Санкт-Петербургский государственный университет

Группа 21.Б08-мм

Обзор сопоставляющих зависимостей и алгоритма их поиска HyMD

Шлёнских Алексей Анатольевич

Отчёт по учебной практике
в форме «Теоретическое исследование»

Научный руководитель:
ассистент кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2024

Оглавление

Введение

Профилирование данных используется для изучения наборов данных. Изучив набор данных, его можно привести в более подходящий для дальнейшей работы вид, например, произвести дедупликацию. Также можно извлечь из данных некоторую полезную информацию.

К сфере профилирования данных относится поиск закономерностей, таких, например, как функциональные зависимости (functional dependencies, далее — ФЗ). Мы говорим, что на экземпляре отношения соблюдается функциональная зависимость из набора атрибутов X в набор атрибутов Y, если все записи из него, попарно совпадающие на атрибутах X, также попарно совпадают на атрибутах Y.

Иными словами, функциональные зависимости работают с равенством атрибутов. Но их не всегда можно использовать для дедупликации, ведь в реальных данных значения атрибутов у записей, представляющих один и тот же реальный объект, могут не быть равны, в том числе из-за ошибок или различных представлений.

Для задач очистки и исследования данных лучше подходит такая закономерность, как сопоставляющие зависимости (matching dependency, далее — СЗ).

СЗ являются обобщением ФЗ. ФЗ работают с равенством атрибутов, а СЗ — с их “схожестью”. При этом конкретное определение схожести зависит от самих данных: для двух чисел это может быть достаточно маленькое абсолютное значение их разности, а для двух строк — достаточно маленькое расстояние Левенштейна между ними.

Помимо этого, СЗ определены не на одном (как ФЗ), а на двух наборах данных.

Неформально говорим, что СЗ соблюдается на паре наборов данных, если в каждой паре из декартова произведения этих наборов данных, в которой значения сопоставленных заранее атрибутов схожи в заданнной степени (левая часть), значения других атрибутов (правая часть) тоже схожи в заданной степени.

Данный отчёт рассматривает алгоритм HyMD, предложенный в [2]. Этот алгоритм позволяет найти соблюдающиеся СЗ, определив их степени схожести для атрибутов в левой и правой частях.

1. Постановка задачи

Целью работы является обзор статьи [2]. Для этого были поставлены задачи ниже.

- Изучить сопоставляющие зависимости
- Рассмотреть алгоритм HyMD
- Рассмотреть тестирование этого алгоритма

2. Сопоставляющие зависимости

2.1. Определение СЗ

Определение 1. Метрика схожести (similarity metric) — двуместная функция, результатом которой является действительное число из диапазона $[0.0, 1.0]$, где 1.0 — максимальное сходство, а 0.0 — минимальное сходство.

Результат применения метрики схожести к паре значений назовём схожестью этих значений.

Два значения можно классифицировать как схожие или несхожие, применив к ним метрику схожести и оценив её результат относительно заданной границы решения.

Определение 2. Граница решения (decision boundary) — действительное число из интервала $[0, 1]$.

Классификатор схожести (similarity classifier) — пара из метрики схожести и границы решения.

Определим $\approx_i (a, b)$ следующим образом: $\approx_i$ — некоторая метрика схожести, например расстояние Левенштейна для строк или количество дней между двумя датами, а a, b — какие-то два значения, на которых она определена. То есть, $\approx_i (a, b)$ — это результат вычисления этой метрики для этих двух значений.

Определение 3. Если $\approx_i (a, b) \geq \lambda_i$, то значения a и b называем **схожими** относительно классификатора схожести $(\approx_i, \lambda_i)$, иначе — **несхожими** относительно классификатора схожести $(\approx_i, \lambda_i)$.

Таким образом, если граница схожести равна нулю, то мы считаем любые два возможных значения схожими, вне зависимости от метрики схожести.

Определение 4. Будем называть C множеством сопоставлений столбцов (column matches), если $C \subseteq R \times S \times \approx$, а его элемент, соответственно, сопоставлением столбцов (column match). Здесь $\approx = \{\approx_1, \approx_2, \ldots, \approx_m\}$ — множество метрик схожести, а R и S некоторые отношения (возможно $R = S$) и r и s их экземпляры, соответственно.

Введём следующие обозначения.

- $dom(Y)$ — домен атрибута Y.
- $t[a]$ — значение атрибута a в записи t некоторого отношения, где атрибут a присутствует.

Далее считаем, что $C = \{C_1, C_2, \ldots, C_m\} = \{(A_1, B_1, \approx_1), (A_2, B_2, \approx_2), \ldots, (A_m, B_m, \approx_m)\}$, при этом $\forall i \in (1...m) \approx_i$ — определена на $dom(A_i) \times dom(B_i)$.

Множество сопоставлений столбцов — это множество троек из атрибута отношения R, атрибута отношения S и метрики схожести, относительно которой алгоритм будет находить границу решения. Для i-го сопоставления столбцов функцией $\approx_i$ будет определяться схожесть двух значений записей a и b, $(a, b) \in r \times s$. Первое значение — $a[A_i]$, второе — $b[B_i]$.

Конкретные тройки в C определяются контекстом. Обычно множество из первых двух элементов троек образует инъективную функцию, а при $R = S$ — ограниченное тождественное отображение, но это необязательно.

Теперь определим сопоставляющую зависимость. Определяемую зависимость в данной части работы назовём φ.

Определение 5. Пусть дано множество сопоставлений столбцов $C \subseteq R \times S \times \approx$ на отношениях R, S и метриках схожести $\approx$. Тогда СЗ φ определяется как:

$$(\bigwedge_{i=1}^{m} R[A_i] \approx_{i,\lambda_i} S[B_i]) \to R[A_j] \approx_{j,\rho_j} S[B_j].$$

Здесь $(\bigwedge_{i=1}^{m} R[A_i] \approx_{i,\lambda_i} S[B_i])$ — левая часть, являющаяся конъюнкцией m классификаторов схожести столбцов. Классификатор схожести столбцов (column similarity classifier) — это пара сопоставления столбцов C_i и границы решения $\lambda_i \in [0.0, 1.0]$. Набор границ решения левой части обозначается как $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$. $R[A_j] \approx_{j,\rho_j} S[B_j]$ — правая часть, являющаяся одним классификатором столбца, состоящим из сопоставления столбцов C_j с метрикой схожести $\approx_j$ и границы решения $\rho_j \in [0.0, 1.0]$, $j \in (1 \ldots m)$.

Определение 6. Для пары записей $(a, b) \in (r \times s)$ **соблюдается левая часть** СЗ φ, если $\bigwedge_{i=1}^{m} a[A_i] \approx_{i,\lambda_i} b[B_i]$. При этом говорим, что СЗ φ **сопоставляет** записи в этой паре. Для пары записей $(a, b) \in (r \times s)$ **соблюдается правая часть** СЗ φ, если $\approx_j (a, b) \geq \rho_j$.

С помощью $a[A_i] \approx_{i,\lambda_i} b[B_i]$ обозначено $\approx_i (a[A_i], b[B_i]) \geq \lambda_i$, то есть "значения $a[A_i]$ и $b[B_i]$ схожи относительно классификатора схожести $(\approx_i, \lambda_i)$".

Определение 7. Для пары записей $(a, b) \in (r \times s)$ СЗ φ **соблюдается**, если из соблюдения левой части этой СЗ для этой пары следует соблюдение её правой части для этой же пары записей: $\bigwedge_{i=1}^{m} a[A_i] \approx_{i,\lambda_i} b[B_i] \implies \approx_j (a, b) \geq \rho_j$ (логическое следование). Если это не верно, то пара записей **противоречит** СЗ φ.

Определение 8. СЗ φ соблюдается для экземпляров r, s, если она соблюдается для всех пар из $(r \times s)$:

$$\forall (a, b) \in (r \times s)((\bigwedge_{i=1}^{m} a[A_i] \approx_{i,\lambda_i} b[B_i]) \implies a[A_j] \approx_{j,\rho_j} b[B_j])$$

Таблица 1: Примеры тривиальных и нетривиальных СЗ. Источник: [2]

тривиальные	нетривиальные
$A_{0.5} \rightarrow A_{0.5}$	$A_{0.5} \rightarrow A_{0.6}$
$A_{0.5} \rightarrow A_{0.4}$	
$A_{0.5}B_{0.6} \rightarrow A_{0.5}$	$A_{0.5}B_{0.6} \rightarrow A_{0.6}$
$A_{0.5}B_{0.6} \rightarrow A_{0.4}$	

Введём ещё одно обозначение. Если φ — СЗ, $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ — границы решения левой части, ρ_j — граница решения правой части, остальные параметры СЗ понятны из контекста, то обозначим СЗ как $\varphi(\lambda, \rho_j)$.

Также СЗ можем обозначать как $attr1_{0.5}attr2_{0.75} \rightarrow attr3_{0.9}$, если $R = S$, а каждый столбец сопоставлен сам себе, индексами в таком случае обозначены границы решения для атрибутов, слева от стрелки находятся атрибуты левой части с ненулевой границей решения, справа — атрибут правой части с границей решения правой части этой СЗ. Если у всех атрибутов левой части граница решения нулевая, то слева от стрелки пишется $\varnothing$.

2.2. Классификация сопоставляющих зависимостей

2.2.1. Тривиальные сопоставляющие зависимости

Если для пары записей (a, b) соблюдается левая часть СЗ, содержащая $a[A_j] \approx_{j,\lambda_j} b[B_j]$, то по определению $\approx_j$ $(a[A_j], b[B_j]) \geq \lambda_j$. Если при этом $\lambda_j \geq \rho_j$, то очевидно, что для этой пары записи соблюдается и правая часть СЗ.

Поэтому если для СЗ $\varphi(\{\lambda_1, \ldots, \lambda_j, \ldots, \lambda_m\}, \rho_j)$ верно $\lambda_j \geq \rho_j$, то она соблюдается вне зависимости от данных.

Определение 9. СЗ, у которой $\lambda_j \geq \rho_j$ называется **тривиальной** (trivial).

