма — малейшие изменения в порядке перебора или в начальных данных приводят к совершенно другим результатам. Изменения, внесённые во время рефакторинга, как раз и оказали такое влияние. Однако, всех сценариях кроме “small cancer” и “big cancer” количество обнаруженных правил различается не более чем на 15%. В случае с “small cancer” и “big cancer” отличие составляет не более чем одно правило. Разница в максимальной приспособленности составила не более, чем 0,06.

Результаты, полученные при помощи старой и новой реализации, отличаются пренебрежимо мало, учитывая стохастическую природу алгоритма. Так был получен ответ на исследовательский вопрос 2 (ИВ2).

Несмотря на введённую структуризацию кода, применённые методы оптимизации сыграли ключевую роль в повышении производительности. В результате время исполнения и затраты памяти были существенно снижены, при этом результаты работы алгоритма остались практически неизменными.

Заключение

В ходе работы был произведён рефакторинг алгоритма DES, созданного в течение предыдущей учебной практики. Были выполнены следующие задачи:

- Проведён анализ и выявлены недостатки нынешней реализации DES, и эти недостатки были классифицированы по типам;
- Эти недостатки были исправлены;
- Произведено экспериментальное сравнение двух реализаций.

Реализация интегрирована в проект Desbordante. Pull request #490 доступен на GitHub[2]. Версия кода до рефакторинга, произведённого в течение этой работы, была перенесёна в личный репозиторий[3] автора курсовой работы.

[2] https://github.com/Desbordante/desbordante-core/pull/490 (дата доступа: 27 февраля 2025 г.).

[3] https://github.com/VanyaVolgushev/desbordante-core/tree/add-des-release-squashed (дата доступа: 28 февраля 2025 г.).

Список литературы

[1] 30 Years of Software Refactoring Research: A Systematic Literature Review / Chaima Abid, Vahid Alizadeh, Marouane Kessentini et al. // CoRR. — 2020. — Vol. abs/2007.02194. — arXiv : 2007.02194.

[2] Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms / Maxim Strutovskiy, Nikita Bobrov, Kirill Smirnov, George Chernishev // 2021 29th Conference of Open Innovations Association (FRUCT). — 2021. — P. 344–354.

[3] Desbordante: from benchmarking suite to high-performance science-intensive data profiler (preprint) / George A. Chernishev, Michael Polyntsov, Anton Chizhov et al. // CoRR. — 2023. — Vol. abs/2301.05965. — arXiv : 2301.05965.

[4] Fister Iztok, Jr. Iztok Fister. uARMSolver: A framework for Association Rule Mining // CoRR. — 2020. — Vol. abs/2010.10884. — arXiv : 2010.10884.