Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 22.Б08-мм

Реализация алгоритма верификации приближенных запрещающих ограничений в проекте “Desbordante”

Аносов Павел Игоревич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
асс. кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2025

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

3. Предварительные сведения **15**

4. Алгоритм **17**

5. Реализация **20**

5.1. Алгоритм верификации ПЗО 20

5.2. Метрики 21

5.3. Pybind11 22

5.4. Интеграционное тестирование 23

Заключение **25**

Список литературы **26**

Введение

Desbordante [4] — это высокопроизводительный наукоемкий профилировщик данных, направленный на поиск скрытых зависимостей и паттернов, неподвластных стандартным методам профилирования.

Ярким примером таких зависимостей являются запрещающие ограничения (Denial Constraints, DC) [2]. Они задают комбинации значений некоторых атрибутов, которых не должно быть в таблице. Запрещающие ограничения (ЗО) могут описывать запреты разной степени сложности: от простых условий, относящихся к одной записи (например, зарплата не может быть отрицательной), до сложных правил, описывающих пары записей (для жителей одного штата у человека с большей заработной ставкой больше налоговая ставка). Это помогает находить различные ошибки и несоответствия в данных, такие как пропущенные значения, опечатки и другие неточности.

Но такие ограничения плохо приспособлены для реальных данных, в которых присутствуют умеренный шум и искажения данных. Из-за этого в ЗО может возникать небольшое количество записей/пар записей, нарушающих зависимость, несмотря на то, что в целом ограничение выполняется. Для решения этой проблемы используются приближенные запрещающие ограничения (Approximate Denial Constraints, ADC). Основная идея данного подхода заключается в том, что вводятся метрики для ЗО, позволяющие количественно оценить степень их соблюдения на определенном наборе данных. Приближенное запрещающее ограничение (ПЗО) удерживается, если значение заданной метрики на наборе данных превышает установленный порог.

Данная работа является продолжением работ [17, 18], в которых была представлена реализация верификатора ЗО, а также поиск кортежей, нарушающих зависимость. В данной работе мы продолжим развивать тему запрещающих ограничений и реализуем верификатор ПЗО. Также мы добавим интеграционное тестирование верификатора и алгоритма для поиска ПЗО, представленного в работе “Fast Approximate Denial Constraint Discovery” [6] и реализованного Иваном Морозко [16].

1. Постановка задачи

Данная работа является продолжением работ [17, 18], в которых отсутствовала верификация приближенных запрещающих ограничений.

Целью работы является реализация алгоритма для эффективной верификации ПЗО, а также интеграционное тестирование верификатора и алгоритма для поиска ЗО.

1. Провести обзор предметной области и систематизировать применимые к ПЗО метрики;
2. Реализовать подсчет метрик, применимых к приближенным запрещающим ограничениям;
3. Реализовать алгоритм верификации ПЗО в платформе Desbordante;
4. Добавить возможность использования верификатора приближенных запрещающих ограничений в Python;
5. Реализовать интеграционное тестирование верификатора и алгоритма для поиска ПЗО;

2. Обзор

Перед разработкой алгоритма верификации приближенных запрещающих ограничений необходимо определить, какие метрики корректно применять в контексте ЗО.

В качестве основного источника для метрик стала работа “Fast Approximate Denial Constraint Discovery” [6], на основе которой реализован алгоритм поиска ПЗО Иваном Морозко [16], поскольку без реализации используемых там метрик невозможно построить интеграционное тестирование. Так как удалось найти малое количество случаев применения метрик в контексте ЗО, то большинство рассматриваемых метрик будут относиться к приближенным функциональным зависимостям (Approximate Functional Dependencies, AFD).

Ниже приведены определения метрик, взятых из различных источник. Адаптация метрик под контекст ЗО метрик будет приведена в конце раздела. Необходимые определения, используемые для их описания можно найти в разделе 3.

Метрика g_1

Начнем с метрики g_1, так как это наиболее часто упоминаемая и используемая метрика.

Большинство приведенных ниже статей опираются на работу “Approximate Dependency Inference from Relations” [9]. Здесь метрика g_1 обозначает долю кортежей, нарушающих зависимость и определяется следующим образом:

$$g_1(X \to A, r) \coloneqq \frac{|G_1(X \to Y, r)|}{|r|^2},$$

где $G_1(X \to A, r) \coloneqq \{(u, v) \mid u, v \in r, u[X] = v[X], \quad u[A] \neq v[A]\}$ — множество исключений, $X \to Y$ — функциональная зависимость (functional dependency).

Работы “Extending Desbordante with Probabilistic Functional Dependency Discovery Support” [5] и “Mining approximate functional dependen-

cies as condensed representations of association rules" [8] опираются на предыдущую статью и определяют метрику g_1 аналогично.

В статье "Measuring Approximate Functional Dependencies" [11], приводится сравнительный анализ метрик для приближенных функциональных зависимостей. Данная работа также опирается на [9], но здесь метрику g_1 определяют другим способом:

$$g_1(\mathbf{X} \to \mathbf{Y}, R) \coloneqq 1 - \frac{|G_1(\mathbf{X} \to \mathbf{Y}, R)|}{|R|^2},$$

что эквивалентно предыдущим определениям, за исключением инвертированности значения относительно середины отрезка $[0, 1]$.

В работе "Approximate functional dependencies: a comparison of measures and a relevance focused tool for discovery" [14] метрика g_1 определяется совсем иначе:

$$g_1(\mathbf{X} \to \mathbf{Y}, R) \coloneqq 1 - \frac{|G_1(\mathbf{X} \to \mathbf{Y}, R)|}{|R|^2 - |R|},$$

данное определение схоже с определением из статьи [11], однако нормировка в знаменателе осуществляется по $|R|^2 - |R|$, что меняет смысл метрики: мы отбрасываем все пары нарушений где первый и второй кортеж в паре равны.