СЗ с нулевой границей решения правой части по определению тривиальны. При этом в нетривиальной СЗ граница левой части того же сопоставления столбца, что находится в правой части, может быть ненулевой, например, в $A_{0.5}B_{0.6} \rightarrow A_{0.6}$. В таблице 1 приведены другие примеры тривиальных и нетривиальных СЗ.

2.2.2. Минимальные сопоставляющие зависимости

Пусть имеются СЗ $\varphi(\lambda, \rho_j)$, $\varphi'(\lambda', \rho'_j)$. Если истинно $\forall i \in (1 \ldots m)\lambda'_i \leq \lambda_i$, тогда для любой пары записей, для которой соблюдается левая часть φ, соблюдается и левая часть φ'.

Определение 10. λ — **специализация** (specialization) λ', λ' — **обобщение** (generalization) λ. λ' **включает** (subsumes) λ.

При этом, если $\rho'_j \geq \rho_j$, то для вышеупомянутой пары записей φ соблюдается при соблюдении φ'.

Для СЗ $\varphi(\lambda, \rho_j)$, $\varphi'(\lambda', \rho'_j)$ говорим, что φ' **включает** φ, если

$$\varphi'(\lambda', \rho'_j) \preceq \varphi(\lambda, \rho_j) \equiv \forall i \in (1 \dots m) \lambda'_i \leq \lambda_i \wedge \rho'_j \geq \rho_j$$

Называем φ' **обобщением** φ, называем φ **специализацией** φ'.

$\preceq$ — бинарное отношение частичного порядка на всех СЗ с теми же сопоставлениями столбцов, что и φ.

$$A_{0.5} \rightarrow C_{0.6} \preceq A_{0.6} \rightarrow C_{0.6} \quad (1)$$
$$A_{0.5} \rightarrow C_{0.6} \preceq A_{0.5} B_{0.1} \rightarrow C_{0.6} \quad (2)$$
$$A_{0.5} \rightarrow C_{0.6} \preceq A_{0.5} \rightarrow C_{0.5} \quad (3)$$

Пусть Φ — множество СЗ. Тогда, согласно определению минимального элемента частично упорядоченного множества, СЗ $\varphi \in \Phi$ является минимальной относительно $\preceq$ (далее просто “минимальной”, minimal), если $\nexists \varphi' \in \Phi(\varphi' \neq \varphi \wedge \varphi' \preceq \varphi)$.

Если СЗ соблюдается на некоторых входных данных, то на них соблюдаются и все её специализации. Как известно из математики, имея некоторое конечное частично упорядоченное множество, можно выбрать все его минимальные элементы. То есть если мы имеем некоторое конечное множество СЗ, то описать все те его СЗ, которые соблюдаются, можно, выбрав минимальные СЗ из его соблюдающихся и сказав, что соблюдаются все их специализации из этого множества.

2.2.3. Естественные сопоставляющие зависимости

Рассмотрим сопоставление столбцов $C_i = (A_i, B_i, \approx_i)$.

Введём обозначение: $L_i = \{v \in \mathbb{R} : \exists (a, b) \in (r \times s) \approx_i (a, b) = v\}$ — множество результатов применения метрики схожести $\approx_i$ к парам значений столбцов, указанных в C_i.

Определение 11. Граница решения b называется естественной (natural) для сопоставления столбцов C_i, если $b \in L_i$.

Следующая граница после b, — это наименьшая естественная граница решения, большая b (если существует).

Предшествующая b граница решения, — это наибольшая естественная граница решения, меньшая b.

Положим, Φ — множество всех соблюдающихся на данных СЗ, соблюдается СЗ $\varphi \in \Phi$ с границами решений левой части $\lambda = \{\lambda_1, \lambda_2, \dots, \lambda_m\}$ и границей решения правой части ρ_j. Считаем, что $\lambda_k \neq 0.0$, при некотором $k \in (1 \dots m)$.

Возьмём СЗ φ' с теми же сопоставлениями столбцов, что и φ, той же границей решения правой части, теми же границами левой части, кроме λ_k, отличающуюся

границу назовём λ'_k. Пусть c — наибольшая граница решения из $L_k \cup \{0.0\}$, строго меньшая λ_k. $\lambda'_k = \frac{c+\lambda_k}{2}$. Тогда пары записей, для которых соблюдается φ, и пары записей, для которых соблюдается φ', совпадают, границы решения правой части этих СЗ также совпадают, а значит φ', как и φ, соблюдается. При этом φ' является обобщением φ, а значит φ не является минимальной.

Иными словами, СЗ, у которой одна из границ решения в левой части ненулевая, не может являться минимальной в Ф. То есть нельзя описать множество соблюдающихся на данных СЗ с помощью подмножества минимальных СЗ как выше, если только СЗ с $\lambda = \{0.0\}$, $\rho_j = 1.0$ не содержится в нём (то есть вообще все возможные СЗ соблюдаются).

Однако есть схожий способ описать все соблюдающиеся СЗ. Заметим, что для всех границ решения в левой части, строго большей некоторой естественной границы и меньшей или равной следующей за ней или единице, СЗ, у которых остальные границы решения левой части совпадают, сопоставляют одинаковые пары записей. Также это верно для такого же полуинтервала между нулём и наименьшей естественной границей решения, если нулевая граница решения не является естественной. Для покрытия всех соблюдающихся СЗ остаётся рассмотреть те из них, которые имеют нулевую границу решения в левой части для некоторого сопоставления столбцов, то есть имеют нулевую границу решения левой части, и те, которые не сопоставляют ни одной пары записей.

Определение 12. Назовём СЗ **естественной** для некоторых экземпляров r, s отношений R, S, если все её границы решения являются естественными.

Из изложенного выше следует, что чтобы описать все соблюдающиеся СЗ, нужно найти минимальные из множества естественных и соблюдающихся на данных СЗ. Для каждой такой СЗ можно заключить, что такие СЗ, у которых граница решения в правой части меньше или равна исходной и у которых i-тая граница решения в левой части больше предшествующей i-той в исходной СЗ, если она не наименьшая или не нулевая, и произвольная в противном случае, соблюдаются на входных данных. Также нужно добавить СЗ с произвольными границами решения в левой части и для каждого сопоставления столбцов в правой части с границей меньшей или равной первой естественной. Затем нужно добавить СЗ, которые не сопоставляют ни одну пару записей — граница решения в правой части у них произвольна. В частности, там может быть значение, превышающее наибольшую естественную границу. После добавления этих СЗ множество соблюдающихся СЗ будет описано полностью.

Последние два шага будет не делать, если вместо множества естественных СЗ рассматривать его объединение с множеством таких СЗ, у которых границы решения в левой части нулевые или естественные, а в правой — естественная или единичная. Обозреваемый алгоритм ищет минимальные из соблюдающихся СЗ именно такого

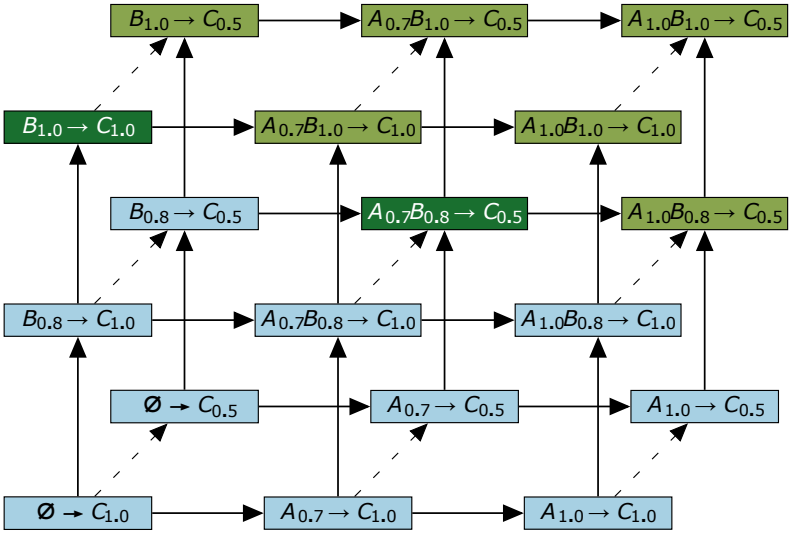

Рис. 1: Решётка границ решения для сопоставления столбцов C для одной из представленных в статье таблиц. A, B — сопоставления столбцов из левой части. Пунктирные и сплошные стрелки обозначают уменьшение и увеличение границы решения в правой и левой частях соответственно. Тёмно-зелёным обозначены минимальные соблюдающиеся СЗ, голубым — не соблюдающиеся СЗ, светло-зелёным — неминимальные соблюдающиеся СЗ. Источник: [2]

множества. Назовём это множество дополненным множеством естественных СЗ.

При этом стоит отметить, что проведение описанной выше процедуры на этом множестве всё ещё не опишет всё множество соблюдающихся на данных СЗ. Если есть сопоставления колонок, для которых 1.0 не является (наибольшей) естественной границей решения, то в итоговое множество нужно также добавить все СЗ, у которых граница решения левой части для этого сопоставления колонок больше наибольшей естественной, а граница решения правой части произвольна, и все их специализации.

2.3. Решётка границ решения

Пусть Ф — дополненное множество естественных СЗ на отношениях R, S, тогда бинарное отношение $\preceq$ задаёт частичный порядок на нём. Такой частичный порядок описывает решётку СЗ. На практике можно отдельно рассматривать решётки для каждого атрибута в правой части, ведь две зависимости с разными атрибутами в правой части не упорядочены относительно $\preceq$. Пример решётки для одного атрибута представлен на рис. 1, для упрощения в ней исключены тривиальные СЗ и СЗ с пересечением атрибутов.

По такой решётке рассматриваемый алгоритм обходит пространство поиска. В корне решётки для каждого атрибута всегда находится СЗ с нулевыми границами решения в левой части и максимальной границей решения в правой.

Для получения СЗ-наследника в решётке нужно увеличить одну границу из левой части до следующей (до наименьшей естественной, если она равна 0.0 и 0.0 не

является естественной границей решения) или уменьшить границу в правой части до предыдущей (до наибольшей естественной, если она равна 1.0 и 1.0 не является естественной границей решения).

Рассмотрение СЗ с разными сопоставлениями столбцов в правых частях можно представлять раздельно, ведь они не упорядочены друг относительно друга отношением $\preceq$, однако СЗ с одинаковыми частями сопоставляют одни и те же пары. Алгоритм HyMD этим пользуется и проводит вычисление для каждой левой части единожды.

Глубина СЗ определяется как длина самого короткого пути от корня до неё в решётке, вес дуги, обозначающей увеличение границы решения в левой части, равен 1, вес дуги, обозначающей уменьшение границы решения в правой части, равен 0. Все СЗ с разной глубиной независимы друг от друга, то есть они не специализируют и не обобщают одна другую. Это используется для распараллеливания поиска.

2.4. Интересность СЗ

Соблюдающихся на наборах данных СЗ может быть очень много, однако не все из них интересны для рассмотрения на практике. Авторы статьи предлагают различные критерии интересности, с помощью которых можно убрать некоторые СЗ из рассмотрения.

Определение 13. Мощность (cardinality) СЗ — количество сопоставлений столбцов, для которых граница решения в левой части СЗ ненулевая.

Мощность. СЗ со слишком большой мощностью предлагается отсекать, поскольку они более сложны для понимания. Максимальную мощность СЗ можно передать HyMD как параметр, и тогда он будет отсекать СЗ с большей мощностью. СЗ с большой мощностью также отсекаются алгоритмом при нехватке памяти.

Определение 14. Поддержка (support) СЗ на экземплярах r, s отношений R, S — количество пар из $r \times s$, для которых соблюдается левая часть СЗ.

Поддержка. Предлагается не рассматривать такие СЗ, у которых поддержка слишком низкая. Скорее всего, такие СЗ соблюдаются не потому, что они дают информацию о данных, а потому, что данных слишком мало, и в них не нашлось контрпримера. Из-за того, что поиск СЗ может использоваться для различных прикладных задач, могут потребоваться различные значения этого параметра, поэтому стандартные значения минимальной поддержки, которые предлагаются в статье, — это 1, если $r \neq s$, и $|r| + 1$, если $r = s$. Последнее обосновано нахождением среди пар записей таких, в которых стоят одни и те же записи, таких всегда $|r|$ штук, у них все схожести равны единице, а значит записи в таких парах будут сооставляться любой из рассматриваемых СЗ.

Отсутствие пересечения атрибутов (disjointness). Предлагается не рассматривать СЗ, у которых пара атрибутов из сопоставления столбца правой части присутствует среди пар атрибутов сопоставлений столбцов с ненулевой границей решения в левой части. Авторы статьи считают, что СЗ с пересекающимися атрибутами не имеют сценариев использования.

Величина границ решения (decision boundaries, value). Предлагается не рассматривать те СЗ, у которых сопоставлениям столбцов соответствует слишком низкая граница решения. Если две записи сопоставляются по некоторым значениям, то подразумевается, что два этих значения считаются одинаковыми (по смыслу), но если схожесть слишком мала, то, скорее всего, они такими не являются.

Количество границ решения (decision boundaries, number). Предлагается уменьшить количество рассматриваемых границ решения.

Не для всех сценариев использований важны точные границы решения. В статье приводятся две СЗ: $animal_{0.75} \to description_{0.989245}$ и $animal_{0.75} \to description_{0.990926}$, — которые в случае исследования данных дают нам только понимание, что значения в столбце *description* почти полностью схожи при соблюдении условия в левой части, и слишком высокая точность для этого не нужна.

Для уменьшения размера пространства поиска предлагается выбирать k равномерно распределённых значений из естественных границ решения.

3. Реализация

Авторы статьи опираются на два подхода к поиску зависимостей из других работ: обход решётки (lattice traversal, из [5]) и вывод из пар записей (inference from record pairs, из [3]). Первый работает с решёткой, задаваемой зависимостями и частичным порядком на них, классифицируя зависимости как соблюдающиеся или не соблюдающиеся. Второй обрабатывает все пары записей, определяя не соблюдающиеся зависимости, из который выводятся соблюдающиеся зависимости.

Для каждого из подходов предлагаются алгоритмы. Затем подходы совмещаются в алгоритме HyMD.

Здесь и далее рассматриваемые в некоторый момент выполнения алгоритма СЗ называются “кандидатами”.

3.1. Обход решётки

Решётка, задаваемая отношением частичного порядка $\preceq$, обходится с корня, в ширину. Начиная с самых общих СЗ, алгоритм проверяет все СЗ текущего уровня (level), отсекая неминимальные соблюдающиеся СЗ из пространства поиска. Хотя концептуально для каждой правой части своя решётка, для эффективности алгоритм проходит их вместе, обрабатывая все СЗ с одинаковыми левыми частями сразу. Более подробно проход описан в алгоритме 1 (рис. 2).

Здесь “уровень” СЗ определён как её глубина, далее это понятие будет переопределено.

Далее структура данных Ф, хранящая на данный момент интересные алгоритму СЗ, называется решёткой. Не путать с решёткой, задаваемой отношением $\preceq$ на рассматриваемом множестве СЗ.

Сначала алгоритм инициализирует решётку Ф самыми общими СЗ (строка 2). Затем алгоритм получает все кандидаты текущего уровня для их обработки (строка 5) с целью определить максимальные границы решения правой части, для которых СЗ с ними и левой частью от кандидата соблюдаются. Такие СЗ являются минимальными из-за отсечения по минимальности (детали далее). Также стоит упомянуть, что в отличие от строки 2, в строке 5 в обозначении $\varphi(\lambda, \rho)$ φ — информация о всех СЗ с некоторыми границами решения левой части, λ — границы решения левой части, ρ — границы решения правых частей сразу всех рассматриваемых кандидатов с границами решения левой части λ.

Далее кандидаты с обрабатываемыми левыми частями удаляются из Ф (строка 6). Перед их проверкой алгоритм получает максимальные границы решения правых частей из находящихся в Ф СЗ с границами решения левой части, являющимися обобщениями текущих (строка 7). В начале выполнения алгоритма, когда Ф становится пустым — это m нулей.

Алгоритм 1: Обход решётки

Входные данные: Экземпляры отношений r , s. Сопоставления столбцов C. Минимальные границы решения RHS ρ_{min}. Минимальная поддержка minSup.

Результат: Множество всех минимальных соблюдающихся на данных СЗ Φ.

```
1  m ← |C|;
2  Φ ← {φ(∅, (1.0, j))|j ∈ [1, m]};
3  l ← 0;
4  while l ≤ getMaxLevel(Φ) do
5      foreach φ(λ, ρ) ∈ getLevel(l, Φ) do
6          Φ ← Φ \ φ;
7          λ_lower ← getLowerBoundaries(λ, Φ);
8          ρ', σ ← validate(λ, ρ, r , s, C, λ_lower, ρ_min);
9          if σ < minSup then
10             markUnsupported(λ);
11         else
12             foreach ρ'_j ∈ ρ' do
13                 if ρ'_j > λ_j ∧
14                   ρ'_j ≥ ρ_min[j] ∧
15                   ρ'_j > λ_lower[j] then
16                     add(φ(λ, ρ'_j), Φ)
17             foreach i ∈ [1, m] do
18                 if canSpecializeLhs(λ, i) then
19                     λ' ← specializeLhs(λ, i);
20                     if isSupported(λ') then
21                         foreach ρ_j ∈ ρ do
22                             if ρ_j > λ'_j then
23                                 addIfMin(φ(λ', ρ_j), Φ);
24     l ← l + 1;
25 return Φ;
```

Рис. 2: Источник: [2]

Затем для СЗ с полученными левыми частями алгоритм вычисляет поддержку (эта процедура описана ниже) и все наибольшие границы решения левых частей для каждой левой части (строка 8). Если поддержка слишком мала, то левая часть отмечается как имеющая слишком маленькую поддержку и обход решётки прекращается для кандидатов с этой левой частью (строка 10). Иначе алгоритм рассматривает полученные СЗ с границами решения левой части λ и границами решения правых частей, полученными ранее (на рис. 2 это ρ') и проверяет их. Условие в строке 13 — проверка на тривиальность, в строке 14 — проверка на интересность (достаточную схожесть в правой части), в строке 15 — проверка на минимальность. Из-за порядка обхода решётки мы знаем, что в Ф находятся только те СЗ рассматриваемого множества СЗ, которые могут быть минимальными соблюдающимися на входных данных на данный момент. Если условие в строке 15 не выполняется, то получившаяся СЗ является специализацией некоторой СЗ, которая уже есть в Ф, то есть она или является специализацией СЗ, о которой уже известно, что она соблюдается, либо ещё не исключён вариант, что она такой станет.

Если все три вышеупомянутых условия соблюдаются, СЗ добавляется в Ф. Также мы знаем, что эта СЗ будет в итоговом результате.

Далее для следующего этапа выводятся специализации, не покрытые ранее проверенными СЗ. Они создаются посредством увеличения каждой границы решения левой части на следующую (строка 19), если это возможно (строка 18). Граница решения правой части для СЗ остаётся неизменной, и получившаяся СЗ добавляется в Ф, если она является минимальной относительно имеющихся в Ф (строка 23).

Рассматривая предыдущие уровни, алгоритм отметил некоторые левые части как имеющие слишком маленькую поддержку (строка 10). Очевидно, что СЗ, левые части которых являются специализацией какой-то из таких, также неподдерживаемы. Такие СЗ в Ф не добавляются (строка 20).

После обработки СЗ текущего уровня алгоритм обрабатывает СЗ следующего уровня. Когда в уровнях отсутствуют новые кандидаты, все минимальные соблюдающиеся СЗ найдены и алгоритм возвращает множество СЗ, которые находятся в Ф (строка 25).

3.2. Вывод из пар записей

Пусть имеется СЗ $\varphi(\lambda, \rho_j)$, пара записей (a, b), множество сопоставлений столбцов $C = \{C_1, C_2, \ldots, C_m\}$. Если СЗ сопоставляет пару записей (a, b), но $\rho_j >\approx_j (a, b)$, то эта пара записей противоречит СЗ. Предложенный далее алгоритм опирается именно на эту идею.