В еще одной статье, посвященной статье приближенным ограничениям "Efficient Discovery of Approximate Dependencies" [10] напрямую не вводят метрику g_1, а называют свое определение "небольшой адаптацией g_1" ("we use a slight adaptation of the well-established g_1 error"), которое также основанно на метрике g_1 из уже упомянутой статьи [9]:

$$e(X \to A, r) = \frac{|\{(t_1, t_2) \in r^2 \mid t_1[X] = t_2[X] \wedge t_1[A] \neq t_2[A]\}|}{|r|^2 - |r|},$$

Стоит заметить, что данная метрика схожа с метрикой, определяемой в статье [14], за исключением инвертированности.

Изучая статьи, посвященные приближенным функциональным зависимостям, можно заключить, что невозможно найти единообразия

даже в пределах одной метрики. Далее рассмотрим упоминание метрики g_1 в контексте ЗО.

В работе "Approximate Denial Constraints" [1] авторы не вводят непосредственно метрику g_1, однако определяют схожую ей функцию, определяющую нормированное число пар кортежей, удовлетворяющих зависимости:

$$f_1(D, S_\varphi) = \frac{|\{(t,t') \mid t,t' \in D,\, \{t,t'\} \models \varphi\}|}{|D|(|D|-1)},$$

где D — это таблица с исходными данными, $\{t,t'\} \models \varphi,\ \{t,t'\}$ — пара кортежей, на которых зависимость удерживается.

Легко заметить, что данное определение эквивалентно определению статье [14], только записано в другой форме — через множество кортежей, удовлетворяющих зависимости.

Также не обойдем стороной статью, на которой основывается сам алгоритм поиска ПЗО. В работе "Fast Approximate Denial Constraint Discovery" [6] авторы вводят определение, аналогичное определению из статьи [10].

$$g_1(\varphi, R) \coloneqq \frac{|\{(s,t) \mid s,t \in R\ , s \neq t, (s,t)\ violates\ \varphi\}|}{|R|^2 - |R|},$$

Таким образом можно заключить, что единообразие случилось только в статьях, посвященных ПЗО. Также отметим, что во всех вышеперечисленных метриках использовались пары кортежей, нарушающих зависимость, что дает возможность адаптировать эти метрики под контекст ЗО, так как мы уже умеем находить нарушения.

Метрика g_1'

Еще одна метрика, которую рассматривают в статьях, посвященных приближенных примитивам — метрика g_1'.

В работе [14] ее определяют так:

$$g_1'(\mathbf{X} \to \mathbf{Y}, r) \coloneqq \frac{|G_1(\mathbf{X} \to \mathbf{Y}, r)|}{|r|^2 - |r|},$$

она повторяет метрику g_1 из той же работы, за исключением того, что она инвертирована.

В уже упомянутой статье [11] метрику вводят по-другому:

$$g'_1(\mathbf{X} \to \mathbf{Y}, R) := 1 - \frac{|G_1(\mathbf{X} \to \mathbf{Y}, R)|}{|R|^2 - \sum_w R(w)^2},$$

где $R(w)$ — количество вхождений w в таблице R (frequency of w in R). Данное определение более точно описывает верхнюю границу $|G_1(\mathbf{X} \to \mathbf{Y}, R)|$, ее величина не более $|R|^2 - \sum_w R(w)^2$.

Таким образом, можем заключить, что все перечисленные метрики из статей по функциональным зависимостям различаются либо инвертированностью, либо знаменателем. Однако все они имеют общую часть — множество нарушений $G_1(X \to Y)$, что также позволяет использовать их для ЗО.

Метрика g_2

Далее рассмотрим метрику g_2, которая также часто упоминается в статьях о приближенных зависимостям.

В статье [9] метрика g_2 определена как доля кортежей, участвующих в нарушениях.

$$g_2(X \to A, r) = \frac{G_2(X \to A, r)}{|r|},$$

где $G_2(X \to A, r) = |\{u \mid u \in r, \exists v \in R : u[X] = v[X], u[A] \neq v[A]\}|$ — множество кортежей, входящих хотя бы в одну нарушающую пару.

Также данная метрика упоминается в статье по сравнению функциональных зависимостей [11], однако здесь авторы приводят другое толкование метрики:

$$g_2(\mathbf{X} \to \mathbf{Y}, R) = 1 - \sum_{w \in G_2(\mathbf{X} \to \mathbf{Y}, R)} p_R(w),$$

где $p_R(w) = \frac{R(w)}{|R|}$ — вероятность вхождения w в R.

Работы [5, 8], как и в случае с метрикой g_1 опираются на статью [9]

и определяют метрику g_2 таким же способом.

В статье [14] метрика g_2 определяется аналогично работе [9], за исключением инвертированности:

$$g_2(X \to A, r) = 1 - \frac{G_2(X \to A, r)}{|r|}.$$

Также затронем работы, затрагивающим приближенные запрещающие ограничения.

В статье [1] метрику g_2 не вводят на прямую, а определяют схожую функцию $f_2(D, S_\varphi)$ — доля "хороших" кортежей (таких, что они не образуют нарушение ни с одним другим кортежем из таблицы):

$$f_2(D, S_\varphi) = \frac{|\{t \mid t \in D,\ \nexists t' \in D : \{t, t'\} \not\models \varphi\}|}{|D|},$$

где $\{t, t'\} \not\models \varphi,\ \{t, t'\}$ — нарушение.