Пусть Ф — набор кандидатов, которые мы предполагаем соблюдающимися, (a, b) — пара записей, не рассмотренная ранее. Эта пара записей позволяет ослабить предпо-

Алгоритм 2: Вывод из пар записей

Входные данные: Экземпляры отношений r , s. Сопоставления колонок C. Минимальные границы решения RHS ρ_{min}.
Результат: Множество всех минимальных соблюдающихся на данных СЗ Φ.

```
1  m ← |C|;
2  Φ ← {φ(∅, (1.0, j))|j ∈ [1, m]};
3  foreach (r_k, s_l) ∈ r × s do
4      sim ← calculateSimilarity(r_k, s_l, C);
5      violated ← findViolated(sim, Φ);
6      foreach φ(λ, ρ_j) ∈ violated do
7          Φ ← Φ \ φ;
8          if sim[j] ≥ ρ_min[j] ∧ sim[j] > λ_j then
9              ρ'_j ← sim[j];
10             addIfMin(φ(λ, ρ'_j), Φ);
11         foreach i ∈ [1, m] do
12             if canSpecializeLhs(λ, i, sim[i]) then
13                 λ' ← specializeLhs(λ, i, sim[i]);
14                 if ρ_j > λ'_j then
15                     addIfMin(φ(λ', ρ_j), Φ);
16 return Φ;
```

Рис. 3: Источник: [2]

ложение, убрав все такие кандидаты, которые сопоставляют записи в паре, но которым эта пара записей противоречит. Если кандидат убран, то можно сделать вывод, что кроме него самого также не соблюдаются его обобщения. При этом кандидаты, являющиеся его специализациями, могут быть соблюдающимися и даже минимальными среди соблюдающихся. Последними могут быть кандидаты с ближайшими к несоблюдающимся границами решения, которым пара не противоречит (будет уточнено далее). То есть для того, чтобы ослабить предположение в соответствии с парой, нужно убрать из Φ все кандидаты, ей противоречащие, и добавить кандидаты с ближайшими границами решения, которые не противоречат ей или ранее рассмотренным парам и могут быть минимальными.

Алгоритм 2 (рис. 3) описывает такой процесс вывода СЗ. Аналогично обходу решётки, Φ инициализируется самыми общими СЗ (строка 2). Затем алгоритм проходит все пары из декартова произведения отношений (строка 3). Для каждой пары алгоритм вычисляет схожести согласно сопоставлениям столбцов C (строка 4). Далее алгоритм находит СЗ, которым пара противоречит, в решётке как описано выше. Каждая такая СЗ удаляется из решётки (строка 7), а затем из неё выводятся новые кандидаты, являющиеся её специализациями, посредством уменьшения границы решения в правой части (строки 8–10) или увеличения одной из границ решения в левой части (строки 11–15).

Так как СЗ с такими же границами решения левой части, что и текущая, может со-

блюдаться для более низкой границы решения правой части, алгоритм рассматривает СЗ с такими же границами решения левой части и границей решения правой части, пониженной до схожести значений из противоречащей пары записей. Если эта СЗ интересна и нетривиальна (строка 8), а также минимальна, то она добавляется в решётку (строка 10). Остальные специализации создаются посредством специализации левой части. Она специализируется так, что получившаяся СЗ не сопоставляет записи из текущей пары. Для этого алгоритм увеличивает границу решения в левой части для каждого сопоставления столбцов до следующей за соответствующей сопоставлению столбцов схожестью для этой пары (строка 13). Если кандидат с получившимися границами решения левой части и прошлой границей правой части нетривиален (строка 14) и минимален, то он добавляется в решётку (строка 15). Граница решения в правой части остаётся прошлой, поскольку она была выведена из рассмотренной ранее пары записей. Если после специализации левой части СЗ всё ещё сопоставляет записи из той пары, то мы не можем повысить границу решения в правой части, ведь тогда та пара будет противоречить получившейся СЗ. Если же новая левая часть не сопоставляет записи той пары, то может быть возможна более высокая граница решения в правой части, но СЗ с такой границей решения, если она есть, будет выведена на другом пути в решётке. Алгоритм завершается после обработки всех пар записей — в Ф содержатся все минимальные соблюдающиеся СЗ из рассматриваемых.

3.3. Гибридный подход: HyMD

Оба подхода, рассмотренных выше, имеют сильные и слабые стороны: вывод из пар записей лучше масштабируется при увеличении количества атрибутов, а обход решётки — при увеличении количества записей во входных экземплярах отношений. При этом подходы имеют сходство в схеме поиска — и тот, и другой начинают с самого общего предположения (то есть “все СЗ соблюдаются”), а затем его специализируют. Обход решётки проверяет кандидаты шаг за шагом, проходя решётку непрерывно. Напротив, вывод из пар записей проверяет пары записей на противоречие СЗ, в процессе чего много кандидатов за раз могут быть определены как несоблюдающиеся.

Алгоритм HyMD совмещает рассмотренные выше подходы. После приведения полученных данных в более удобную форму (об этом далее) HyMD начинает выполнение с вывода из пар записей, но вместо перебора всех пар сразу, он выбирает их одну за другой, проверяя перебор на эффективность. Когда перебор пар записей становится неэффективным, алгоритм меняет фазу на обход решётки. После обработки СЗ текущего уровня, алгоритм меняет фазу обратно на вывод из пар записей (если уровень был последним, то алгоритм завершает работу). Чтобы сделать вывод из пар записей более эффективным, во время проверки СЗ на текущем уровне решётки алгоритм собирает рекомендации — пары записей, которые вывод из пар записей проверит в

первую очередь. Таким образом, фазы алгоритма обмениваются кандидатами и рекомендациями, поддерживая друг друга.

3.3.1. Вывод из пар записей

Для использования алгоритма 2 в качестве метода в гибридной конфигурации в него вносится несколько изменений. Первое — набор СЗ Ф инициализируется не при каждом вызове метода, а только в начале выполнения. Второе — в алгоритм добавляется параметр, являющийся набором рекомендаций, которые передаёт фаза обхода решётки. Эти рекомендации проверяются перед основным циклом в строке 3 этого алгоритма.

Третье изменение — считаются *количество сравнений* пар записей (количество итераций цикла в строке 3) и *количество уточнённых* СЗ, то есть тех, которые были удалены из Ф (строка 7). Отношение этих чисел (*количество уточнённых*/*количество сравнений*) уменьшается по мере выполнения. Это отношение отражает эффективность фазы вывода из пар записей. Когда оно становится слишком низким, фаза меняется. Авторы предлагают изначально установить эту границу в значение 0.01, а затем делить её на 2 после каждого перехода на стадию вывода из пар записей.

Четвёртое изменение — алгоритм не обрабатывает те пары записей, у которых все схожести для сопоставлений столбцов совпадают с некоторой ранее обработаной парой. Такие пары записей не позволят уточнить ни одной СЗ, поэтому их можно не обрабатывать. При описанном использовании такая ситуация более вероятна, чем если бы алгоритм работал сам по себе — рекомендации от фазы обхода решётки содержат те пары записей, которые ещё не были обработаны в основном цикле (иначе они не были бы рекомендованы, об этом далее), а значит в основном цикле они могут встретиться повторно.

3.3.2. Обход решётки

Алгоритм 1 также модифицирован для использования в HyMD.

Набор СЗ Ф инициализируется фазой вывода из пар записей. Поэтому строка 2 удаляется, а Ф становится параметром.

HyMD переключается с фазы обхода в фазу вывода из пар записей сразу после проверки уровня в решётке. Здесь эффективность фазы не замеряется, поскольку если переключать фазу было не нужно, фаза вывода заканчивается достаточно быстро. Для реализации переключения фазы оператор цикла `while` в строке 4 заменяется на оператор `if` с тем же условием, и в конце алгоритма происходит переход на другую фазу.

Для увеличения эффективности фазы вывода из пар записей фаза обхода решётки собирает рекомендации, которые являются парами записей, из которых был сделан

вывод о несоблюдении некоторой СЗ. Фаза вывода ещё их не проверяла, так как все СЗ, находящиеся в Ф, соблюдаются на уже проверенных парах. Авторы статьи отмечают, что из таких пар записей со значительной вероятностью может быть сделан вывод о несоблюдении других СЗ из Ф, поэтому в качестве рекомендаций следует использовать именно их. Функция в строке 8 естественным образом находит такие пары. Алгоритм может объединить все такие наборы пар для всех СЗ уровня и передать получившийся набор фазе вывода из пар записей после завершения обработки уровня.

В строке 5 функция `getLevel` возвращает все кандидаты некоторого уровня из решётки. Все кандидаты независимы, поэтому их можно проверять параллельно, то есть цикл в строке 5 можно распараллелить. Однако поскольку вывод из пар записей отсекает части решётки, количество кандидатов каждой глубины сильно уменьшается, что снижает эффективность параллельного выполнения. Поэтому уровень кандидата переопределяется как его мощность, так как кандидатов с одинаковой мощностью больше, чем кандидатов с одинаковой глубиной. Кандидаты мощности 1 включают в себя все кандидаты глубины 1 и ещё некоторые, кандидаты мощности 2 включают в себя все кандидаты глубины 2 с неединичной мощностью и ещё некоторые другие и так далее. Например, в решётке на рис. 1 есть 2 кандидата глубины 1 ($A_{0.7} \to C_{1.0}$, $B_{0.8} \to C_{1.0}$), но 4 кандидата мощности 1 ($A_{0.7} \to C_{1.0}$, $B_{0.8} \to C_{1.0}$, $B_{1.0} \to C_{1.0}$, $A_{1.0} \to C_{1.0}$). Стоит обратить внимание, что здесь рассматриваются только левые части, так как правые части расчитываются алгоритмом на ходу. При этом в паре кандидатов одной и той же мощности один может специализировать другой (например, $B_{0.8} \to C_{1.0}$ и $B_{1.0} \to C_{1.0}$). Чтобы находить только минимальные СЗ, обобщения нужно проверять перед их специализациями. Следовательно HyMD сначала анализирует каждый уровень на предмет специализаций, проверяя кандидаты попарно, чтобы отсрочить проверку специализаций. В примере выше это значит, что проверка границ решения левой части $A_{0.7}$ завершается перед $A_{1.0}$, а $B_{0.8}$ — перед $B_{1.0}$. При этом, например, $A_{0.7} \to C_{1.0}$ и $B_{1.0} \to C_{1.0}$ могут проверяться параллельно. Анализ на специализации накладывает дополнительные расходы, но выигрыш от распараллеливания перекрывает их, как показывают эксперименты (о них далее).

Перед проверкой кандидаты уровня (не специализирующие друг друга) группируются по границам решения левой части λ. Получается набор границ решения правых частей для некоторых сопоставлений столбцов ρ. Затем алгоритм вычисляет максимальные границы решения правых частей для полученных сопоставлений столбцов ρ' при границах решения левой части λ, чтобы потом их проверить (условия в строках 12–15 в Алгоритме 1). Процесс вычисления описан далее.

Даже с вышеописанными изменениями все СЗ, найденные Алгоритмом 1 соблюдающиеся и минимальные. Поэтому HyMD завершается в фазе обхода решётки после обработки всех кандидатов.

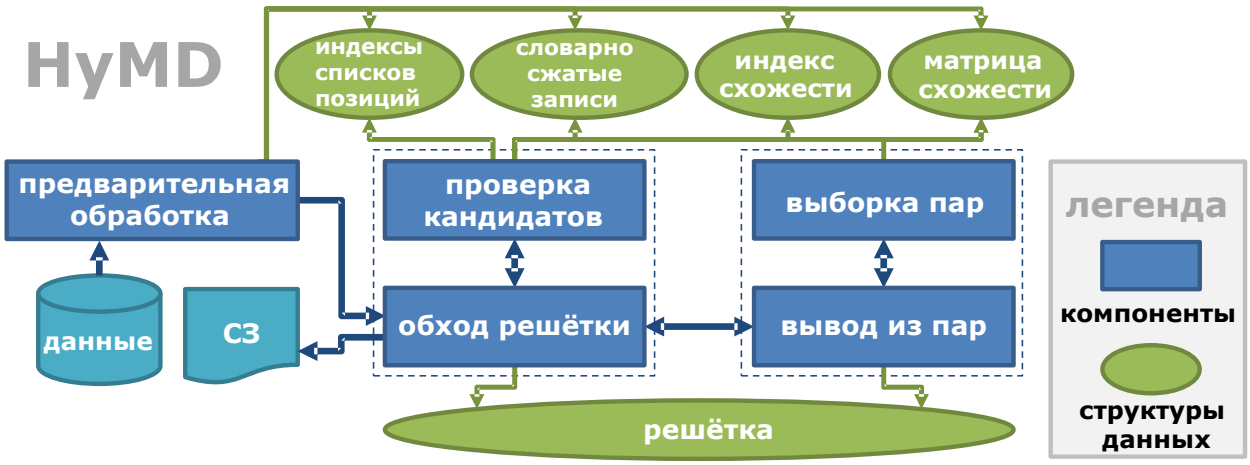

Рис. 4: Источник: [2]

4. Подробности работы HyMD

На рис. 4 изображены компоненты и структуры данных, которые использует HyMD в процессе своей работы. Индексы списков позиций (PLI, Position List Index) и словарно сжатые записи (Dictionary Compressed Records) представляют собой сжатые версии исходных данных, а матрица схожести (Similarity Matrix) и индекс схожести (Similarity Index) хранят данные о схожести значений. Решётка является префиксным деревом, хранящим текущие кандидаты. Компоненты “обход решётки” и “проверка кандидатов” реализуют обход решётки, а компоненты “вывод из пар” и “выборка пар” реализуют вывод из пар записей. Стрелки показывают взаимодействие между компонентами и доступ компонентов к разным структурам данных.

Далее в секции 4.1 описываются 4 структуры данных, изображённых сверху, и этап предварительной обработки, который их создаёт. Затем в секции 4.2 предлагается структура данных “решётка” для обработки промежуточных кандидатов. В секции 4.3 описывается алгоритм проверки кандидатов.

4.1. Предварительная обработка

Перед нахождением СЗ HyMD преобразует экземпляры отношений r и s в 4 структуры данных. Это происходит на этапе предварительной обработки.

Индексы списков позиций. Входные данные сначала преобразуются в индексы для каждой колонки, такие, что каждый номер места в списке, которым является этот индекс, уникального значения колонки указывает на кластер позиций исходного экземпляра отношения, в которых это значение встречается. Далее эти индексы будут обозначаться как π. Для их создания на этапе предварительной обработки входные данные читаются по одной записи, и каждая позиция записывается в кластер, соответствующий значению в ней, в PLI соответствующего атрибута. Поскольку для

поиска зависимостей не требуются сами значения, а только списки позиций с одинаковыми значениями, во всех PLI хранятся только эти списки. При этом номер места каждого кластера в PLI служит идентификатором значения. Это стандартный подход к проверке зависимостей, использовавшийся ещё в алгоритме TANE [5].

Словарно сжатые записи. Создавая PLI, HyMD также сжимает записи, заменяя каждое значение в записи индексом его кластера в PLI соответствующего атрибута. Получающиеся словарно сжатые записи нужны потом для сравнения значений записей и проверки кандидатов. Структура данных реализована как массив массивов записей, где по идентификатору (индексу в этом массиве) записи находится массив идентификаторов значений её атрибутов.

Матрицы схожести. Нахождение СЗ требует вычисления схожестей у пар значений из двух столбцов каждого сопоставления столбцов. Поскольку получать схожести приходится часто и поскольку их вычисление затратно, авторы статьи предлагают вычислить их заранее и хранить нужные в матрицах схожести (по одной на каждое сопоставление столбцов). Одно измерение матрицы — идентификаторы значений первой колонки, второе — идентификаторы значений второй колонки. Идентификаторы значений выводятся из PLI, которые были вычислены ранее, и обозначают различные значения колонок. Элементами матрицы являются схожести для сопоставления столбцов.

Для создания матриц схожести алгоритму требуются исходные значения и идентификаторы значений. Поэтому матрицы схожести создаются после PLI, но перед тем, как ключи к PLI отбрасываются.

Поскольку вычисления схожестей ресурсозатратны и независимы друг от друга, алгоритм распараллеливает их. Кроме этого, все значения проверяются на равенство перед выполнением функции схожести, так как многие из значений не только схожи, но и равны. При этом в худшем случае для каждого сопоставления столбцов нужно будет вычислить $|r| * |s|$ схожестей. Если входные экземпляры слишком велики для хранения в памяти, предлагается использовать техники секционирования, чтобы вычисление схожести проводилось только для тех пар значений, которые схожи в достаточной степени. Значениям, которые не были сравнены, в матрицах схожести соответствует 0.0. Стоит отметить, что применимость каждого подхода к секционированию зависит от конкретной метрики. Кроме того, секционирование — это приближение, из-за чего не все СЗ могут быть найдены при её использовании, поэтому в экспериментах оно не используется.

Чтобы уменьшить размер матриц схожести, схожести ниже заранее заданного порога не хранятся. Не хранящаяся схожесть неявно предполагается равной нулю. Поскольку эти схожести не хранятся, матрицы схожести становятся разрежёнными, что позволяет уменьшать использование памяти, так как теперь эту структуру данных можно представить как массив словарей, где на позициях в массиве, соответствующим

идентификаторам значений в первом столбце, хранятся словари, ключами которых являются идентификаторы значений второго столбца, а значениями — схожести значений двух столбцов.

Индекс схожести. Матрица схожести позволяет определить схожесть двух значений относительно сопоставления столбцов. Во время поиска СЗ часто выполняется ещё одна операция — относительно сопоставления столбцов для некоторого идентификатора значения в столбце первой таблицы найти все записи второй таблицы с достаточной схожестью в её столбце из сопоставления относительно метрики в этом сопоставлении. Для её поддержки авторы предлагают индекс, называемый “индекс схожести”. В нём каждый идентификатор значения столбца из первой таблицы в сопоставлении столбцов сопоставлен всем схожестям между значением, на которое этот идентификатор указывает, и различными значениями атрибута в сопоставлении столбцов записей из второй таблицы. Каждая схожесть, в свою очередь, указывает на список идентификаторов записей из второй таблицы, для которых схожесть изначального значения и значения атрибута из сопоставления у этой записи больше или равна этой схожести. Таким образом, этот индекс инвертирует матрицы схожести, а значит вычисляется с их использованием.

4.2. Решётка

Решётка компактно хранит все текущие кандидаты и обеспечивает выполнение различных операций. Из-за специфики этих операций она реализована как префиксное дерево, где корень обозначает СЗ с нулевыми границами решения в левой части, путь от корня до вершины определяет границы решения в левой части, а пометки для каждой вершины являются границами решения для правых частей зависимостей, которые не были определены как несоблюдающиеся или отсечены и которые имеют LHS, соответствующие этой вершине.

Изначально это префиксное дерево содержит только корневую вершину, и в ней все границы решения правой части равны 1.0. Затем эти кандидаты специализируются и дерево растёт. Далее описаны поддерживаемые операции.

`add` вставляет СЗ в решётку, проходя путь в префиксном дереве согласно границам решения левой части вставляемой СЗ. Если вершина на пути отсутствует, то она создаётся. В конце пути эта операция помечает вершину границей решения правой части.

`addIfMin` вставляет СЗ в решётку, используя `add`, если эта СЗ минимальная в решётке. Минимальность проверяется поиском обобщений в решётке с помощью поиска в глубину среди СЗ, у которых границы решения в левой части у одинаковых сопоставлений столбцов меньше или равны вставляемым, но все не совпадают, то есть среди левых частей, являющихся обобщением левой части данной СЗ и не совпадаю-

Алгоритм 3: Проверка кандидатов (процедура validate)

Входные данные: Границы решения λ, ρ. PLI *таблицы* r $\pi[r]$. Экземпляры отношений r, s. Сопоставления столбцов C. Максимальные границы решения левой части и минимальные границы решения правой части λ_{lower}, ρ_{min}. Индекс схожести simIndex.