В работе [6] не стали использовать использовать метрику g_2 так как "она может быть слишком чувствительна на практике" ("can be too sensitive in practice"), ограничившись лишь метрикой g_1.

Отметим, что никаких специфичных для функциональных зависимостей объектов в определениях метрики g_2 не вводилось, поэтому, так мы также сможем использовать ее для ПЗО.

Метрика g_2'

Также рассмотрим метрику g_2', которая упоминается лишь в единственной статье [14] и определяется так:

$$g_2(X \to A, r) = \frac{G_2(X \to A, r)}{|r|},$$

что полностью совпадает с определением, приведенным в статье [9].

Метрика g_3

Рассмотрим последний тип метрик, которые возможно адаптировать под запрещающие ограничения. В работе [9] авторы, помимо g_1 и

g_2, вводят метрику g_3 и определяют ее так:

$$g_3(X \to A, r) = \frac{G_3(X \to A, r)}{|r|},$$

где $r,\ s$ — подмножества кортежей, $G_3(X \to A, r) = |r| - \max\{\, |s| \mid s \subseteq r,\ s \models X \to A \,\}$ — доля кортежей, которые необходимо удалить, чтобы зависимость начала удерживаться, а g_3 соответственно определяет долю таких кортежей.

Далее в работе [11] метрику g_3 определяют таким же образом, используя лишь различающиеся обозначения:

$$g_3(X \to Y, R) \coloneqq \max_{R' \in G_3(X \to Y, R)} \frac{|R'|}{|R|},$$

где $G_3(X \to Y, R) := \{\, R' \mid R' \subseteq R,\ R' \models X \to Y \,\}$.

В работе [1] не определяют явно метрику g_3, но определяют схожую функцию:

$$f_3(D, S_\phi) = \frac{\max_{D'}\{|D'| \mid D' \subseteq D, D' \models \phi\}}{|D|}$$

Несложно заметить, что данная функция аналогична введенной для функциональных зависимостей метрике в статье [9].

Также схожую метрику определяют в статье [13]:

$$e(X \to A, \varphi) = \frac{\min\{|s| \mid s \subseteq r \text{ and } X \to A \text{ holds in } r \setminus s\}}{|r|},$$

что равносильно метрике g_3, описанной в статье [9].

Авторы статей [5, 8] как и в случае с метриками g_1 и g_2 опираются на статью [9] и используют аналогичное определение.

Метрика g_3'

Метрика g_3' упоминается в статье [11] посвященной функциональным зависимостям и определяется так:

$$g_3'(X \to Y, R) \coloneqq \max_{R' \in G_3(X \to Y, R)} \frac{|R'| - |\mathrm{dom}_R(X)|}{|R| - |\mathrm{dom}_R(X)|},$$

где $\mathrm{dom}_R(X) = \{x|_Y \mid x \in R\}$, $x|_Y$ — ограничение кортежа x на множество атрибутов Y (то есть значения кортежа x только по атрибутам Y).

Данная метрика не подходит адаптации под запрещающие ограничения, так как использует специфичные только для функциональных зависимостей сущности (левую и правые части функциональной зависимости).

Метрика g_3' также упоминается в статьях, посвященных приближенным зависимостям включения (Approximate Inclusion Dependencies, AINDs). Так, например, в работе "Unary and n-ary inclusion dependency discovery in relational databases" [3] приводится адаптация метрики g_3 из статьи [9], однако она представляет лишь адаптацию под зависимости включения, и использовать в контексте ЗО мы ее не сможем.

Таким образом, так как метрика g_3' использует специфику функциональных зависимостей и зависимостей включения, то мы также в дальнейшем не будем рассматривать данную метрику.

Другие метрики

В статье [11], посвященной сравнению метрик для функциональных зависимостей помимо g метрик также обсуждается большое количестве метрик. Там упоминаются метрики FI, RFI, SFI, SFI_α, RFI^+, $pdep$, τ, μ, μ^+, g_1^S, RFI'^+. Однако все эти метрики возможно применять только в контексте функциональных зависимостей.

Итог

Поскольку запрещающие ограничения представляют собой зависимости другого рода, многие упомянутые метрики (g_3', FI, RFI и большинство метрик, определяемых в статье [11]) невозможно применить к ЗО напрямую, поэтому в данной работе мы ограничимся тремя: g_1, g_1' и g_2. Их применение к запрещающим ограничениям возможно благодаря тому, что они определяются только через множества нарушающих

пар кортежей — исключений, а именно такие пары мы уже умеем находить. Также отдельно напомним, что подсчет метрики g_3 является [6] NP-полной задачей, и ее подсчет — это весьма трудоемкая задача, поэтому данную метрику мы также не будем рассматривать. Более подробно о способах вычисления данной метрики в контексте ЗО рассказано в работе "Улучшение существующих пользовательских сценариев в Desbordante" [15].

Также необходимо определиться с конкретными определениями используемых метрик, поскольку зачастую в каждой статье используется необходимое авторам определение. Кроме этого необходимо навести порядок в определениях, чтобы уменьшить путаницу из-за разных обозначений.

Введем классы эквивалентности метрик (статей, в которых упоминаются данные метрики) с точностью до инвертированности относительно середины отрезка $[0, 1]$.

Для метрики g_1 с точностью до инвертированности получатся следующие классы: [5, 8, 9, 11] — здесь метрики нормируются по величине $|R^2|$, и класс [1, 6, 10, 14] — здесь метрики нормируются по величине $|R^2| - |R|$.

Теперь рассмотрим метрику g_1'. Определение в статье [14] попадает в один из классов эквивалентности для метрики g_1, поэтому данное определение мы не будем используем для метрики g_1'. Иное толкование для метрики g_1' приводится в статье [11]: здесь нормировка множества исключений происходит по величине $|R|^2 - \sum_w R(w)^2$. В случае с g_1' получается один класс эквивалентности из единственной статьи — [11].