Результат: Максимальные границы решения ρ'. Поддержка σ .

```
1  ρ', σ ← ρ, 0;
2  if |λ|= 0 then
3      return getMinSims(ρ, simIndex), |r|·|s|;
4  if |λ|= 1 then
5      foreach value, r' ∈ π[r][i] do
6          s' ← getSimRecs(value, λ_i, simIndex[i]);
7          σ ← σ + |r'|· |s'|;
8          foreach r_k, s_l ∈ r'× s' do
9              ρ' ← computeMaxRhs(r_k,s_l,ρ',λ,λ_lower,ρ_min);
10 else
11     foreach rep, r' ∈ groupByValue(π[r], λ, C) do
12         s' ← getSimRecs(rep[j], λ_j, simIndex[j]);
13         foreach λ_i ∈ λ \ λ_j do
14             s' ← s' ∩ getSimRecs(rep[i], λ_i, simIndex[i]);
15         σ ← σ + |r'|· |s'|;
16         foreach r_k, s_l ∈ r'× s' do
17             ρ' ← computeMaxRhs(r_k,s_l,ρ',λ,λ_lower,ρ_min);
18 return ρ', σ;
```

Рис. 5: Источник: [2]

щих с ней.

`findViolated` получает из решётки все СЗ, которым противоречит данная пара записей. Для этого решётка проходится в глубину по всем СЗ, которые сопоставляют записи из пары, и все границы правых части проверяются. Те СЗ, которым пара противоречит, собираются в возвращаемом наборе СЗ.

`getLowerBoundaries` определяет максимальные границы решения правой части среди всех находящихся в решётке обобщений данной СЗ. Эта операция обходит все обобщения левой части СЗ поиском в глубину, находя максимальные границы решения правых частей среди всех обойдённых обобщений. Если все эти границы становятся равными 1.0, обход прекращается сразу, поскольку они не могут стать больше.

`getLevel` получает все СЗ данного уровня. Уровень СЗ может быть определён как её глубина или её мощность. Операция использует поиск в глубину для нахождения всех вершин с указанной глубиной или мощностью.

4.3. Проверка кандидатов

Алгоритм проверки (рис. 5) кандидатов различает 3 случая в зависимости от мощности СЗ: 0, 1, больше.

Если мощность равна нулю (строка 2), то СЗ сопоставляет все пары и её поддержка максимальна — произведение мощностей входных экземпляров отношений. Максимальные границы решения ρ' в таком случае являются минимальными схожестями для соответствующих сопоставлений колонок и могут быть получены линейным проходом индекса схожести (строка 3).

У СЗ с мощностью 1 (строка 4) достаточно рассмотреть только одно сопоставление столбцов, здесь это сопоставление столбцов с индексом i. Алгоритм итерируется по всем идентификаторам значений ($value$) и их кластерам позиций (r') в столбце первого экземпляра отношения из этого сопоставления столбцов, используя его PLI, обозначенный как $\pi[r][i]$ (строка 5). Для каждого идентификатора значения из экземпляра отношения r Алгоритм 3 собирает все позиции записей из экземпляра отношения s в s', в которых схожесть этого значения и значения атрибута второго отношения больше или равна λ_i, используя индекс схожести (строка 6). Поскольку СЗ сопоставляет все выбранные записи из пар, к поддержке добавляется $|r'| * |s'|$ (строка 7). Затем HyMD проверяет все пары записей, на которые указывают идентификаторы из $r' \times s'$ (строка 8), и сохраняет минимальные схожести для каждого сопоставления столбцов в ρ' (строка 9) — эти минимальные схожести являются максимально возможными границами решения для СЗ с данной левой частью.

Если у данных СЗ мощность больше единицы (строка 10), то получить данные из какого-то одного PLI, в отличие от случая с мощностью 1, не получится. Поэтому записи первой таблицы сначала группируются по комбинациям значений атрибутов, для каждого из которых есть сопоставление столбцов, у которого граница решения ненулевая, с этим атрибутом в качестве атрибута первого экземпляра отношения. Для каждой такой комбинации (rep) находится набор идентификаторов записей из первой таблицы, у которых значения в этих атрибутах равны значениям в этой комбинации (r') путём пересечения PLI столбцов, из которых эти значения были взяты (строка 11). В итоге получается новый временный PLI, уже для нескольких атрибутов сразу, в нём rep — значения, r' — кластеры. Для нахождения таких записей из второй таблицы, в которых все значения атрибутов в левой части достаточно схожи с соответствующими значениями в rep (s'), HyMD пересекает множества достаточно схожих с каждым значением из левой части в соответствующих атрибутах записей (строки 12–14). Затем, схоже со случаем для мощности 1, пары записей, на которые указывают пары из $r' \times s'$, используются для определения значений поддержки σ и максимальных границ решения правых частей ρ' (строки 15–17).

Таблица 2: Производительность HyMD на различных наборах данных. Источник: [2]

набор данных	столбцы [#]	строки [#]	размер	LHS [#]	СЗ [#]	пред. [сек]	поиск [сек]	выполнение
adult	15	32 561	3.5 МБ	10^5	91	2.0	12.8	14.8 сек
restaurant	6	864	63.0 КБ	10^6	7	1.1	0.7	1.8 сек
VTTS	14	10 042 044	588.0 МБ	10^6	76	269.8	11 957.7	3.4 ч
NCVoters*	12	466 388	45.7 МБ	10^8	472	13.9	133.5	2.5 мин
NCVoters	12	32 413 515	3.1 ГБ	10^9	435	1 627.7	32 338.2	9.5 ч
hospital*	10	9 342/4 830	2.7 МБ	10^9	188	9.5	3.0	12.5 сек
HGI	14	4 830	0.7 МБ	10^{10}	289	10.0	5.3	15.3 сек
PCM	14	9 342	2.0 МБ	10^{10}	557	9.3	6.4	15.7 сек
CORA	16	1 879	367.0 КБ	10^{12}	2 001	5.6	379.4	6.5 мин
flight	38	1 000	187.0 КБ	10^{12}	14 170	0.9	65.4	1.1 мин
hospital	134	9 342/4 830	2.7 МБ	10^{22}	3 544	30.3	83.5	1.9 мин

5. Эксперименты

В секции экспериментов, авторы оценивают различные аспекты алгоритма HyMD. Сначала оценивается его время работы на разных наборах данных (представлены в секции 5.1). Затем измеряется эффект различных деталей имплементации и параметров конфигурации на времени работы и количестве результатов (в секции 5.2). Далее исследуется масштабируемость алгоритма при увеличении числа строк и сопоставлений столбцов (описано в секции 5.3).

Условия эксперимента. HyMD реализован на языке Java 1.8 на платформе Metanome [1]. Эксперименты были проведены на OpenJDK 64-bit Server 1.8.0_151 JVM и системе CentOS 6.8 64-bit. На используемом устройстве было установлено 128 ГБ оперативной памяти и 2 ЦПУ Intel Xeon E5-2650 частотой 2 ГГц, в экспериментах для параллельной обработки были использованы все 16 ядер.

Стандартная конфигурация алгоритма. Если не указано иное, для каждого сопоставления столбцов используются естественные границы решения, а минимальные границы решения для левых и правых частей устанавливаются в значение 0.7. Минимальная поддержка устанавливается в 1, если рассматриваются два разных экземпляра отношений и в размер экземпляра + 1, если отношение одно. Для запусков с одним отношением все столбцы сопоставлены сами себе, для двух — используется некоторое заранее заданное сопоставление. Стандартная метрика схожести — схожесть по Левенштейну. HyMD вычисляет все схожести для каждого сопоставления столбцов, то есть приближения не используются.

5.1. Наборы данных и время работы

В таблице 2 перечислены семь наборов данных, используемых в экспериментах: *adult*, содержащий общую информацию о людях; *restaurant*, описывающий рестораны США; *VTTS* с ERP-данными из системы SAP R3; *NCVoters* о предпочтениях избира-

телей в Северной Каролине; *hospital*, состоящий из таблиц Hospital General Information (HGI) и Preventive Care Measures (PCM), предоставляющих информацию о больницах США; *CORA* с библиографической информацией; *flight* с данными о рейсах в США.

Для *hospital* используется сопоставление схем, созданное вручную, поскольку обе таблицы этого набора данных содержат записи с информацией о больницах и атрибуты имеют чёткие соответствия. Для набора данных *CORA* вместо схожести Левенштейна используется схожесть Монжа-Элкана [4], а количество границ решения для каждого сопоставления столбцов ограничивается до 5 вместо использования всех естественных границ решения, поскольку *CORA* содержит особенно много различных, но очень похожих значений. При этом количество LHS для *CORA* остаётся одним из самых больших.

Количество возможных LHS СЗ в таблице 2 показывает, что пространства поиска огромны, а найденные СЗ составляют лишь малую их часть. Если учесть, что для каждой возможной левой части находится в среднем $\frac{\text{количество столбцов}}{2}$ СЗ —для каждого сопоставления столбцов в правой части выбираем наибольшую границу решения, при которой СЗ всё ещё соблюдается, убираем неинтересные, — то количество реально обнаруженных СЗ ничтожно мало. Это говорит о том, что критерии интересности, которыми в рассматриваемом эксперименте являются только минимальная граница решения, количество границ решения и отсутствие пересечения атрибутов, являются эффективными. Использование более строгих значений критериев интересности, чем те, которые были использованы в экспериментах, вполне применимо для многих случаев использования СЗ и значительно снижает количество обнаруженных зависимостей. Например, увеличение минимальной границы решения до 0.9 и минимальной поддержки до размера отношения, умноженного на 1.05, уменьшает количество обнаруженных СЗ с 3544 до 320 для *restaurant* и с 14170 до 2295 для *flight*. После обнаружения СЗ могут быть ранжированы также по критериям интересности, чтобы показать пользователю только наиболее значимые из них.

В столбцах *pre* и *disc* приведено время работы HyMD с разбивкой на время предварительной обработки (*pre*) и время поиска (*disc*). Они показывают, что время предварительной обработки, в частности, вычисление схожестей, на самом деле, значительно превосходит время поиска для наборов данных с большим количеством различных значений и с СЗ, которые легко найти с помощью любой из двух стратегий поиска СЗ. В последнем столбце приведено общее время выполнения HyMD.

Несмотря на относительно небольшой размер, перечисленные наборы данных представляют реальную проблему для алгоритмов поиска СЗ из-за большого пространства поиска кандидатов. Однако HyMD с лёгкостью обработал все наборы данных за несколько секунд или минут. Только наборы *VTTS* и *NCVoters* заняли больше нескольких минут, что вполне приемлемо, учитывая их большой размер — 588.0 МБ и 3.1 ГБ соответственно. Наборы данных *flight* и *hospital* имеют не только наибольшее

количество сопоставлений столбцов, но и содержат наибольшее количество СЗ из всех наборов данных, но время их обработки всё ещё приемлемо, поскольку вывод из пар записей работает очень хорошо, когда в наборе данных всего несколько тысяч записей. Наборы данных *adult*, *VTTS* и *NCVoters*, напротив, особенно длинные, но из-за небольшого количества сопоставлений столбцов и СЗ обход решётки обрабатывает их достаточно эффективно.

Измерения, приведённые в таблице 2, также показывают, что количество обнаруженных СЗ может быть очень большим. Например, в наборе данных *flight* объёмом всего 187 КБ HyMD обнаружил более 14 тысяч СЗ, что намного больше, чем может обработать вручную специалист по анализу данных. Но они всё равно полезны, так как некоторые алгоритмы, использующие СЗ для решения определённых задач, могут обрабатывать их автоматически. Для случаев, требующих ручного анализа найденных СЗ, таких как исследование данных, предлагается определить более жёсткие пороги отсечения по интересности. В экспериментах были заданы чрезмерно свободные пороги, чтобы показать, на что способен алгоритм. Например, при требовании максимальной мощности, равной трём атрибутам, минимальной поддержки, равной размеру набора данных, умноженному на 1.5, минимальной границы решения, равной 0.9, и максимального количества границ решения, равного 100, все наборы результатов сократились до менее чем 100 СЗ.

5.2. Подробные эксперименты

В обозреваемой работе предлагается применить гибридную стратегию поиска зависимостей для поиска СЗ. Поскольку пространство поиска СЗ значительно больше, чем пространство поиска ФЗ, а обход решётки, соответственно, более затратен, не очевидно, что гибридный поиск действительно является эффективным для данной задачи. Поэтому раздельно оцениваются три подхода к поиску: обход решётки, вывод из пар записей и гибридный поиск. Также детально оцениваются функции уровня и отсечение по пересечению атрибутов, поскольку оба эти параметра важны для эффективности обхода решётки. В четвёртом, заключительном, эксперименте оценивается влияние выбора различных мер схожести на время работы HyMD и количество найденных СЗ.

Гибридный поиск. HyMD сочетает в себе обход решётки с выводом из пар записей. Поскольку обе стратегии способны самостоятельно обнаружить все СЗ, в статье сравнивается их производительность с производительностью их гибридной комбинации. Для *restaurant* время поиска составило 37.0 с (обход решётки), 1.2 с (вывод из пар записей) и 0.7 с (гибридный подход). Для *hospital* — 65.1 с (обход решётки), 11.7 с (вывод из пар записей) и 2.8 с (гибридный подход). Аналогичное соотношение наблюдалось и для других наборов данных. Поскольку обход решётки плохо масшта-

Таблица 3: Производительность HyMD с разными функциями схожести. Источник: [2]

набор данных	функция	СЗ [#]	пред. [сек]	поиск [сек]	выполнение
NCVoters*	Левенштейна	472	13.9	133.5	2.5 мин
	Монжа-Элкана	1 219	20.9	382.2	6.7 мин
CORA	Левенштейна	997	2.1	145.7	2.5 мин
	Монжа-Элкана	2 001	5.6	379.4	6.5 мин

бируется с увеличением числа границ решения, их число в данном эксперименте было ограничено до 10 и 5 соответственно. Поскольку оба набора данных относительно коротки, стратегия вывода из пар записей явно выигрывает у стратегии обхода решётки по времени. Однако гибридная стратегия выигрывает у обоих подходов, даже у оптимального для данного набора данных, за счёт того, что подходы дополняют друг друга.

Функция уровня. Ранее была затронута тема использования разных функций уровня для получения кандидатов СЗ для проверки: глубины и мощности. При использовании мощности можно параллельно проверять большее число кандидатов, но при этом возникает необходимость в дополнительном шаге — упорядочивании кандидатов. Поэтому функция уровня в HyMD является настраиваемой. В экспериментальных условиях статьи было измерено время поиска 4.6 с (глубина) и 3.0 с (мощность) для *hospital*, 13.6 с (глубина) и 12.8 с (мощность) для *adult* и последующие аналогичные времена работы, меньшие при использовании мощности для других наборов данных. Повышенная степень распараллеливания, достигнутая на 16 ядрах, компенсировала дополнительные затраты на упорядочивание.

Отсечение по пересечению атрибутов. В разделе 2.4 было предложено исключить СЗ с пересекающимися атрибутами из пространства поиска, несмотря на то, что они не являются тривиальными. При отключении отсечения по пересечению атрибутов HyMD позволяет найти больше СЗ за счёт увеличения времени поиска. При включённом отсечении для *hospital* алгоритм находит 188 СЗ за 3.0 секунд, при отключённом — 974 СЗ за 18.7 секунд. Для *adult* алгоритм находит 91 СЗ за 12.8 секунд, при отключённом — 138 СЗ за 52.7 секунд. Таким образом, время поиска растёт быстрее, чем количество результатов.

Функции схожести. HyMD может использовать любую функцию схожести, способную сравнивать значения заданных сопоставлений столбцов. Алгоритм также позволяет смешивать функции схожести и использовать разные функции для одних и тех же пар столбцов. Поэтому авторы статьи оценивают влияние выбранных функций схожести на время поиска и количество обнаруженных СЗ. В этом эксперименте сравниваются две функции схожести: Левенштейна и Монжа-Элкана, — на *NCVoters* и *CORA*. Результаты приведены в таблице 3.

Поскольку вычисление схожести Монжа-Элкана более затратное, чем вычисление

схожести Левенштейна, время предварительной обработки при использовании этой функции схожести увеличивается. Однако в процессе поиска используются предварительно вычисленные схожести, поэтому ресурсозатратность выбранных функций схожести не влияет на него. Но на него влияет количество СЗ, которые могут быть обнаружены с полученными схожестями. Функция Монжа-Элкана менее чувствительна к преобразованиям символов, чем функция Левенштейна, и даёт более высокие схожести для различных значений. Поэтому количество соблюдающихся СЗ при использовании функции Монжа-Элкана увеличивается, что заметно сказывается на времени поиска. В частности, это влияние намного больше, чем влияние затрат на вычисление функции схожести. Таким образом, чем больше значений может сопоставить функция схожести, тем выше затраты на поиск СЗ.

5.3. Масштабируемость

На время выполнения HyMD влияют три свойства входных наборов данных: количество записей, количество сопоставлений столбцов и количество границ решения. С ростом объёма данных, как правило, увеличиваются все три показателя. Поэтому авторы исследуют время работы HyMD при увеличении каждого из них на наборах данных *CORA* и *NCVoters**.

Количество записей. Измерения начинаются для обоих наборов данных с 10% случайной выборки записей, далее добавляются ещё 10% записей на каждом шаге измерения. Количество сопоставлений столбцов и границ решения равно соответствующим количествам для предыдущей выборки данных, чтобы масштабировалось только количество записей. При добавлении новых записей в набор данных количество минимальных СЗ может как увеличиваться, так и уменьшаться. Оно увеличивается, если добавленные записи образуют противоречащие ранним СЗ пары и выведенные СЗ рассматриваются на следующих шагах алгоритма. Оно уменьшается, если соблюдающиеся СЗ вытесняются в области решётки, которые отсекаются ("исчезают").

На рис. 6 показан результат этого эксперимента. Как видно из него, на время работы HyMD влияют различные факторы, хотя изменяется только количество записей: в общем случае большее количество записей увеличивает затраты на предварительную обработку и проверку кандидатов, то есть увеличиваются затраты на вычисление схожестей, структуры индексов становятся больше, вывод из пар записей должен сравнивать больше пар записей, обход решётки должен проходить по большему числу кандидатов на более высоких уровнях решётки, а проверка кандидатов должна проверять больше значений. С другой стороны, новые пары записей, противоречащие СЗ, могут позволить фазе вывода из пар записей быстрее выявлять несоблюдающиеся СЗ и уменьшить количество минимальных СЗ, которые необходимо обнаружить (и проверить).

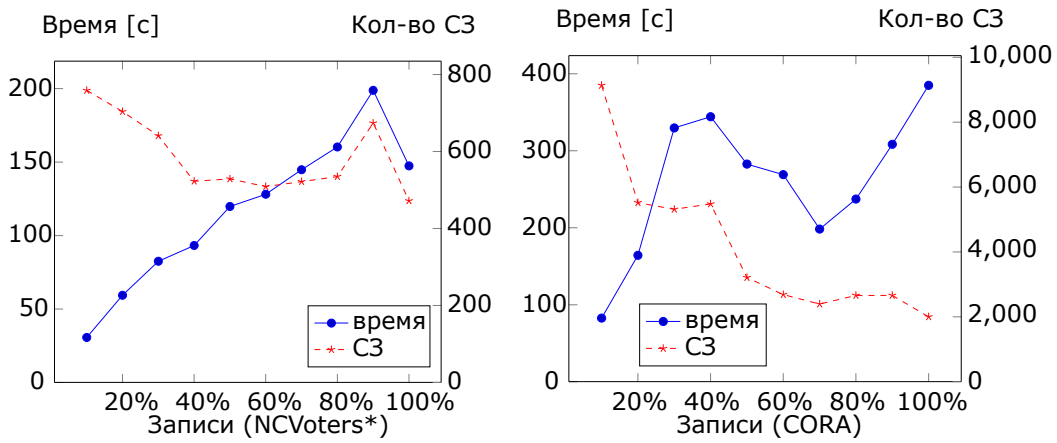

Рис. 6: Время работы при масштабировании количества записей. Источник: [2]

Для *NCVoters** видно, что время работы растёт примерно линейно. Время предварительной обработки действительно растёт квадратично, но время поиска по-прежнему превалирует, то есть в основном растут затраты на проверку кандидатов. Некоторые легко проверяемые кандидаты соблюдаются в первых выборках данных, и их исчезновение не влияет значительно на время работы, а вот исчезновение некоторых СЗ в последней выборке явно сказывается на времени работы.