Затем рассмотрим метрику g_2. Первый класс составляют статьи [1, 5, 8, 9, 14], которые определяют метрику с точностью до инвертированности следующим образом:

$$g_2(X \to A, r) = 1 - \frac{G_2(X \to A, r)}{|r|}.$$

второй класс составляет статья [11], где авторы расширили определение метрики:

$$g_2(\mathbf{X} \to \mathbf{Y}, R) = 1 - \sum_{w \in G_2(\mathbf{X} \to \mathbf{Y}, R)} p_R(w).$$

И наконец рассмотрим последнюю применимую к ЗО метрику g_2'. Единственное определение в статье [14] попадает в один из классов эквивалентности для метрики g_2, поэтому данное определение метрики для определения g_2' мы использовать не будем.

Всего получилось пять классов эквивалентности определений для метрик g_1, g_1' и g_2. Поэтому, по принципу Дирихле, уже имеющихся названий нам не хватит и необходимо придумать уникальные названия для имеющихся классов эквивалентности метрик.

Проиндексируем метрики согласно первым буквам фамилий авторов, в чьих статьях данные определения были впервые введены. Так, у нас получится два разных обозначения для метрики g_1, согласно двум классам эквивалентности — g_1^{KM}, для метрики впервые описанной в статье [9], а также g_1^{KN} — впервые описанной в работе [10]. Для единообразия также переименуем метрику g_1' в $g_1'^{\text{PWN}}$, определение которой впервые дано в [11]. Подобно g_1, для метрики g_2 у нас также есть два класса эквивалентности, поэтому у нас получится две метрики g_2 — g_2^{KM} из статьи [9], и g_2^{PWN} — метрики, впервые описанной в работе [11].

Для того, чтобы привести порядок во всем многообразии определений, мы разграничим само понятие метрики g и ее инвертированного значения — ошибки по метрике ρ_g. Будем считать, что за значение метрик $g_1^{KM}, g_1^{KN}, g_1'^{\text{PWN}}, g_2^{KM}, g_2^{\text{PWN}}$ мы берем долю "хороших" кортежей, соответственно, долю "плохих" кортежей (нарушений) будет отображать ошибка по заданной метрике.

В таблице 1 приведены адаптированные формулы для записи перечисленных метрик в контексте ПЗО. Также в ней приведены формулы для вычисления ошибок g_ρ по соответствующим метрикам. В качестве множеств G_1, G_2, G_3 для функциональных зависимостей в контексте ЗО используются аналогичные множества:

$G_1(\varphi, R) = \{(s,t) \mid s,t \in R\ , s \neq t, (s,t) \not\models \varphi\}$ — множество пар кортежей, которые нарушают зависимость.

$G_2(\varphi, R) = \{s \mid s \in R,\ \exists t \in R,\ (s,t) \not\models \varphi\}$ — множестве кортежей, которые встречаются в нарушениях.

$G_3(\varphi, R) = \{R' \mid R' \subseteq R,\ R \setminus R' \models \varphi\}$ — множество подмножеств R, таких, что при их удалении зависимость удерживается.

Таблица 1: Итоговые метрики

Номер	Список работ	Название метрики	Ошибка по метрике	Значение метрики	Применимо к ПЗО
1	[5, 8, 9, 11]	g_1^{KM}	$\rho_{g_1^{KM}} = \frac{\lvert G_1(\varphi,R)\rvert}{\lvert R\rvert^2}$	$1 - \rho_{g_1^{KM}}$	Да
2	[1, 6, 10, 14]	g_1^{KN}	$\rho_{g_1^{KN}} = \frac{\lvert G_1(\varphi,R)\rvert}{\lvert R\rvert^2 - \lvert R\rvert}$	$1 - \rho_{g_1^{KN}}$	Да
3	[11]	$g_1'^{\text{PWN}}$	$\rho_{g_1'^{\text{PWN}}} = \frac{\lvert G_1(\varphi,R)\rvert}{\lvert R\rvert^2 - \sum_w R(w)^2}$	$1 - \rho_{g_1'^{\text{PWN}}}$	Да
4	[1, 5, 8, 9, 14]	g_2^{KM}	$\rho_{g_2^{KM}} = \frac{\lvert G_2(\varphi,R)\rvert}{\lvert R\rvert}$	$1 - \rho_{g_2^{KM}}$	Да
5	[11]	g_2^{PWN}	$\rho_{g_2^{\text{PWN}}} = \sum_{w \in G_2(\varphi,R)} p_R(w)$	$1 - \rho_{g_2^{\text{PWN}}}$	Да
6	[5, 8, 9, 11]	g_3^{KM}	$\rho_{g_3^{KM}} = \min_{R' \in G_3(\varphi,R)} \frac{\lvert R'\rvert}{\lvert R\rvert}$	$1 - \rho_{g_3^{KM}}$	Да
7	[3, 11]	g_3', FI, RFI, SFI, SFI_α, RFI^+, *pdep*, τ, μ, μ^+, g_1^S, RFI'^+	—	см. статьи [3, 11]	Нет

Также отметим, что ошибка $\rho_{g_1^{KN}}$ равна метрике g_1 из раннее упомянутой работы [6] по поиску ПЗО, поэтому наш алгоритм верификации будет согласован с реализацией алгоритма поиска приближенных запрещающих ограничений, реализованного Иваном Морозко.

Все используемые в данной работе метрики будут определяться согласно таблице 1.

3. Предварительные сведения

Данные определения повторяют соответствующий раздел моего предыдущего отчета [17].