Для *CORA* затраты на предварительную обработку пренебрежимо малы и составляют менее 2% времени выполнения. Время работы увеличивается (при 10-40% и 70-100% записей), когда кандидаты продвигаются выше в решётке и их становится сложнее обрабатывать, однако время работы также значительно сокращается, когда СЗ исчезают или когда их становится проще вывести из пар записей.

Таким образом, трудно предсказать время работы HyMD по количеству записей. Однако тенденция такова, что увеличение количества записей увеличивает время работы и уменьшает количество СЗ.

Количество сопоставлений столбцов. Эксперимент начинается для обоих наборов данных с поиска СЗ для двух сопоставлений столбцов, а затем добавляются новые, пока не будут использованы все. На практике количество сопоставлений столбцов увеличивается, если поиск СЗ осуществляется для наборов данных с большим количеством столбцов, а также если необходимо учитывать дополнительные функции схожести. Добавление сопоставлений столбцов может привести к появлению дополнительных СЗ, но не влияет на существующие. Решётка пространства поиска при этом растёт экспоненциально, и для большинства наборов данных число минимальных СЗ также растёт экспоненциально.

На рис. 7 показаны результаты этого эксперимента. Ожидаемо, количество СЗ растёт примерно экспоненциально с ростом числа сопоставлений столбцов для обоих наборов данных. При этом трудоёмкость предварительной обработки увеличивается линейно с каждым сопоставлением столбцов. Добавление сопоставлений столбцов в

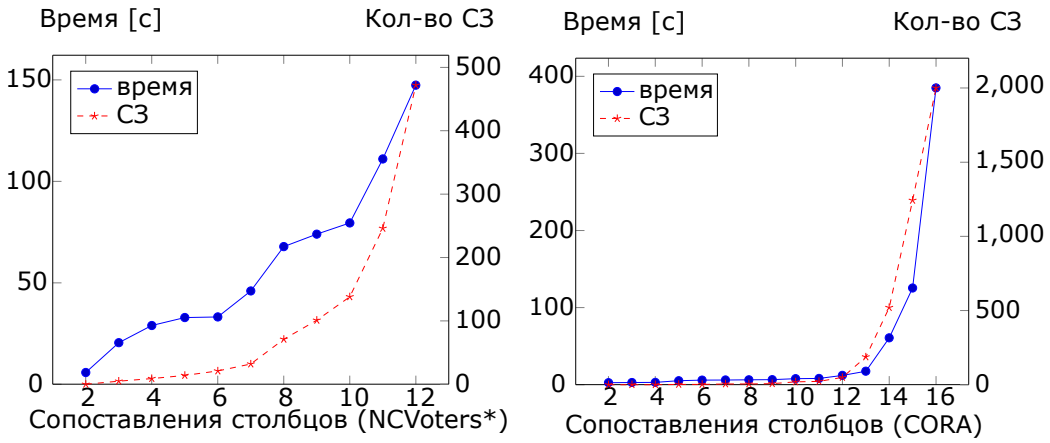

Рис. 7: Время работы при масштабировании количества сопоставлений столбцов. Источник: [2]

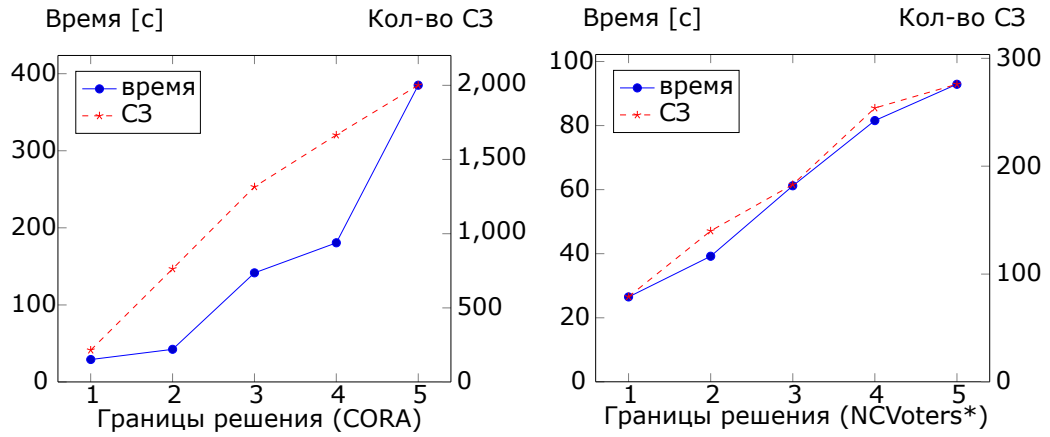

Рис. 8: Время работы при масштабировании количества границ решения. Источник: [2]

разном порядке даёт одинаковые кривые времени выполнения. Следовательно, можно ожидать, что время работы и количество СЗ будут расти экспоненциально с увеличением числа сопоставлений столбцов.

Количество границ решения. Для каждого сопоставления столбцов выбирается некоторое количество равномерно распределённых границ решения из его набора естественных границ решения. Это число линейно увеличивается от одной до пяти границ решения. При добавлении дополнительных границ решения пространство поиска растёт экспоненциально, так как, по сути, добавляется один шаг к размерности каждого атрибута.

Результаты, полученные на *CORA* (рис. 8), подтверждают это теоретическое соображение, демонстрируя в целом экспоненциальное увеличение времени работы. Хотя число минимальных СЗ растёт даже медленнее, чем линейно, пространство поиска, которое необходимо исследовать алгоритму, растёт экспоненциально. Для *NCVoters** такого экспоненциального поведения времени работы не наблюдается, по-

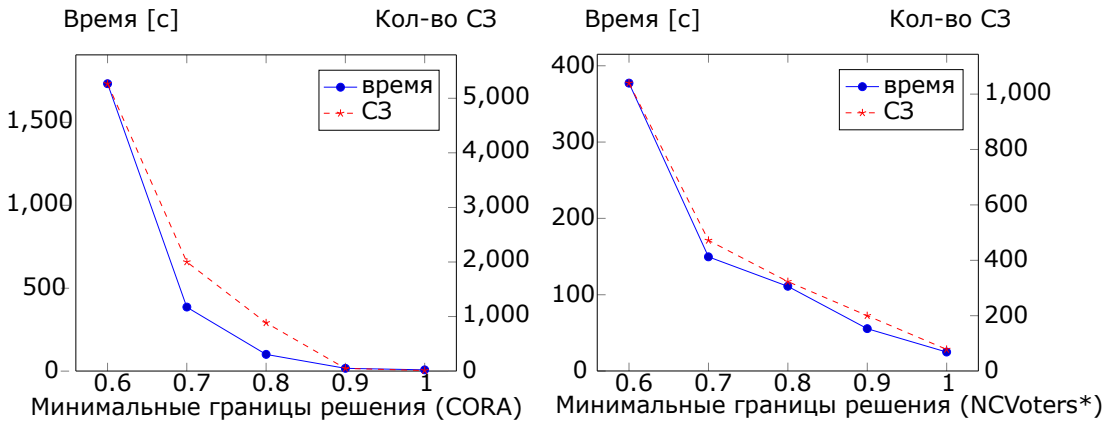

Рис. 9: Время работы при уменьшении минимальных границ решения. Источник: [2]

скольку некоторые сопоставления столбцов имеют менее пяти естественных границ решения и, следовательно, при увеличении этого параметра не увеличивают размер пространства поиска. В целом, с увеличением количества границ решения время работы сначала растёт экспоненциально, а затем сходится к времени работы при использовании всех естественных границ решения в наборе данных.

Минимальная граница решения. Минимальная граница решения — это ещё один параметр, контролирующий количество границ решения, исследуемых при поиске СЗ. Во всех предыдущих экспериментах была использована минимальная граница решения, равная 0.7, для всех сопоставлений столбцов левой и правой частей. Рассмотрение значений, совпадающих всего лишь на 70%, представляется довольно малополезным для большинства реальных сценариев использования, однако это позволяет продемонстрировать эффективность гибридного подхода к поиску. Чтобы показать эффективность работы алгоритма при более высоких минимальных границах решения, в данном эксперименте измеряется время работы HyMD для различных значений минимальных границ решения.

На рис. 9 показаны результаты для *NCVoters* и *CORA*. Видно, что для обоих наборов данных увеличение минимальной границы решения значительно сокращает не только время поиска, но и количество обнаруженных СЗ. Если говорить более конкретно, то увеличение порога с 0.7 до 0.9 уменьшает время поиска для *NCVoters* примерно в 3 раза, а для *CORA* — в 23 раза. Количество обнаруженных СЗ снижается на 57.6% и 97.5% соответственно. Таким образом, увеличение минимальных границ решения может быть использовано для значительного сокращения времени поиска и сужения области поиска до СЗ с более высокими требованиями к схожести.

6. Заключение

По итогам работы был проведён обзор статьи “Efficient Discovery of Matching Dependencies”:

- Изучены сопоставляющие зависимости;
- Рассмотрен алгоритм HyMD;
- Рассмотрено тестирование этого алгоритма;

Список литературы

[1] Papenbrock Thorsten, Bergmann Tanja, Finke Moritz, Zwiener Jakob, and Naumann Felix. Data Profiling with Metanome // Proc. VLDB Endow. — 2015. — aug. — Vol. 8, no. 12. — P. 1860–1863. — Access mode: http://dx.doi.org/10.14778/2824032.2824086.

[2] Schirmer Philipp, Papenbrock Thorsten, Koumarelas Ioannis, and Naumann Felix. Efficient Discovery of Matching Dependencies // ACM Transactions on Database Systems. — 2020. — Aug. — Vol. 45, no. 3. — P. 1–33. — Access mode: https://doi.org/10.1145/3392778.

[3] Flach Peter A. and Savnik Iztok. Database Dependency Discovery: A Machine Learning Approach // AI Commun. — 1999. — aug. — Vol. 12, no. 3. — P. 139–160.

[4] Monge Alvaro E. and Elkan Charles P. The Field Matching Problem: Algorithms and Applications // Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. — Portland, Oregon : AAAI Press. — 1996. — KDD'96. — P. 267–270.

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