Перед тем, как начать реализацию алгоритма проверки приближенных ЗО, необходимо ввести основные определения и понятия. Все определения взяты из статьи [12].

Таблица 2: Основные понятия.

Понятие	**Значение**
R	таблица с исходными данными
$vars(R)$	конечное множество атрибутов (колонок) таблицы
$\|R\|$	количество строк в R
$t,\ s$	кортежи в R
$t.A$	значение атрибута A в кортеже t
предикат p	выражение вида: $s.A\ op\ t.B$, где $s,t \in R,\ A,\ B \in vars(R)$ и $op \in \{=\ ,\ \neq\ ,\ \geq\ ,\ >\ ,\ \leq\ ,\ <\}$
ЗО φ	конъюнкция предикатов: $\forall t,s \in R,\ t \neq s : \neg(p_1 \wedge \cdots \wedge p_m)$
$vars_{op}(\varphi)$	множество атрибутов гомогенного ЗО, для которых есть предикат с оператором op в ЗО

Определение 1 *Пара кортежей (s,t) будет считаться нарушением (violation), если все предикаты в φ верны ($(s,\ t)$ violates φ).*

Определение 2 *φ удерживается (holds) на R, если нет нарушений.*

Определение 3 *Предикат (predicate) называется гомогенным (homogeneous), если он представим в виде $s.A\ op\ t.A$ или $s.A\ op\ s.B$*

Пример. В таблице 3 приведен набор данных, в котором содержится информация о налоговой ставке для жителей разных штатов США. Вот некоторые правила, которые выполнятся на данном наборе данных: (1) SSN — потенциальный ключ, (2) $Zip \rightarrow State$ — функциональная зависимость и (3) для всех жителей одного штата у человека с большей заработной платой больше налоговая ставка.

Таблица 3: Налоговые ставки для жителей разных штатов США.

	SSN	Zip	Salary	FedTaxRate	State
t1	100	10108	3000	20%	NewYork
t2	101	53703	5000	15%	Wisconsin
t3	102	53703	6000	20%	Wisconsin
t4	103	53703	4000	10%	Wisconsin

Все вышеперечисленные правила можно записать в терминах запрещающих ограничений следующим образом:

1. $\varphi_1 : \forall t, s \in R,\ t \neq s : \neg(s.SSN = t.SSN)$
2. $\varphi_2 : \forall t, s \in R,\ t \neq s : \neg(s.Zip = t.Zip \wedge s.State \neq t.State)$
3. $\varphi_3 : \forall t, s \in R,\ t \neq s : \neg(s.State = t.State \wedge s.Salary < t.Salary \wedge s.FedTaxRate > t.FedTaxRate)$

Все вышеперечисленные ЗО являются гомогенными, т.е. содержат строчные гомогенные предикаты. Также приведем пример гетерогенного ЗО: $\varphi_4 : \forall t, s \in R,\ t \neq s : \neg(s.Salary < t.FedTaxRate)$. Отметим, что $vars_{=}(\varphi_3) = \{State\}$, $vars_{<}(\varphi_3) = \{Salary\}$ и $vars_{>}(\varphi_3) = \{FedTaxRate\}$.

4. Алгоритм

В данном разделе представлена реализация алгоритма верификации приближенных запрещающих ограничений.

Алгоритм верификации ПЗО полностью опирается на алгоритм 2, предназначенный для выявления кортежей, нарушающих зависимость. Более подробное описание базового алгоритма представлено в работе [18].

На вход алгоритму 1 подается таблица R, на которой задано запрещающее ограничение φ, множество пар кортежей, нарушающих зависимость V — полученных в результате работы алгоритма 2, и максимальное значение ошибки, превышение которой будет означать тот факт, что приближенное запрещающее ограничение не удерживается.

Algorithm 1: Верификация ПЗО

Input: Таблица R, ЗО φ, V − результат работы алгоритма 2, максимальная величина ошибки ρ_{max}, measure — тип метрики

Output: true или false

$\rho \leftarrow 0$
if $measure =$ "g_1^{KM}" **then**
  $\rho = \frac{|G_1(\varphi,R)|}{|R|^2}$
else if $measure =$ "g_1^{KN}" **then**
  $\rho = \frac{|G_1(\varphi,R)|}{|R|^2-|R|}$
else if $measure =$ "$g_1^{\prime PWN}$" **then**
  $\rho = \frac{|G_1(\varphi,R)|}{|R|^2-\sum_w R(w)^2}$
else if $measure =$ "g_2^{KM}" **then**
  $\rho = \frac{|G_2(\varphi,R)|}{|R|}$
else if $measure =$ "g_2^{PWN}" **then**
  $\rho = \sum_{w \in G_2(\varphi,R)} p_R(w)$
return $\rho \leq \rho_{max}$

Algorithm 2: Поиск нарушений ЗО

Input: Таблица R, ЗО φ

Output: Список пар записей таблицы R

$H \leftarrow$ пустая хэш-таблица
$V \leftarrow$ пустой список
foreach $t \in R$ **do**
    $v \leftarrow \pi_{\text{vars }=(\varphi)}(t)$
    if $v \notin H$ **then**
        $H[v] \leftarrow$ new OrthogonalRangeSearch()
    $\mathrm{L}, \mathrm{U}, \mathbf{L}', \mathbf{U}' \leftarrow$ CreateBothSearchRanges (t, φ)
    SearchRes $\leftarrow H[v]$.RangeSearch $(\mathbf{L}, \mathbf{U})$
    InvSearchRes $\leftarrow H[v]$.RangeSearch $(\mathbf{L}', \mathbf{U}')$
    $H[v]$.insert $\left(\pi_{\text{vars }(\varphi)\setminus \text{vars}=(\varphi)}(t)\right)$
    foreach $p \in SearchRes \cup InvSearchRes$ **do**
        $V \leftarrow V \cup \{(p, t)\}$
return V
Procedure `CreateBothSearchRanges`(r, φ)
    $\mathbf{L} \leftarrow (-\infty, \ldots, -\infty), \mathbf{U} \leftarrow (\infty, \ldots, \infty)$
    $\mathbf{L}' \leftarrow (-\infty, \ldots, -\infty), \mathbf{U}' \leftarrow (\infty, \ldots, \infty)$ `/* L, L' и U, U'`
        `индексированы предикатами неравенства` p_i`.` `*/`
    foreach *предикат* $p_i \in$ *предикаты неравенства* in φ **do**
        if $p_i.op$ *is* $\leq$ *or* $\leq$ **then**
            $U.C \leftarrow min\{U.C, r.D\}$
            $L'.D \leftarrow max\{L'.D, r.C\}$
        if $p_i.op$ *is* $>$ *or* $\geq$ **then**
            $L.C \leftarrow max\{L.C, r.D\}$
            $U'.D \leftarrow min\{U'.D, r.C\}$
        return $\mathbf{L}, \mathbf{U}, \mathbf{L}', \mathbf{U}'$
Procedure `InvertRange(L, U)`
    $\mathrm{U}' \leftarrow \mathrm{L}, \mathrm{L}' \leftarrow \mathrm{U}$
    изменить $-\infty$ на ∞ и ∞ на $-\infty$ в $\mathbf{U}'$ и $\mathbf{L}'$ соответственно
    return L', U'

Ниже представлено краткое описание работы алгоритма верификации ЗО.

H — хэш-таблица, в ключах которой хранятся проекции кортежей на некоторое количество атрибутов (кортеж значений), а в ключах хранится структура данных для ортогонального поиска (в нашем случае это k-d дерево). V — это множество, в которое мы будем помещать пары нарушений.

Алгоритм работы верификатора можно представить следующим образом:

1. В цикле проходим по всем кортежам из R. Для каждого кортежа t в R сначала проецируем его на все колонки, которые есть в предикатах с равенством, чтобы вычислить v — проекцию кортежа на колонки с равенством;

2. Если v еще не встречалось ранее, то помещаем ее в хэш-таблицу H и инициализируем;

3. Перед вставкой кортежа (точки) в дерево выполняем прямой и обратный поиск по дереву для того, чтобы найти точки, с которыми данный кортеж образует нарушение;

4. Добавляем данные нарушения в множество V;

5. По завершении просмотра всех записей в таблице возвращаем все уникальные пары нарушений — наше множество V.

5. Реализация

Как уже было сказано выше, если рассматривать работы, посвященные в той или иной мере метрикам (цитируемые статьи в разделе 2), то в совокупности они выглядят хаотично: нет единого подхода к определению метрики — каждый автор определяет ее так, как ему необходимо, названия метрик перемешиваются так, что под одной и той же метрикой могут подразумевать совершенно разные сущности, а порой и вовсе, разными метриками обозначать один и тот же объект. Дополнительно усугубляет ситуацию то, что нет различия между метрикой и инвертированной величиной — ошибкой, что также вносит неразбериху.

5.1. Алгоритм верификации ПЗО

Реализация алгоритма верификации приближённых запрещающих ограничений была выполнена в виде отдельного класса `ADCVerifier`, интегрированного в систему Desbordante. На этапе проектирования предполагалось, что данный алгоритм можно реализовать в пределах уже существующего `DCVerifier`. Однако не в пользу данного решения выступило то, что значительно увеличивалось бы количество передаваемых опций для алгоритма, а также усложнялся бы интерфейс взаимодействия с классом в Python.

В качестве опций для `ADCVerifier` задаются запрещающее ограничение ϕ, набор данных R, а также выбранная метрика и допустимый уровень ошибки $\rho_{\max}$.

Данный алгоритм переиспользует уже реализованный алгоритм верификации запрещающих ограничений `DCVerifier`, который подробно описан в работах [17, 18], который предназначен для верификации запрещающих ограничений и поиска нарушений.

В ходе выполнения верификации ПЗО:

1. `ADCVerifier` загружает данные в `DCVerifier`;
2. Запускает выполнение `DCVerifier` для поиска нарушений;

3. Вычисляет необходимую метрику основываясь на переданном типе `MeasureType`;

4. Определяет, удерживается ли ограничение в зависимости от переданного порога максимальной ошибки метрики.

Бóльшая часть работы выполняется на втором этапе, когда `DCVerifier` запускается на заданном ЗО для поиска нарушений. После этого `ADCVerifier` получает список всех нарушений, используя предоставленный интерфейс. С помощью этих нарушений в дальнейшем и подсчитываются метрики.

5.2. Метрики

Для управления метриками был создан класс `Measure`, агрегирующий подсчет метрики на основе переданных ему объекта верификатора и типа необходимой метрики.

Так как алгоритм поиска ПЗО построен на использовании метрики g_1^{KN}, то, для того чтобы провести интеграционное тестирование, необходимо было научиться вычислять данную метрику. Также дополнительно был реализован подсчет $g_1^{KM}, g_1'^{PWN}, g_2^{KM}, g_2^{PWN}$.

Для определения типов метрик было создано именованное множество `MeasureType`, содержащее в себе строковые представления для рассчитываемых типов метрик: `G1_KN`, `G1_KM`, `G1_PRIME_PWN`, `G2_KM`, `G2_PWN`.

Также `Measure` предоствляет унифицированный интерфейс для подсчета метрик: необходимо лишь передать необходимый тип метрики, который хочется подсчитать. Данное решение делает использование метрик гибким, а также упрощает добавление новых типов метрик: достаточно добавить новый тип метрики в множество `MeasureType`, а также добавить соответствующую функцию, вычисляющую метрику необходимым образом.

На диаграмме 1 представлено взаимодействие класса `ADCVerifier`. Зеленым цветом выделены новые добавленные сущности, серым цветом отмечены уже существовавшие на момент работы классы.

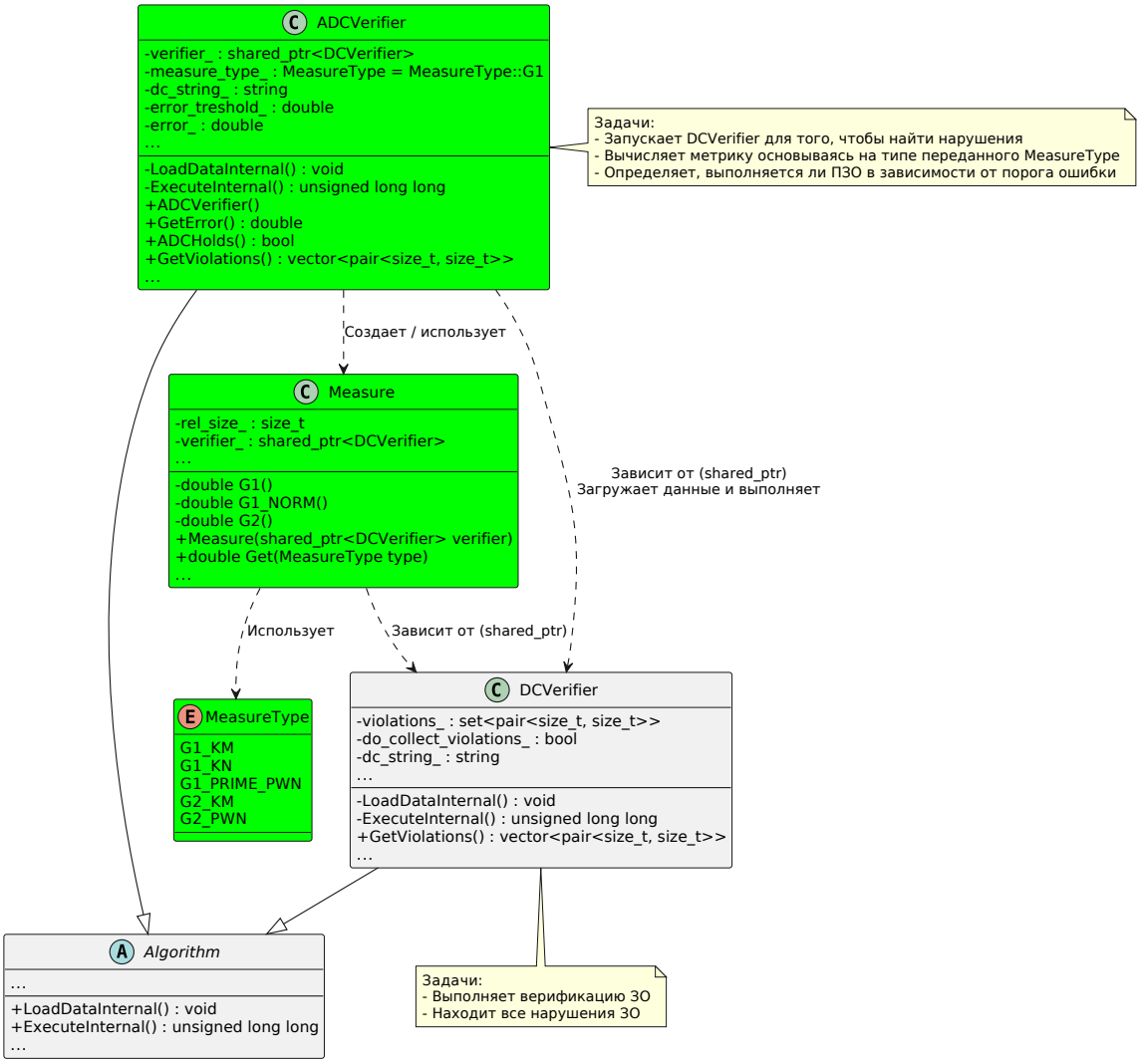

Рис. 1: Диаграмма класса алгоритма и вспомогательных классов.

5.3. Pybind11

С помощью библиотеки pybind11 [7] была добавлена возможность использования алгоритма верификации в Python. В качестве доступных опций алгоритм принимает указанное пользователем запрещающее ограничение, тип метрики, а также максимальный порог ошибки, превышение которой будет означать, что ПЗО не удерживается. Также определен интерфейс, с помощью которого пользователь может получать информацию о проверке ПЗО: ошибка по указанной метрике, и булево значение — удерживается ли приближенное запрещающее ограничение или нет.

Пример использования:

example.py:

```
import desbordante as db

dc = "!(s.Salary < t.Salary and s.State = t.State and
s.FedTaxRate > t.FedTaxRate)"
table = "datasets/taxes_3.csv"

# Setting max error threshold and measure type
max_treshold = 0.1
measure = "g1_KN"

# Creating a verificator and loading data in algortihm
verificator = db.adc_verification.algorithms.Default()
verificator.load_data(table=(table, ',', True))

# Algorithm execution
verificator.execute(denial_constraint=dc, error=max_treshold,
adc_error_measure=measure)

# Obtaining the result
result: bool = verifier.adc_holds()
error: float = verifier.get_error()

print("DC " + dc + " holds: " + str(result))
print("With error: " + str(error))
```

5.4. Интеграционное тестирование

Для проверки корректной работоспособности верификатора ПЗО и алгоритма по нахождению приближенных запрещающих ограничений [16], было реализовано интеграционное тестирование. Данное тестирование можно на разбить на два этапа:

1. Поиск всех приближенных запрещающих ограничений в наборе данных;

2. Верификация каждого найденного ограничения, если хотя бы одно ограничение не удерживается — тестирование не пройдено.

Поиск ПЗО осуществляется на определенном наборе данных с заданным максимальным значением ошибки, превышение которой будет указывать на то, что ограничение не будет включено в результат, поиск осуществляется только по метрике g_1^{KN}.

Верификация ПЗО осуществляется с теми же параметрами: на том же наборе данных, с тем же максимальным значением ошибки и также по метрике g_1^{KN}.

Заключение

По итогам учебной практики стало возможным использование алгоритма верификации приближенных запрещающих ограничений. По результатам выполнения работы:

1. Проведен обзор предметной области и систематизация применимых к ПЗО метрики;
2. Реализован подсчет метрик, применимых к приближенным запрещающим ограничениям;
3. Реализован алгоритм верификации ПЗО в системе Desbordante;
4. Добавлена возможность верификации приближенных запрещающих ограничений в Python;
5. Реализовано интеграционное тестирование верификатора и алгоритма для поиска ПЗО.

Исходный код[1] доступен на GitHub, PR 589. Изменения на стадии рассмотрения.

[1]https://github.com/Desbordante/desbordante-core/pull/589

Список литературы

[1] Livshits Ester, Heidari Alireza, Ilyas Ihab F., Kimelfeld Benny. Approximate Denial Constraints. — 2020. — 2005.08540.

[2] Chu Xu, Ilyas Ihab F., Papotti Paolo. Discovering denial constraints // Proc. VLDB Endow. — 2013. — Aug.. — Vol. 6, no. 13. — P. 1498–1509. — URL: https://doi.org/10.14778/2536258.2536262.

[3] De Marchi Fabien, Lopes Stéphane, Petit Jean-Marc. Unary and n-ary inclusion dependency discovery in relational databases // J. Intell. Inf. Syst. — 2009. — 02. — Vol. 32. — P. 53–73.

[4] 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.

[5] Extending Desbordante with Probabilistic Functional Dependency Discovery Support / Ilia Barutkin, Maxim Fofanov, Sergey Belokonny et al. // 2024 35th Conference of Open Innovations Association (FRUCT). — 2024. — P. 158–169.

[6] Fast approximate denial constraint discovery / Renjie Xiao, Zijing Tan, Haojin Wang, Shuai Ma // Proc. VLDB Endow. — 2022. — Oct.. — Vol. 16, no. 2. — P. 269–281. — URL: https://doi.org/10.14778/3565816.3565828.

[7] Jakob Wenzel, Rhinelander Jason, Moldovan Dean. pybind11 – Seamless operability between C++11 and Python. — 2017. — URL: https://github.com/pybind/pybind11.

[8] Kalavagattu Aravind. Mining Approximate Functional Dependencies as Condensed representations of association rules. — 2008.

[9] Kivinen Jyrki, Mannila Heikki. Approximate dependency inference from relations // Database Theory — ICDT '92 / Ed. by

Joachim Biskup, Richard Hull. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1992. — P. 86–98.

[10] Kruse Sebastian, Naumann Felix. Efficient discovery of approximate dependencies // Proc. VLDB Endow. — 2018. — Mar.. — Vol. 11, no. 7. — P. 759–772. — URL: https://doi.org/10.14778/3192965.3192968.

[11] Measuring approximate functional dependencies: a comparative study / Marcel Parciak, Sebastiaan Weytjens, Niel Hens et al. // VLDB J. — 2025. — Vol. 34, no. 4. — P. 56. — URL: https://doi.org/10.1007/s00778-025-00931-x.

[12] Rapidash: Efficient Detection of Constraint Violations / Zifan Liu, Shaleen Deep, Anna Fariha et al. // Proc. VLDB Endow. — 2024. — may. — Vol. 17, no. 8. — P. 2009–2021. — URL: https://doi.org/10.14778/3659437.3659454.

[13] Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies / Yká Huhtala, Juha Kärkkäinen, Pasi Porkka, Hannu Toivonen // The Computer Journal. — 1999. — Vol. 42, no. 2. — P. 100–111.

[14] Weytjens Sebastiaan. Approximate functional dependencies: a comparison of measures and a relevance focused tool for discovery : Master thesis / Sebastiaan Weytjens. — tUL, 2021.

[15] Артем Бурашников. Улучшение существующих пользовательских сценариев в Desbordante. — 2025. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/User%20Stories%20-%20Artem%20Burashnikov%20-%202024%20autumn.pdf (дата обращения: 2025-12-02).

[16] Иван Морозко. Реализация алгоритма поиска запрещающих ограничений в платформе Desbordante. — 2024. — URL: https://github.com/Desbordante/desbordante-core/blob/

main/docs/papers/FastADC%20-%20Ivan%20Morozko%20-%202024%20spring.pdf (дата обращения: 2025-11-09).

[17] Павел Аносов. Верификация DC в Desbordante. — 2024. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/DC%20Verification%20-%20Anosov%20Pavel%20-%202024%20spring.pdf (дата обращения: 2025-11-09).

[18] Павел Аносов. Улучшение алгоритма верификации запрещающих ограничений в Desbordante. — 2024. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/Constant%20DC%20Verification%20and%20Minimization%20-%20Anosov%20Pavel%20-%202024%20autumn.pdf (дата обращения: 2025-11-09).